



FACULDADE 7 DE SETEMBRO - FA7
CURSO DE ESPECIALIZAÇÃO EM ARQUITETURA, DESIGN E
IMPLEMENTAÇÃO DE SISTEMAS PARA INTERNET

EVANDRO CUSTÓDIO GONÇALVES

IMPLEMENTANDO AS PRÁTICAS ÁGEIS TESTES AUTOMATIZADOS E
INTEGRAÇÃO CONTÍNUA EM UMA EMPRESA DE ECONOMIA MISTA

FORTALEZA - 2016

EVANDRO CUSTÓDIO GONÇALVES

**IMPLEMENTANDO AS PRÁTICAS ÁGEIS TESTES AUTOMATIZADOS E
INTEGRAÇÃO CONTÍNUA EM UMA EMPRESA DE ECONOMIA MISTA**

Monografia apresentada à Faculdade 7 de Setembro
como requisito parcial para obtenção do título de
Especialista em Arquitetura, Design e
Implementação de Sistemas para Internet.

Orientador: Prof. Marum Simão Filho, Msc.

Fortaleza - 2016

IMPLEMENTANDO AS PRÁTICAS ÁGEIS TESTES AUTOMATIZADOS E INTEGRAÇÃO CONTÍNUA EM UMA EMPRESA DE ECONOMIA MISTA

Monografia apresentada à Faculdade 7 de Setembro
como requisito parcial para obtenção do título de
Especialista em Arquitetura, Design e
Implementação de Sistemas para Internet.

Evandro Custódio Gonçalves

Monografia apresentada em ____/____/____

Prof. Marum Simão Filho, Msc.
Orientador

1º Examinador: _____
Prof. Régis Patrick Silva Simão, Msc.

2º Examinador: _____
Prof. Ciro Carneiro Coelho, Msc.

Prof. Marum Simão Filho, Msc.
Coordenador do Curso

AGRADECIMENTOS

Agradeço primeiramente a Deus que sempre me ajudou a superar as dificuldades em todos os momentos de minha vida, a Otávio Frota, Superintendente da CAGECE, que prontamente apoiou a realização deste trabalho, aos meus colegas de trabalho Aldivone Correia e Carlos Aguiar que não mediram esforços para que fossem dadas as reais condições para a prática do estudo apresentado.

DEDICATÓRIA

Dedico este trabalho única e exclusivamente a Maria Aurea Custódio, com quem não pude compartilhar este momento tão importante da minha vida profissional e pessoal, por tudo que me ensinou como amiga, companheira, cúmplice, e, acima de tudo, uma grande mãe.

RESUMO

Um dos grandes desafios das empresas públicas que desenvolvem os seus próprios softwares é atender a uma demanda cada vez mais crescente por sistemas, sem comprometer a evolução daqueles já em utilização, produzindo soluções com qualidade e em tempo hábil. Para tanto, faz-se necessário alinhar a produção de softwares a processos e modelos sugeridos pela Engenharia de Software que melhor se adequem à realidade da organização. Este trabalho descreve como a Companhia de Água e Esgoto do Estado do Ceará – CAGECE melhorou o processo de desenvolvimento do Sistema Comercial PRAX por meio da utilização de testes automatizados e da implantação da integração contínua. São apresentadas, ainda, as intervenções realizadas nos processos existentes, o processo de testes, a arquitetura de testes implementada e a gestão de mudanças. São também fornecidos os benefícios e as dificuldades encontradas durante a implantação da Integração Contínua.

Palavras-chave: Testes automatizados, Integração contínua.

RÉSUMÉ (se for em inglês, usar Abstract)

L'un des grands défis des entreprises publiques qui développent leur propre logiciel est de répondre à une demande sans cesse croissante pour les systèmes sans compromettre le développement futur de ceux qui sont déjà en cours d'utilisation, la production de solutions de qualité et en temps voulu. À cette fin, il est nécessaire d'aligner les processus de production de logiciels et des modèles proposés par le génie logiciel le mieux adapté à la réalité de l'organisation. Ce document décrit comment Société de l'eau et égoûts de l'État du Ceará - CAGECE amélioré le processus de développement du système de PRAX commerciale grâce à l'utilisation des tests et de mise en œuvre de l'intégration continue automatisée. Sont également présentés, les interventions dans les processus existants, le processus de test, les essais mis en œuvre l'architecture et la gestion du changement. Également fourni sont les avantages et les difficultés rencontrées lors de la mise en œuvre de l'intégration continue.

Mots-clés: L'automatization des tests , L'Intégration continue

SUMÁRIO

1. INTRODUÇÃO.....	12
1.1. Motivação	12
1.2. Objetivos	14
1.2.1. Objetivo Geral	14
1.2.2. Objetivos Específicos	14
1.3. Organização do Trabalho	15
2. TESTES DE SOFTWARES	16
2.1. Conceitos de Teste.....	16
2.1.1. Erro, Falhas e Defeito	17
2.1.2. Verificação e Validação.....	18
2.2. Custo de um defeito	18
2.3. Qualidade do Software	19
2.4. Atividades de testes	20
2.5. Estratégias de testes	21
2.6. Modelo V	22
2.6.1. Teste Unitário	23
2.6.2. Teste de Integração	23
2.6.3. Teste de Sistema	24
2.6.4. Teste de Aceitação.....	24
2.7. Dimensões de Teste.....	24
2.8. Métodos de Testes	25
2.8.1. Testes Caixa-preta	25
2.8.2. Testes Caixa-branca	26
2.8.3. Testes exploratórios	26
2.8.4. Testes automatizados	27
2.9. Gestão de Defeitos	27
2.10. Gestão de Mudanças	29
2.10.1. Controle de Versão	29
2.11. Integração Contínua.....	31
2.11.1. Componentes de um sistema de Integração Contínua	32
2.12. Considerações Finais.....	34
3. AMBIENTE DO ESTUDO DE CASO	35
3.1. Descrição do Ambiente	35

3.1.1. Equipe de TI.....	36
3.2. Sistema Comercial PRAX.....	36
3.2.1. Arquitetura do PRAX.....	38
3.3. Controle de Versão.....	39
3.4. Processo de Desenvolvimento	40
3.5. Gestão de Mudanças	41
3.6. Problemas / Dificuldades	41
3.7. Considerações Finais	42
4. ESTUDO DE CASO: ESTRATÉGIA DE INTEGRAÇÃO CONTÍNUA E METODOLOGIA DE TESTES.....	43
4.1. Tecnologias utilizadas	43
4.2. Macroprocessos	44
4.2.1. Processo de Migração de Projetos	45
4.2.2. Estrutura do Projeto	48
4.2.3. Processo de Geração de Versão	50
4.2.4. Processo de Gestão de Defeitos.....	54
4.2.5. Processo de Gestão de Mudanças	56
4.2.6. Testes automatizados	58
4.2.7. Integração contínua.....	58
4.3. Implantação.....	59
4.4. Dificuldades para implantação.....	60
4.5. Benefícios.....	61
4.6. Considerações Finais	61
5. CONCLUSÃO	62
5.1. Trabalhos futuros.....	63
REFERÊNCIAS.....	64

LISTA DE FIGURAS

Figura 1 - Regra 10 de Myers.....	19
Figura 2 - Grupos de Atividades de testes	20
Figura 3 - Estratégia de teste	21
Figura 4 - Modelo V	22
Figura 5 - Teste caixa-preta	25
Figura 6 - Teste Caixa-branca.....	26
Figura 7 - Elementos chave de um Processo de Gestão de Defeitos	28
Figura 8 - Fluxo de construção de conteúdo sem uso de sistema de controle de versão	30
Figura 9 - Fluxo de construção de conteúdo com o uso de sistema de controle de versão	31
Figura 10 - Componentes de um sistema IC.....	32
Figura 11 - Sistema Comercial PRAX	37
Figura 12 - Arquitetura do PRAX.....	39
Figura 13 - Organização dos projetos no Servidor SVN.....	40
Figura 14 - Ferramentas utilizadas.....	43
Figura 15 - Macroprocessos da GETIC.....	44
Figura 16 - Processo de migração de projetos.....	46
Figura 17 - Exibindo versão da aplicação	46
Figura 18 - Estrutura do Projeto no repositório SVN	48
Figura 19 - Padrão da GETIC para realização de <i>commits</i>	49
Figura 20 - Tela do SGD contendo informações do <i>Commit</i> realizado	50
Figura 21 - Processo de Geração da Versão	50
Figura 22 - Tela do SGD para cadastro de versões	51
Figura 23 - Log do processo de construção da versão	52
Figura 24 - Testes estáticos realizados pela ferramenta SonarQube.....	53
Figura 25 - Selenium Grid realizado teste funcionais em paralelo.	54
Figura 26 - Processo de Gestão de Defeitos	55
Figura 27 - Processo de Gestão de Mudanças	56
Figura 28 - Formulário e comunicado de mudança	57
Figura 29 - Tela do Servidor Jenkins.....	59

LISTA DE ABREVIATURAS E SIGLAS

CAGECE: Companhia de Água e Esgoto do Estado do Ceará

EJB: Enterprise Java Bean

IC: Integração Contínua

MDB: Message Driven Bean

TI: Tecnologia da Informação

XP: Extreme Programming

1. INTRODUÇÃO

1.1. Motivação

Molinari (2003, p. 19) define de maneira bastante subjetiva o que seria, de fato, qualidade de software na visão daqueles que compõem equipes de trabalhos de desenvolvimento de software. É o que todos querem, desejam e esperam ao produzirem softwares. É aquilo que o software precisa ter para gerar confiança e consistência.

Proporcionar qualidade em produtos de softwares não é uma tarefa fácil quando equipes de desenvolvedores muitas vezes buscam a qualidade de maneira isolada. Cada um garante a qualidade na sua atividade de produção de software. Molinari (2003, p.19) reforça ainda que muitos esquecem que o trabalho de produção de software é uma tarefa de grupos e não de indivíduos isolados.

Segundo Hiram (2011, p.7), a crise do software nos anos 70 refletia uma série de problemas, que iam de atrasos nas entregas a custos elevados de produção, e que influenciavam diretamente na qualidade do software. Essa preocupação levou à utilização de princípios da engenharia na produção de softwares, o que, em seguida, fundamentaria a disciplina Engenharia de software.

A Engenharia de Software, segundo Pressman (2011, p.39), é o estabelecimento e a utilização dos princípios de engenharia para a obtenção de softwares de maneira econômica, confiável e eficiente.

No entanto, para a Engenharia de Software proporcionar o desenvolvimento de produtos com a qualidade esperada, é necessário que as equipes de desenvolvimento sejam efetivamente disciplinadas.

Através de processos e modelos de desenvolvimento, a engenharia orienta e ordena a produção de software de maneira racional e dentro do prazo (PRESSMAN, 2011, p.39).

A Engenharia de software nos apresenta processos e modelos que aplicam atividades de engenharia com o intuito de produzir um software de maneira ordenada visando atender aos interesses dos clientes num prazo e custo aceitável.

A importância dos métodos de desenvolvimento de software é lembrada por Hiram (2011, p.13) como sendo um canal de comunicação entre os membros de

equipes de desenvolvimento, estabelecendo padrões para a realização dos trabalhos de produção de softwares.

Corillo e Jubileu (2014, p.1) lembram ainda que, ao longo do tempo, foram criados vários modelos de processo de software que apresentam sistemáticas para o desenvolvimento de software. Dentre eles, estão o modelo em cascata, incremental, espiral, processo unificado, e outros. Esses têm um grande enfoque na documentação, o que é considerado um fator crítico para muitos analistas/desenvolvedores.

Em fevereiro de 2001, surgiu a *Agile Alliance* a partir do estudo de 17 metodologias com o intuito de melhorar o desenvolvimento. O resultado deste estudo foi o manifesto ágil, que encoraja a utilização dos melhores meios para o desenvolvimento de software (CORILLO; JUBILEU, 2014).

A partir deste manifesto, surgiram as metodologias ágeis, que potencializam a comunicação entre as pessoas e, por consequência, a minimização de documentação (SOMMERVILLE, 2011).

Cabe às equipes de desenvolvimento identificarem qual ou quais processos e modelos melhor se adequam ao processo de produção do software desejado. É possível a utilização combinada de modelos em um processo de desenvolvimento.

Para Pressman (2011, p. 87), dentre as metodologias ágeis, a *Extreme Programming* - XP é uma das abordagens com ampla utilização para o desenvolvimento de software ágil. Ainda segundo Pressman (2011, p. 87), Kent Beck foi o responsável pelas primeiras publicações sobre XP.

Comunicação, simplicidade, feedback, coragem e respeito foram os cinco valores estabelecidos por Kent Back em suas publicações, para direcionar todos os trabalhos realizados seguindo esta metodologia (PRESMAN, 2011, p.87).

Corillo e Jubileu (2014, p.3) enfatizam que XP estabelece integrações curtas para o desenvolvimento de software podendo variar de uma a três semanas e que seguem uma série de práticas, das quais é apresentada abaixo apenas uma seleção, com aquelas que estão diretamente relacionadas ao trabalho.

- O código é propriedade de todo o grupo, assim, todos os membros da equipe podem se sentir à vontade para fazer alterações no código;

- Refatorar o código sempre que necessário, ou seja, limpar o código, mantendo as funcionalidades e tornando-o mais enxuto e mais fácil de ser entendido. Todos os membros da equipe têm a liberdade de refatorar o código;
- Realizar testes de unidade considerando cada unidade do sistema;
- Realizar testes de aceitação considerando cada história de usuário;
- Criar e manter padrões de codificação para que a equipe de desenvolvimento consiga compreender todo o código do sistema;
- Integrar o código no repositório para que a equipe de desenvolvimento trabalhe sempre com a versão mais recente do sistema;

Essas práticas serão apresentadas em mais detalhes no decorrer do desenvolvimento do trabalho.

1.2. Objetivos

1.2.1. Objetivo Geral

O objetivo geral deste trabalho consiste em descrever a implementação das práticas integração contínua e testes automatizados em uma empresa de economia mista.

1.2.2. Objetivos Específicos

Os objetivos específicos deste trabalho são listados a seguir:

- Pesquisar sobre testes automatizados;
- Pesquisar sobre integração contínua;
- Propor um processo de testes de software alinhado às práticas ágeis;
- Sugerir ferramentas que auxiliem os processos de desenvolvimento e de testes;
- Definir uma arquitetura de testes automatizados;

- Simplificar o processo de entregas de versões aos clientes através da utilização da integração contínua.

1.3. Organização do Trabalho

Este trabalho está organizado em cinco capítulos, incluindo esta introdução, descritos resumidamente a seguir.

No Capítulo 2, Testes de Softwares, é apresentada uma revisão sobre os princípios de testes, processo de testes, tipos de testes, níveis de testes e técnicas de testes.

No Capítulo 3, Ambiente do Estudo de Caso, será apresentado o ambiente onde foi desenvolvido o trabalho. É demonstrada a arquitetura do sistema comercial, o processo de desenvolvimento e processo de testes existentes, como se dava a gestão de mudanças, problemas e dificuldades existentes.

No Capítulo 4, Implantação das práticas, é apresentado o conjunto de intervenções sugeridas a partir da observação do cenário demonstrado no capítulo anterior, os ajustes no processo de desenvolvimento, o processo de testes sugerido, ferramentas utilizadas, as arquiteturas de testes automatizados e de hardware e softwares que foram necessários para a implantação da integração contínua, como também as adequações na gestão de mudanças. Também serão apresentados, no final deste capítulo, os benefícios e dificuldades para adoção das práticas sugeridas.

No Capítulo 5, Conclusão, são fornecidas as conclusões e sugestões para trabalhos futuros.

2. TESTES DE SOFTWARES

Neste capítulo, o leitor fará uma imersão no mundo dos testes, passando pelos conceitos básicos, técnicas de testes, níveis de testes e tipos de testes. Entenderá o processo de gestão de mudanças, gestão de defeitos e a arquitetura por trás da integração contínua.

2.1. Conceitos de Teste

Segundo Hirama (2011, p.93), a origem dos testes vem do Latim, e o termo *testum* significa pote de barro usado em ensaios como metais cujo objetivo era provar a existência dos mesmos ou medir massa de diversos elementos. Portanto, colocar um software em testes significa provar a existência de defeitos levando em consideração diversos aspectos, sejam eles estruturais, lógicos ou sistêmicos.

Para Sommerville (2011, p. 145), o teste consiste em uma atividade capaz de mostrar que um programa, antes do seu uso, faz aquilo que de fato se propõe a fazer, permitindo assim, a descoberta antecipada de defeitos. Ao contrário do que muitos pensam sobre o propósito da realização de testes, eles não provam que o sistema está livre de defeitos, mas sim, demonstram que terá o comportamento esperado em qualquer situação.

De acordo com Pressman (2011, p. 402), as atividades de testes precisam de planejamento. Sua execução é sistemática e definida a partir de técnicas e modelos de testes em um processo de software.

Segundo Hirama (2011, p. 94), o atendimento dos objetivos de testes com um custo esperado, os conceitos, estratégias, técnicas e métricas de testes devem estar alinhadas em um processo de testes bem definido e controlado.

Molinari (2003, p. 28) cita o teste de software como sendo a estratégia de gerenciamento de riscos mais popular, mas reforça que nem todos os defeitos são encontrados durante os testes.

É comum que as pessoas tenham uma visão do processo de teste de que ele consiste apenas da fase de execução, como executar o programa. Esta, na verdade, é uma parte do teste, mas não engloba todas as atividades do teste.

Planejamento e controle, escolha das condições de teste, modelagem dos casos de teste, checagem dos resultados, avaliação do critério de conclusão, geração de relatórios sobre o processo de teste e sobre sistema alvo e encerramento ou conclusão são atividades de teste que acontecem antes e depois da fase de execução. A revisão de documentos (incluindo o código fonte) e análise estática também estão incluídas dentre as atividades de testes (THOMAS MULLER, 2011).

2.1.1. Erro, Falhas e Defeito

Molinari (2003, p. 54) define falha (bug) como sendo qualquer problema que venha ocorrer no software, com exceção dos problemas causados pelo usuário oriundos da má utilização do mesmo. Porém, nem sempre uma falha é um defeito, já que, pela sua definição, é uma não-conformidade em relação ao que o software se propõe a fazer.

Quando um software apresenta comportamento diferente do esperado pelo cliente, nem sempre é um defeito, pois, para ser considerado como tal, é necessário que suas características e comportamentos tenham sido bem definidos através dos seus requisitos.

Thomas Muller (2011, p. 11) reforça que os erros são cometidos pelos homens, gerando defeitos no código, no software ou sistema ou em documentos (requisitos). Uma vez executado um código com defeito, o sistema certamente falhará ao tentar realizar o que se propõe, causando, então, uma falha.

As pressões do dia a dia, códigos complexos, mudanças de tecnologias e/ou outras interações do homem com o sistema são os principais fatores que fazem com que o homem cometa falhas.

As falhas não se limitam só a erros humanos, Thomas Muller (2011, p. 11) cita os fatores ambientais como radiação, magnetismo, campos eletrônicos e poluição como causadores de muitas falhas em softwares embarcados (*firmwares*) ou influenciar a execução de softwares.

2.1.2. Verificação e Validação

Molinari (2003, p. 23) define verificação como a técnica responsável por provar que um produto foi construído de maneira correta, ou seja, alinhado com as especificações das atividades de desenvolvimento.

Já validação, de acordo com Molinari (2003, p.23), é a técnica que vai de encontro com as necessidades dos clientes, ou seja, com as especificações do consumidor.

Hirama (2011, p. 108) apresenta a Verificação e Validação como o processo com o intuito de determinar se os requisitos de sistemas ou componentes estão completos e corretos, se em cada fase de desenvolvimento, os produtos gerados atendem aos requisitos e às condições impostas pela fase anterior e se, ao final, o sistema ou componente está aderente às especificações.

Segundo Luppi e Nogueira (2011, p. 271), teste de software é o processo de execução controlada do software com o intuito de avaliar se o mesmo se comporta conforme a especificação.

Pressman (2011, p. 402) considera o teste de softwares algo muito amplo, porém, lembra que, ao falar de teste de software há de início uma referência imediata aos termos verificação e validação.

Essa afirmação é confirmada por Molinari (2003, p. 23), quando ele diz que essa referência é algo tradicional, considerado ainda o teste de software um processo de validação como parte do ciclo de desenvolvimento do produto, porém, enfatiza que o teste de um produto está bem mais perto de validação do que verificação.

2.2. Custo de um defeito

Roseli Pinheiro (2014, p. 24) lembra que em 1979, Myers, escritor do livro “A arte do teste de softwares”, introduziu a regra 10 de Myers, um dos conceitos mais importantes relacionados a testes de softwares: A Figura 1 logo em seguida, mostra o custo de descobrimos e corrigimos um defeito durante as fases de desenvolvimento do software, mostrando que quanto mais cedo, menor é seu custo para o projeto.

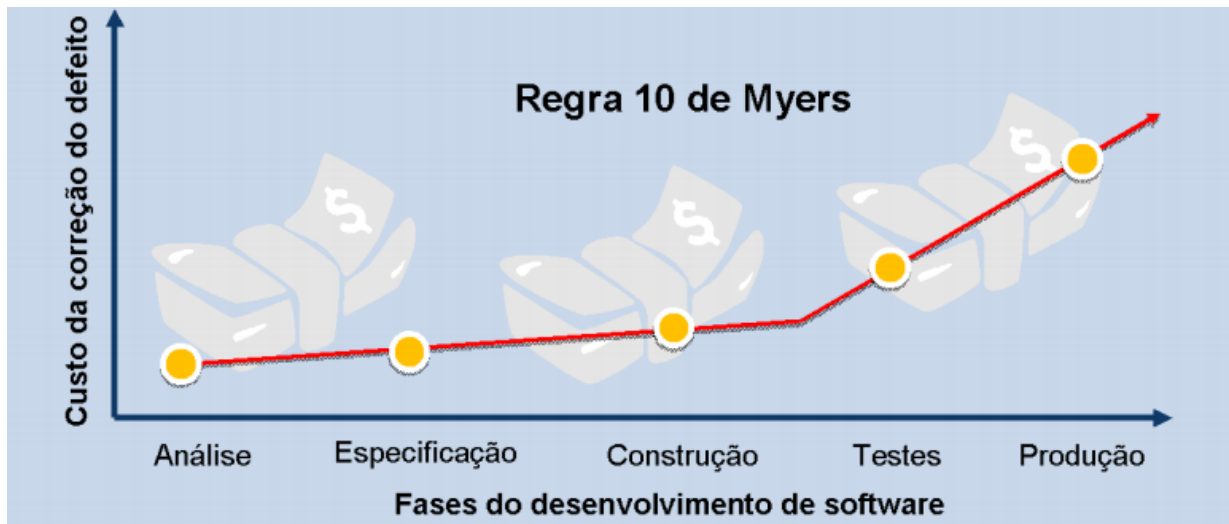


Figura 1 - Regra 10 de Myers

Fonte: Roseli Pinheiro (2014, p.24)

Defeitos encontrados nas fases iniciais da etapa de desenvolvimento do software são mais baratos de serem corrigidos do que aqueles encontrados na etapa de produção.

Molinari (2011, p. 56) exemplifica que, caso um defeito seja descoberto no início (mais próximo à fase de especificação), custa cerca de \$ 0,10, e quando descoberto tardiamente (o mais próximo do release), custa \$ 100,00 ou mais, estabelecendo na prática um custo na ordem de 10^n .

2.3. Qualidade do Software

Um ponto importante é lembrado por Roseli Pinheiro (2014, p. 29) quando se trata de qualidade de software. Para muitos, teste é igual a qualidade de software, ou mais ainda, software testado é sinônimo de software com qualidade.

“Qualidade (*quality*) é o grau com o qual um componente, sistema ou processo atende aos requisitos especificados e/ou necessidades e expectativas do usuário ou consumidor.” Roseli Pinheiro (2014, p. 29)

Roseli Pinheiro (2014) lembra ainda que a partir do conceito de qualidade apresentado, pode-se afirmar:

- O teste por si só não constrói a qualidade do software;

- O teste tem a função de ajudar a medir a qualidade;
- O teste pode fornecer confiança na qualidade de software;
- Testes bem projetados e executados reduzem os riscos de falhas;

2.4. Atividades de testes

A execução é a parte mais visível do teste. Mas, para se obter eficácia e eficiência, os planos de teste precisam conter o tempo a ser gasto no planejamento dos testes, modelagem dos casos de testes e preparação da execução e avaliação de resultados.

Thomas Muller (2011, p. 15) apresenta as atividades do processo de teste básico como sendo: Planejamento e Controle, Análise e Modelagem, Implementação e execução, Avaliação dos critérios de saídas e relatórios e Atividades de encerramento de teste, que mesmo sendo apresentadas sequencialmente, as atividades durante o processo podem sobrepor-se ou acontecer de forma concorrente.

Para Roseli Pinheiro (2014, p. 44), se pudéssemos delimitar fronteiras entre as atividades de testes, existiriam dois grandes conjuntos de atividades: testes estáticos e testes dinâmicos.

As atividades de cada grupo são apresentadas na Figura 2 adiante:

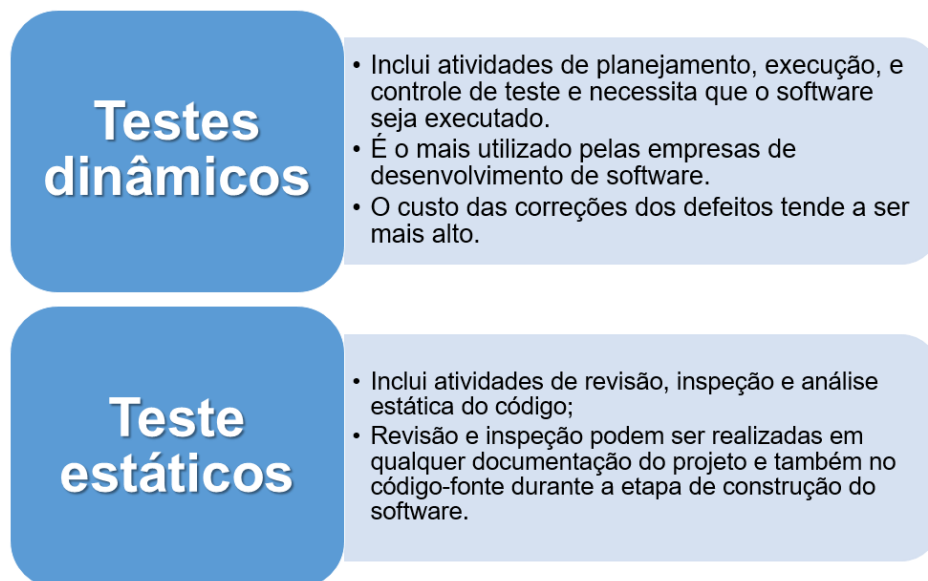


Figura 2 - Grupos de Atividades de testes

Fonte: Roseli Pinheiro (2014, p. 45).

Não existe teste isolado. A atividade de teste está intimamente relacionada com as atividades de desenvolvimento do software. Modelos de ciclo de vida de desenvolvimento diferentes necessitam de abordagens diferentes para testar.

2.5. Estratégias de testes

Pressman (2011, p. 404) declara que o processo de testes pode se assemelhar a uma espiral. À medida que há um deslocamento para o interior da espiral, há uma diminuição da granularidade do teste, havendo a necessidade de utilização de estratégias diferentes para cada nível.

As estratégias de testes são apresentadas na Figura 3 a seguir.



Figura 3 - Estratégia de teste

Fonte: Pressman (2011, p.404)

No centro da estratégia espiral estão os testes de unidade, cujo foco é justamente o código fonte. À medida que o teste se move de dentro para fora na espiral, há uma expansão dos objetivos de testes.

Em um nível acima dos testes unitários, estão os testes de integração, focados, principalmente, no projeto e arquitetura do software em teste. No próximo nível, estão os testes de validação, onde são confrontados os resultados produzidos com os requisitos estabelecidos, de forma a validar se o que vai ser entregue ao cliente está de acordo com o projetado. Em um nível mais externo, estão os testes

de sistema, focados em testar o software produzido e os demais componentes que compõem a solução desenvolvida.

2.6. Modelo V

Segundo Roseli Pinheiro (2014, p. 93), os testes de software não são atividades isoladas e estão diretamente ligados ao processo de desenvolvimento de software, ou seja, fazem parte do ciclo de vida de um projeto de software.

A Figura 4 a seguir apresenta o modelo V, que preconiza que as atividades de teste devem ocorrer durante todo o ciclo de vida de desenvolvimento, para conseguir uma detecção adiantada de erros. Ao final de cada etapa do processo de desenvolvimento, os produtos gerados devem ser avaliados, verificados e validados.

Desta forma, cada etapa produz algo que pode ser testado, sem a necessidade de que todas etapas sejam percorridas, permitindo, assim, uma antecipação no processo de teste.

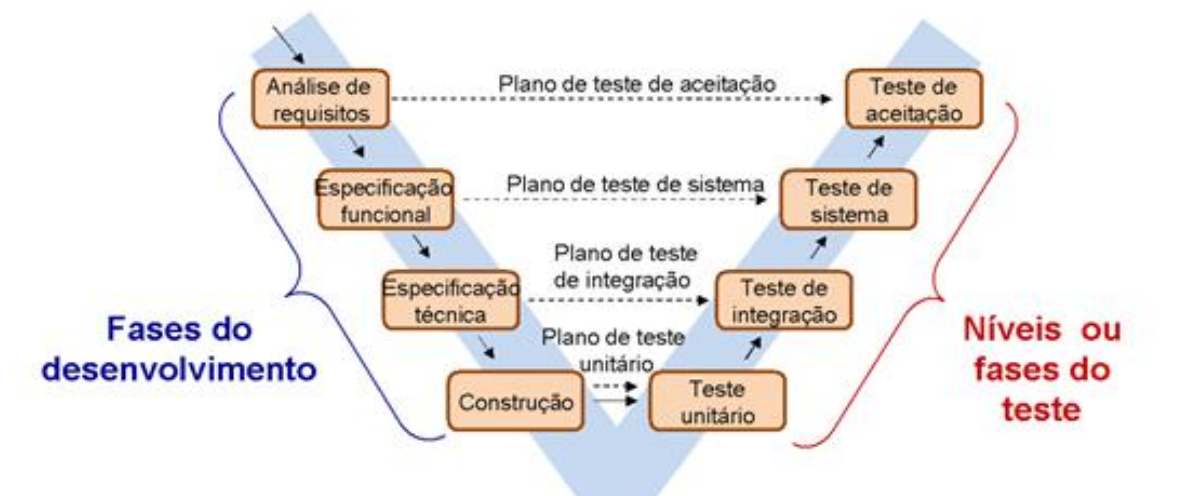


Figura 4 - Modelo V

Fonte: Roseli Pinheiro (2014, p. 97)

A seguir serão apresentados os níveis de teste sugeridos pelo Modelo V.

2.6.1. Teste Unitário

Teste unitário é o primeiro nível de teste, que inicia durante e após a fase de construção do software. São realizados pelos programadores (ROSELI PINHEIRO, 2014).

Segundo Luppi e Nogueira (2011, p. 272) o objetivo de um teste unitário é verificar um elemento logicamente tratado como uma unidade de implementação.

Para Thomas Muller (2011, p. 23), o teste unitário procura defeitos e verifica o funcionamento do software (ex.: módulos, programas, objetos, classes, etc.), que são testáveis separadamente. Geralmente, é realizado de maneira isolada do resto do sistema, dependendo do contexto do ciclo de desenvolvimento e do sistema. É comum o uso de Controladores (*“drivers”*) e simuladores (*“stubs”*) durante a realização deste tipo de teste.

Teste unitário pode também ser realizado para validar características não funcionais do software, como comportamento dos recursos (ex.: falta de memória) e testes de robustez, além de cobertura de código.

À medida que são construídas e integradas pequenas partes do código, são executados testes de componente até que eles passem. Caso ocorram erros, as correções são implementadas pelo desenvolvedor diretamente e testadas novamente logo em seguida.

2.6.2. Teste de Integração

Para Pressman (2011, p. 409), o objetivo do teste de integração é verificar se os componentes antes testados isoladamente, ao serem combinados, funcionarão como o desejado. São testadas as interfaces compartilhadas entre os diversos componentes do sistema, para garantir que tudo se comportará da maneira esperada.

A afirmação acima é reafirmada por Luppi e Nogueira (2011, p. 272) quando apresenta a verificação das interfaces entre as partes de uma arquitetura de software como sendo o objetivo do teste de integração.

2.6.3. Teste de Sistema

Pressman (2011, p. 419) define teste de sistema com uma série de diferentes testes com a finalidade primordial de exercitar totalmente o sistema. A realização de diferentes testes funciona como uma verificação de que todos os elementos do sistema estão adequadamente integrados e em pleno funcionamento.

Luppi e Nogueira (2011, p. 272) reafirmam que o teste de sistema consiste no teste do sistema computacional como um todo.

2.6.4. Teste de Aceitação

A validação se o produto atende aos requisitos especificados fica a cargo do teste de aceitação. Este teste determina a aceitação do usuário mediante a confirmação de que o sistema faz aquilo que de fato deve fazer (MOLINARI, 2011).

Os testes de aceitação deverão ser realizados em um ambiente muito semelhante ao ambiente de produção. Podem ser divididos em testes funcionais e não funcionais. Podem ser testes manuais ou automatizados (LUPPI e NOGUEIRA, 2011).

Pressman (2011, p. 419) cita, dentre os testes que podem fazer parte do conjunto de testes de aceitação, o teste de segurança, teste por esforços, teste de recuperação, teste de desempenho, teste de disponibilidade, dentre outros.

2.7. Dimensões de Teste

Segundo Molinari (2011, p. 59) os tipos de testes (teste de unidade, teste de integração, teste de sistema e teste de aceitação) fazem parte da primeira dimensão de testes, a qual ele denomina de Estado do Teste (“o momento”).

Molinari (2011) sugere ainda três outras dimensões:

- Segunda dimensão: Técnicas do teste (“Como vou testar”): compreende aos testes operacional, negativo-positivo, regressão, caixa-preta, caixa-branca, beta e verificação de versão;
- Terceira dimensão: Metas do teste (“o que tenho que testar”): compreende aos testes funcional, não-funcional, performance, carga, e aceitação do

usuário, estresse, volume, configuração, instalação, documentação, integridade, segurança e integridade;

- Quarta dimensão: Onde será o teste (“O ambiente de teste): compreende aos testes de aplicações mainframe, aplicações cliente, aplicações server, aplicações network, aplicações web.

2.8. Métodos de Testes

De acordo com Luppi e Nogueira (2011, p. 272), basicamente, os testes podem ser construídos de duas maneiras:

2.8.1. Testes Caixa-preta

O componente de software é testado como se fosse uma caixa-preta, sem levar em consideração o comportamento interno do mesmo. Dessa forma, o analista de teste não precisa conhecer como, em nível de código, o sistema foi construído. O foco se volta para as entradas e para as saídas do teste. Compara-se a saída com o resultado esperado. Os testes de aceitação e regressão são geralmente testes caixa-preta.

A Figura 5 apresentada abaixo, demonstra como podemos visualizar um teste de caixa-preta. Não visualizamos o que há dentro da caixa (“o como”), mas, nos preocupamos apenas com as entradas e saídas do processo.

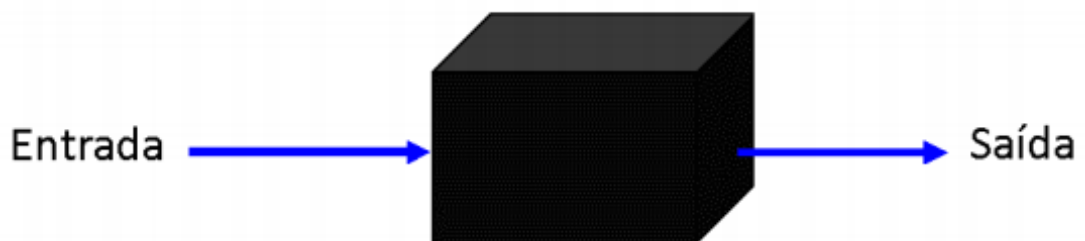


Figura 5 - Teste caixa-preta

Fonte: Roseli Pinheiro (2014, p. 107)

As principais técnicas utilizadas durante os testes caixa-preta são: partição equivalência, análise do valor limite, tabelas de decisão, diagrama de transição de teste, teste por caso de uso (ROSELI PINHEIRO, 2014).

2.8.2. Testes Caixa-branca

O componente de software é testado levando-se em consideração o código fonte. São considerados os possíveis caminhos de execução, realizando, assim, a cobertura dos possíveis fluxos. Os testes unitários são exemplos clássicos de testes caixa-branca.

A Figura 6 apresentada em seguida, demonstra como podemos visualizar um teste de caixa-branca. Visualizamos o que há dentro da caixa (“o como”) e nos preocupamos apenas com as entradas e saídas do processo.

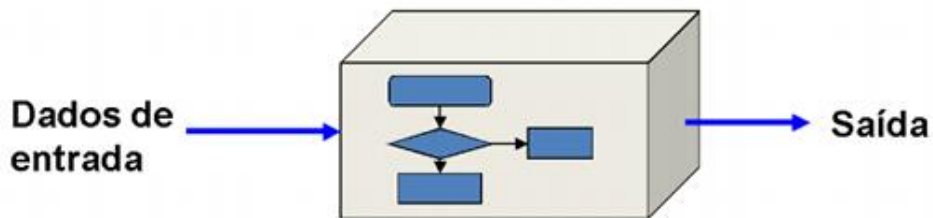


Figura 6 - Teste Caixa-branca

Fonte: Roseli Pinheiro (2014, p. 109)

Dentre as principais técnicas utilizadas durante os testes caixa-branca podem-se citar: teste de sentença/comando e teste de decisão. (ROSELI PINHEIRO, 2014).

2.8.3. Testes exploratórios

Os testes exploratórios baseiam-se no conhecimento do analista de testes, que se utiliza da sua experiência e intuição para adivinhar onde os erros podem ocorrer.

“Os testes baseados na experiência são testes derivados da intuição e conhecimento dos testadores através de sua experiência em aplicações e tecnologia similares. Quando usado para aumentar a técnica sistemática, testes intuitivos podem ser úteis para identificar testes específicos que não são facilmente identificados pelas técnicas formais, especialmente quando aplicado após ter estabelecido o processo mais formal.” (THOMAS MULLER, 2011, p. 41).

Uma ampla variedade e graus de eficiência desta técnica podem ser produzidos, o que vai depender do quão experiente é testador.

Segundo Roseli Pinheiro (2014, p. 92), os testes exploratórios não devem ser realizados como uma única forma de teste. Eles deverão complementar outras técnicas formais.

Dentre as principais técnicas utilizada durante os testes exploratórios citam-se: adivinhação ou suposição e exploração (ROSELI PINHEIRO, 2014).

2.8.4. Testes automatizados

Teste automatizado é uma das práticas ágeis preconizadas pela *Extreme Programming* – XP e é caracterizado pela construção de *scripts* que podem ser facilmente executados de maneira automatizada.

Para Aniche (2012, p.3), a automatização dos testes é uma maneira para conseguir testar o sistema todo de maneira constante e contínua a um preço justo. Ou seja, escrevendo um programa que testa o seu programa, o programa testador se responsabiliza por invocar os comportamentos do sistema e verifica se as saídas serão sempre as esperadas.

Crispin e Gregory (2009, p.258) explicam que existem várias razões para se automatizar um teste. Dentre elas, eles citam:

- Testes manuais são demorados;
- Processos manuais são susceptíveis a erros;
- Os testes automatizados livres de intervenções humanas são melhores;
- Testes automatizados dão feedback com cedo;
- Testes de regressão são potencializados pela automatização, etc.

2.9. Gestão de Defeitos

Segundo Bastos (2007, p. 188) apud Bennertz (2011, p. 17) a gestão de defeitos tem como objetivo principal evitar o aparecimento de defeitos e pode ser realizada

através da utilização de ferramentas computacionais pelas equipes de qualidade e de desenvolvimento.

Um processo de gestão de defeitos bem definido é composto por atividades de prevenção, uma linha de base, identificação de defeitos, solução de defeitos, melhoria do processo e relatórios de gestão de defeitos.

A Figura 7 a seguir, mostra os elementos chave de um Processo de Gestão de Defeitos.

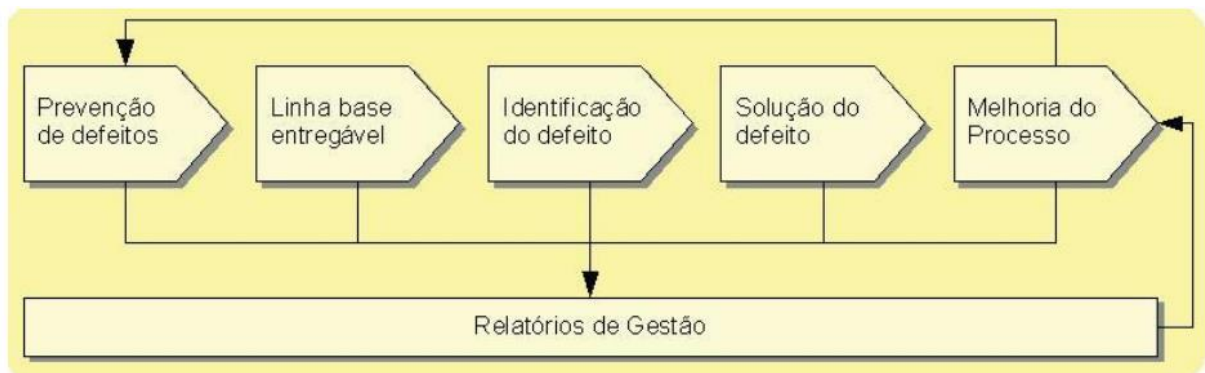


Figura 7 - Elementos chave de um Processo de Gestão de Defeitos

Fonte: Bennertz (2011, p. 18)

Os elementos que compõem o processo de gestão de defeitos são apresentados abaixo (BENNERTZ, 2011).

- **Prevenção de Defeitos:** a identificação dos riscos críticos e a estimativa dos impactos esperados são as atividades de maior importância a serem realizadas para a prevenção de defeitos;
- **Linha de Base:** É o elemento que define um ou mais marcos do projeto com o objetivo de monitorar e controlar as alterações realizadas (LENZI, 2012);
- **Identificação de Defeito:** Durante o desenvolvimento, é necessário que haja o reconhecimento do defeito. Defeitos geralmente são relatados por usuários, pela equipe de teste e, em seguida, devem ser aceitos pelos desenvolvedores. Aceitar o defeito significa reconhecer a sua existência;
- **Correção dos defeitos:** Uma vez aceito o defeito, é hora de dedicar-se à realização da melhoria do processo, no qual se avalia o processo que

originou o defeito e o que o causou. Se for pertinente, o processo pode ser alterado para minimizar ou eliminar a causa do defeito;

- Elaboração dos relatórios de defeitos: é a atividade final do processo de gestão de defeitos. Os relatórios têm por finalidade listar todos os desvios encontrados durante o processo de teste. As informações que compõem os relatórios serão utilizadas para a realização das medições de qualidade.

2.10. Gestão de Mudanças

Segundo Martins (2008, p. 8), mudanças fazem parte da vida de todos. Em TI, as mudanças são de fundamental importância para a garantia das atualizações e correções dos sistemas de informação. As mudanças podem envolver hardware, equipamentos de comunicação, software, sistemas operacionais e toda documentação associada ao funcionamento, suporte e manutenção dos sistemas em produção.

O tópico a seguir aborda o mecanismo básico e necessário para a gestão de mudanças durante o desenvolvimento de sistemas.

2.10.1. Controle de Versão

Somasundaram (2013, p. 8) define Controle de Versão como sendo um sistema capaz de gravar as alterações feitas em um arquivo ou um conjunto de arquivos durante um período de tempo, de tal forma que nos permita recuperar uma versão específica destes a qualquer momento futuro.

Uma explicação mais formal define um sistema de controle de versão como um pacote de software, responsável pelo monitoramento de mudanças em arquivos, permitindo ao usuário marcar as mudanças em diferentes níveis, com o intuito de facilitar possíveis reversões sempre que necessário.

Ao longo do desenvolvimento, mudanças ocorrem, e muitas vezes se faz necessário a recuperação de versões anteriores. Desta forma, o controle de versão se apresenta como uma solução básica e importante quando se deseja realizar uma gestão eficiente de artefatos produzidos durante a construção de um produto de software.

Somasudaram (2013, p. 10) apresenta a seguir a Figura 8, mostrando o fluxo de criação de conteúdo em diferentes momentos. Como se pode observar, o fluxo da construção do conteúdo se dá da esquerda para a direita. Nesta abordagem, não é possível voltar o conteúdo a um momento anterior.

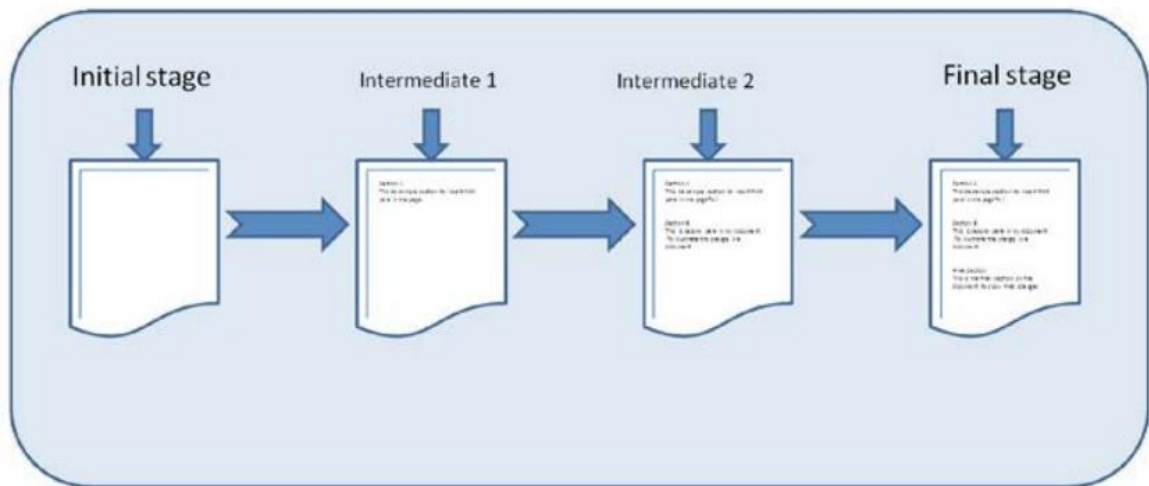


Figura 8 - Fluxo de construção de conteúdo sem uso de sistema de controle de versão

Fonte: Somasundaram (2013, p. 10)

Uma saída sugerida pelo citado autor para a problemática apresentada anteriormente é o uso de cópias do conteúdo.

Contrastando com o fluxo anterior, Somasundaram (2013, p.11) apresenta, na Figura 9 adiante, o fluxo da geração de conteúdo com a utilização de um sistema de controle de versão.

Claramente pode se perceber que há possibilidade de retroceder em versões do conteúdo sem a perda de informações, uma vez que as mudanças fazem parte do histórico mantido pelo sistema.

Os sistemas de controles podem ser classificados de acordo com o seu modo de operação, podendo ser de três tipos:

- Local,
- Centralizado, e,
- Distribuídos.

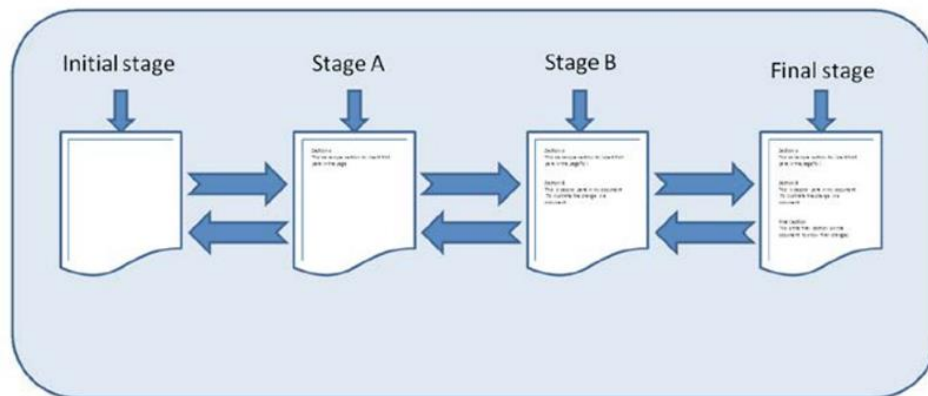


Figura 9 - Fluxo de construção de conteúdo com o uso de sistema de controle de versão

Fonte: Somasundaram (2013, p. 11)

Um sistema de controle de versão oferece uma série de operações que possibilitam a sua utilização. Existem diversos sistemas de controles disponíveis no mercado, dentre os quais podemos citar os sistemas: CVS, SVN, Git, etc.

2.11. Integração Contínua

Smart (2010, p. 36) ressalta que a Integração Contínua, também conhecida sob o termo IC, é um dos pilares de desenvolvimento de software na atualidade e que quando a Integração Contínua é implementada em uma organização, é tido como um marco por proporcionar uma mudança radical na maneira como as equipes lidam com o processo de desenvolvimento. Ela é capaz de ajudar e induzir uma série de melhorias no processo de entrega regular impulsionando até construção contínua e automatizada.

Uma boa infraestrutura de IC pode agilizar o processo de desenvolvimento até a implantação, ajuda detectar e corrigir bugs mais rapidamente, fornece um painel de controle muito útil para desenvolvedores, mas também para não-desenvolvedores, e levada ao extremo, permite que as equipes possam fornecer mais valor do negócio para os usuários finais.

A Integração Contínua, em sua forma mais simples, consiste em uma ferramenta que monitora alterações de código em seu gerenciador de configuração. Uma vez que é detectada uma alteração, esta ferramenta irá automaticamente

compilar e testar sua aplicação. Se algum erro acontecer, então a ferramenta irá imediatamente notificar os desenvolvedores para que eles possam corrigir o problema.

A Integração Contínua vai bem mais além, podendo acompanhar um projeto, monitorando as métricas de qualidade e cobertura de código e, assim, ajudar a reduzir custos de manutenção.

Tornar visível publicamente métricas de qualidade de código incentiva os desenvolvedores a terem orgulho da qualidade do seu código e tentar melhorar sempre SMART (2010).

Combinado com uma sequência de testes de aceitação automatizada, a IC pode também se transformar em ferramenta de comunicação, exibindo uma imagem clara do estado de evolução da aplicação em desenvolvimento.

2.11.1. Componentes de um sistema de Integração Contínua

Duvall (2007, p. 40) apresenta os componentes básicos de um sistema de integração contínua. Os componentes estão demonstrados na Figura 10 e descritos em seguida.

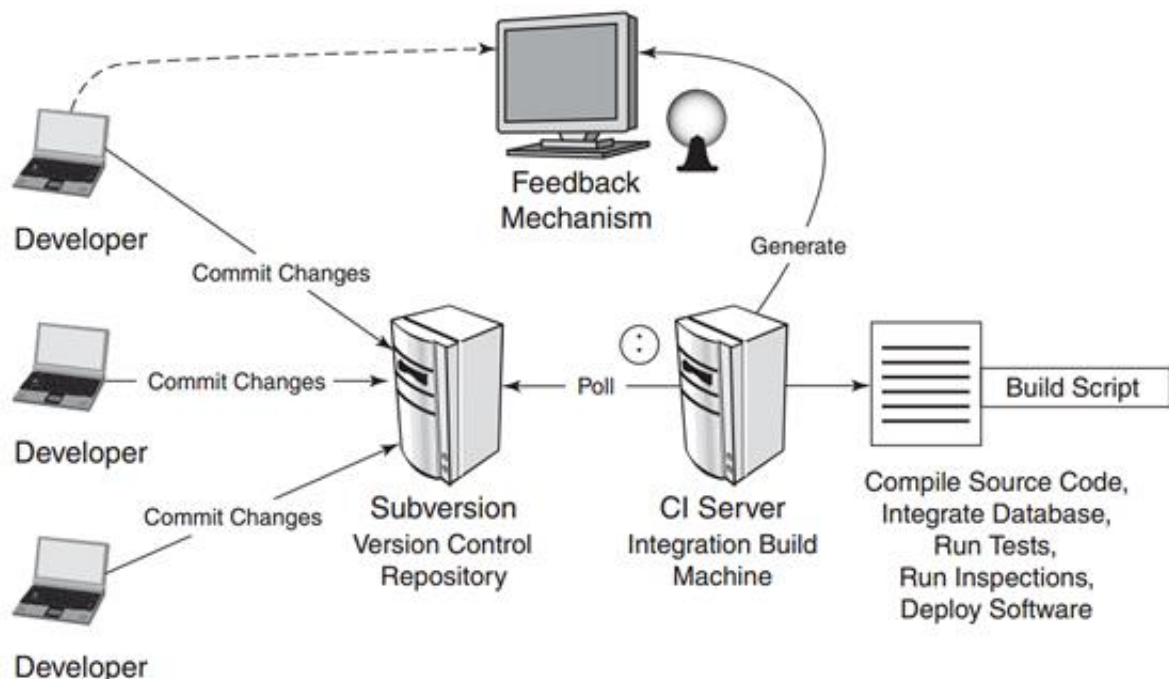


Figura 10 - Componentes de um sistema IC

Fonte: Duvall(2007, p.40)

- Desenvolvedor (Developer): São os responsáveis pelas modificações enviadas ao repositório do sistema de controle de versão, mediante a execução de atividades de desenvolvimento. Inicialmente os desenvolvedores geram versões individuais (locais) e depois compartilham com os demais desenvolvedores. Não existe integração sem que haja mudanças enviadas ao sistema de controle de versão.
- Repositório do Sistema de Controle de Versão (Version Control Repository): É o componente responsável pela gestão de mudanças no código ou outros artefatos, usando um controle de acesso ao repositório. Ele provê um ponto central para o código fonte, servindo mais adiante como um local primário para a geração de versões.
- Servidor IC (CI Server): É o componente do sistema responsável pela construção da aplicação a partir da integração de todas as mudanças enviadas ao repositório do controle de versão. O servidor recupera o código fonte do repositório e executa os scripts de construção da aplicação.
- Script de construção (Build Script): É o componente do sistema responsável por dizer ao Servidor IC como compilar, construir, testar, inspecionar e publicar a aplicação. Este componente pode ser simplesmente um script ou até mesmo um conjunto de scripts relacionados.
- Mecanismo de Feedback (Feedback Mechanism): É um dos principais componentes do sistema, pois produz informações referentes ao andamento das atividades do servidor, gerando um retorno de todo o processo, relatando problemas o quanto antes. Este feedback pode ser dado através do envio de e-mail, SMS (Short Message Service), etc.

A essência de um sistema IC é fornecer *feedback* em tempo hábil para os desenvolvedores e interessados no projeto, disponibilizando resultados rápidos e significativos.

Um sistema de integração contínua pode incorporar diversas atividades, como integração de banco de dados, testes, inspeções, implantação e *feedback*. O sistema deve ser adequado à realidade de cada empresa e às necessidades de cada projeto.

2.12. Considerações Finais

Este capítulo forneceu uma breve explanação sobre a área de testes, fornecendo uma visão macro dos conceitos primordiais para a compreensão do estudo a ser detalhado nos capítulos seguintes.

Conhecer os tipos, níveis e estratégias de testes e o conceito de IC permitirão ao leitor o entendimento do ponto principal deste trabalho, que é a apresentação da arquitetura do sistema de integração contínua e a utilização de testes automatizados pela empresa alvo.

3. AMBIENTE DO ESTUDO DE CASO

Neste capítulo, será apresentado o ambiente onde foi desenvolvido o estudo de caso, mostrando, de maneira sucinta, a arquitetura do sistema selecionado no estudo, a composição da equipe de desenvolvimento, os processos de desenvolvimento e de teste adotados, a estratégia de controle de versão antes utilizada e o processo de gestão de mudanças existente. Por fim, os problemas e os desafios identificados para implantação das práticas selecionadas serão fornecidos.

3.1. Descrição do Ambiente

A Companhia de Água e Esgoto do Estado do Ceará – CAGECE é uma empresa de economia mista, vinculada à Secretaria das Cidades do Governo do Estado do Ceará, que atua em 151 municípios, provendo serviços de água tratada para mais de 5 milhões de pessoas, correspondendo a 98,62% da população, e em 73 municípios, provendo serviço de esgotamento sanitário, correspondendo a 40,09% da população em todo o estado.

A companhia tem uma estrutura descentralizada que oferece à população serviços de atendimento, manutenção das redes de água e esgoto, faturamento, arrecadação, dentre outros serviços. Fazem parte da empresa as seguintes estruturas:

- 15 Unidades de Negócio (UN): São unidades responsáveis pela manutenção da rede água e esgoto, faturamento e arrecadação, e estão distribuídas de forma setorial e estratégica em todo o estado;
- 37 Unidades de Serviço (US): São unidades especializadas nas áreas de faturamento e arrecadação ou ainda das áreas contábil e logística, que prestam serviços às UN;
- 20 Lojas de Atendimento: São pontos de atendimento presencial onde a população pode solicitar serviços;
- 189 Núcleos de Atendimento: São pontos de atendimento presencial que oferece serviços à população em localidades do interior;
- 1 Central de atendimento: É um ponto de atendimento 24 horas que oferece serviços à população através uma central telefônica 0800.

3.1.1. Equipe de TI

A CAGECE tem uma Gerência de Tecnologia da Informação e Comunicação – GETIC, que é composta por 11 equipes, com um total de 70 profissionais, entre funcionários de carreira, terceirizados e estagiários, distribuídos entre as áreas de segurança da informação, desenvolvimento de sistemas, *Business Intelligence* - BI, infraestrutura e suporte técnico.

A GETIC é responsável por cerca de 60 sistemas, entre aqueles em plena utilização e outros em desenvolvimento, que estão distribuídos entre 8 equipes: Comercial, Administrativa, Operacional, Informações Gerenciais, Georreferenciamento, Web, ERP, ERP Operacional.

A Comercial é a equipe composta pelo maior número de profissionais. Conta atualmente com 16 profissionais, sendo subdivididos em gerência, análise, desenvolvimento e testes. O principal sistema mantido pela equipe é o Sistema Comercial PRAX, que será apresentado em seguida.

3.2. Sistema Comercial PRAX

O PRAX é o sistema comercial responsável pelas principais atividades executadas pelas UN e US. Possui 6 módulos: Cadastro, Atendimento, Faturamento, Arrecadação, Cobrança e Encerramento e Contabilização. O PRAX tem ainda um conjunto de sistemas auxiliares, desenvolvidos em plataformas diversas, além de estar integrado com uma série de outros sistemas externos.

O sistema é, diariamente, utilizado por cerca de mil usuários, através das estruturas de rede da companhia e/ou via internet pelos clientes e empresas parceiras, em busca de serviços online, sejam estas solicitações de serviços através do Portal de Atendimento, ou ainda por aplicativos mobile (Android e IOs).

O PRAX serviu de base para a implantação das práticas de testes automatizados e integração contínua, por ser um dos sistemas mais críticos da companhia e pela ampla integração com outras soluções.

A Figura 11 apresentada logo em seguida traz um esboço do relacionamento do PRAX com os seus sistemas auxiliares.

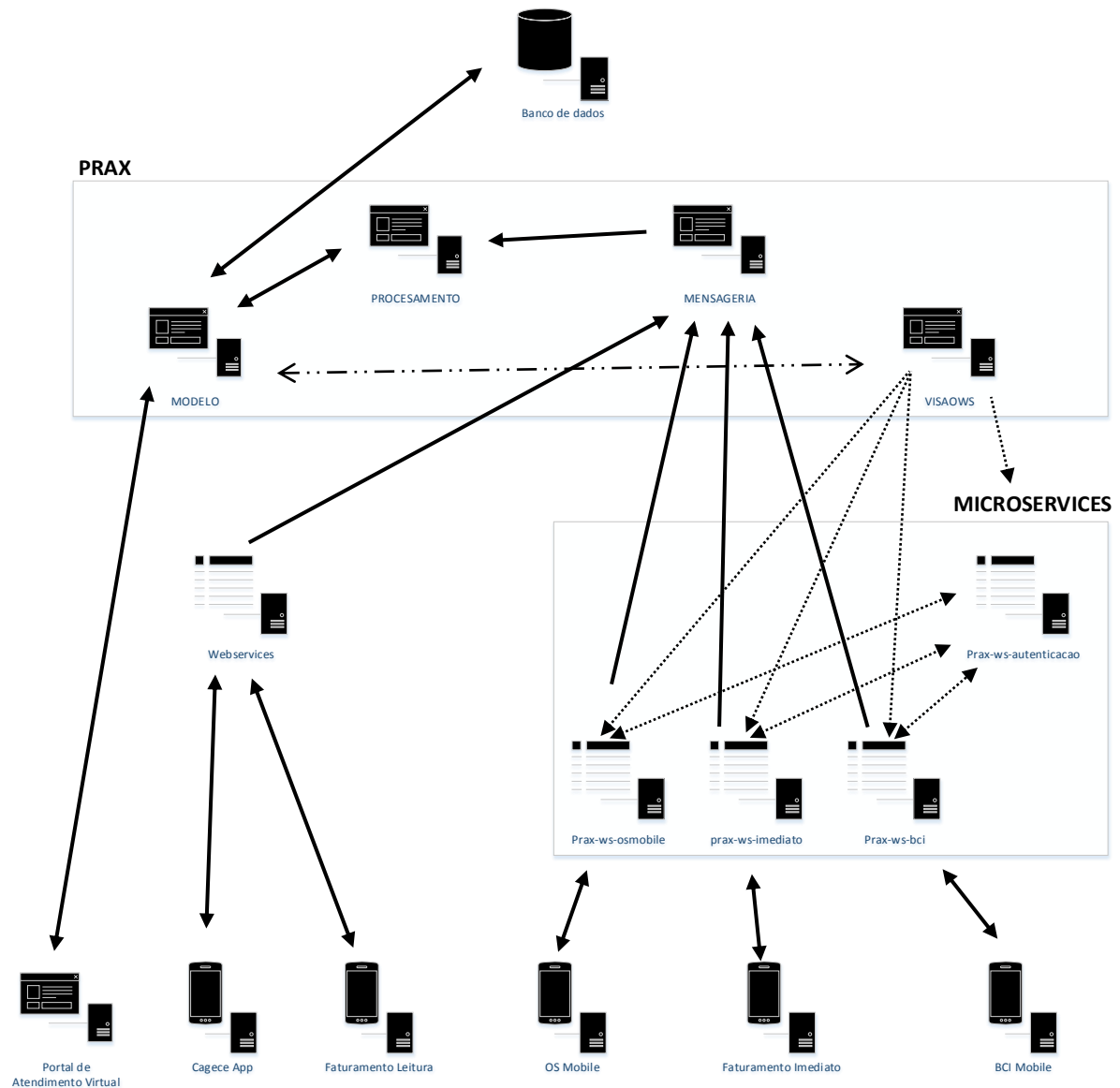


Figura 11 - Sistema Comercial PRAX

Fonte: Elaborado pelo autor.

Os aplicativos e o Portal de Atendimento se integram com o PRAX por meio dos serviços disponibilizados através dos Webservices e *microservices*. Os serviços disparam mensagens MDB's (Mensageria) que, por sua vez, disparam tarefas em lote para processamento.

Os aplicativos que se integram com o PRAX são:

- **Faturamento leitura:** Aplicação Android utilizadas para realização de leituras de medições para faturamento posterior (Faturamento Off-line). É utilizado pelos leituristas contratados pelas empresas terceirizadas;

- Faturamento Imediato: Aplicação Android utilizada para realização de faturamento em tempo real a partir das leituras de medições (Faturamento Online). É utilizado pelos leituristas contratados pelas empresas terceirizadas;
- OS Mobile: Aplicação Android que provê a geração de roteiros para execução de serviços, assim como o acompanhamento em campo. É utilizado pelas equipes executoras das US;
- BCI Mobile: Aplicação Android que permite a atualização cadastral dos imóveis dos clientes. É utilizado pelas equipes de cadastro das US;
- CageceApp: Aplicativo IO's que permite aos clientes a solicitação dos principais serviços realizados/oferecidos pela companhia;
- Portal Atendimento Virtual: Aplicação WEB que permite aos clientes a solicitação dos principais serviços realizados/oferecidos pela companhia;

A integração do PRAX com as aplicações apresentadas se dá por meio de um conjunto de bibliotecas geradas a partir dos módulos Webservice e Modelo.

Foram excluídos deste trabalho os aspectos de segurança, arquitetura de banco de dados e arquitetura de *hardware*.

3.2.1. Arquitetura do PRAX

O PRAX é uma solução distribuída, desenvolvida na linguagem Java, sob a plataforma J2EE, com 6 módulos distintos, que serão listados a seguir, destacando suas responsabilidades.

Os módulos que compõem o PRAX são:

- Ear: Nesse módulo estão todas as configurações necessárias ao processo de construção da aplicação como um todo a partir dos seus módulos (build) e geração de bibliotecas java (jars);
- Mensageria: Nesse módulo estão localizadas as classes (*Message Driven Beans* - MDB's) responsáveis pelo envio de mensagens ao provedor JMS do servidor de aplicações.

- **Modelo:** Nesse módulo são disponibilizadas as classes das entidades (Entity), serviços EJB e fachadas EJB, responsáveis pelas regras de negócio. A partir deste módulo são geradas diversas bibliotecas java (jars) responsáveis pela integração das aplicações, apresentadas no tópico anterior, com o PRAX. É a camada de negócio.
- **Processo:** Nesse módulo estão disponíveis as classes responsáveis por processamentos em lote (Tarefas *batches*); é a camada de processamento.
- **VisaoWS:** Módulo onde estão disponíveis os webservices sob a arquitetura de serviços REST. É a camada de webservices;
- **Visao:** Módulo que disponibiliza os recursos de imagens, páginas htmls, *Java Server Pages* (JSP), relatórios, *templates*, configurações, etc, utilizados pela aplicação. É a camada de apresentação.

A Figura 12 exhibe adiante os módulos componentes do PRAX.

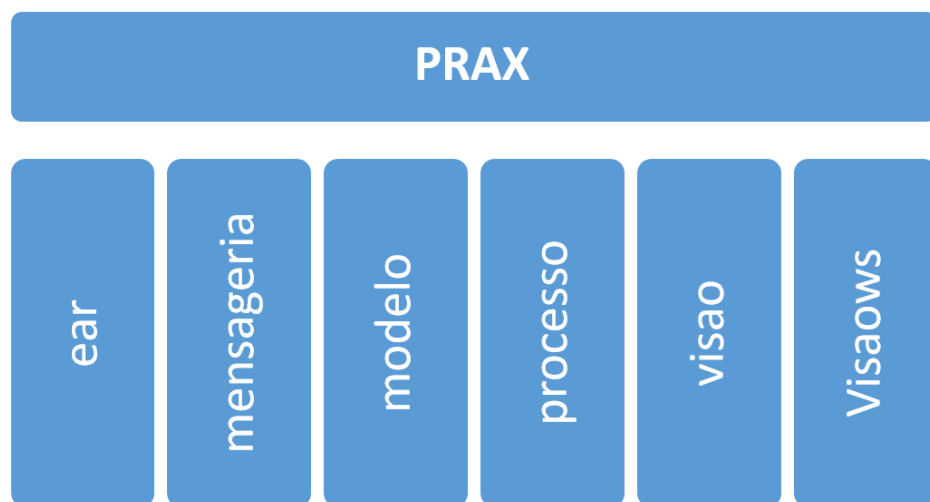


Figura 12 - Arquitetura do PRAX

Fonte: Elaborado pelo autor.

3.3. Controle de Versão

Todos os projetos da companhia eram publicados em um servidor SVN (*Subversion*) sem uma estratégia definida para gestão da versão.

A não utilização de uma estratégia de gestão de versão, com o auxílio de uma ferramenta como aquela em uso, era apenas, uma possibilidade de compartilhamento do código fonte das aplicações, através de um repositório centralizado, entre os desenvolvedores das diversas equipes.

As únicas operações utilizadas pelos desenvolvedores quando da manipulação do código fonte das aplicações eram: *checkout*, *commit* e *update*.

Todos os profissionais da gerência tinham acesso aos códigos fontes de todos os projetos da companhia.

As versões eram geradas a partir do diretório TRUNK, sem a utilização nem BRANCHES nem de TAGS. Os líderes de equipes eram responsáveis pela construção das versões e pela publicação delas no ambiente de produção.

A Figura 13, exibida a seguir, demonstra como os projetos eram organizados no servidor SVN.

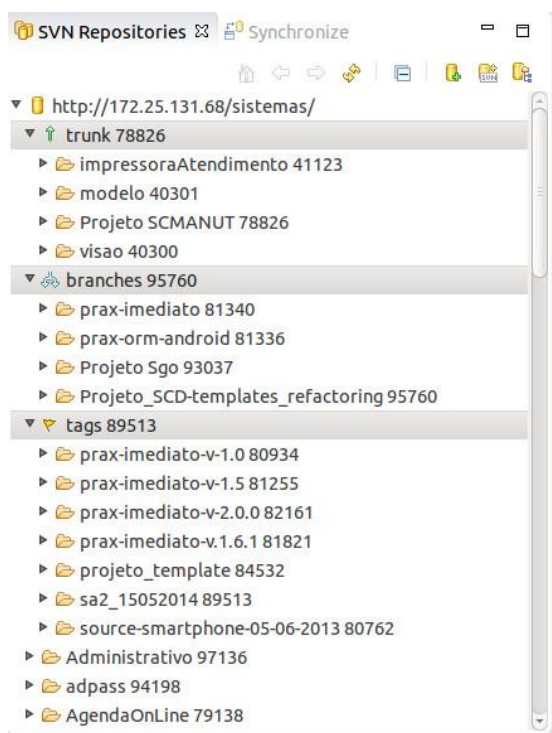


Figura 13 - Organização dos projetos no Servidor SVN.

Fonte: Elaborada pelo autor.

3.4. Processo de Desenvolvimento

A companhia tem um processo de desenvolvimento definido, documentado e disseminado, resultado da combinação da metodologia *Rational Unified Process* (ou

Processo Unificado *Rational*) - RUP com o *framework* SCRUM, que é adotado por todas as equipes de desenvolvimento da GETIC.

O Sistema de Gestão de Demandas – SGD é uma ferramenta da companhia construída para suportar o processo de desenvolvimento, que permite às equipes de desenvolvimento o planejamento de *Sprints* baseadas em demandas sugeridas tanto pelos usuários, pelos gestores, pela equipe de desenvolvimento, como também pela equipe de negócio da empresa.

3.5. Gestão de Mudanças

Nenhum procedimento para gerenciar mudanças de versões do PRAX era adotado pela companhia, apesar de uma definição clara do processo de mudança.

A realização de mudanças de versões era uma atividade corriqueira, não havendo planejamento. A qualquer momento, durante o dia, em que um problema fosse identificado e corrigido, uma versão era gerada e publicada no ambiente de produção, sem comunicação aos usuários e nem aprovação pela equipe de teste.

3.6. Problemas / Dificuldades

São apresentados a seguir alguns dos problemas provenientes da não utilização dos processos, modelos e técnicas adequadamente:

- Inexistência de um processo de teste;
- Equipe de teste limitada, em número de profissionais e ferramentas;
- Todos os testes para a liberação de versões do PRAX eram manuais, exploratórios e demorados;
- As versões do PRAX eram geradas e publicadas diariamente, causando problemas aos usuários finais, que tinham que interromper suas atividades de forma inesperada;
- Todos os desenvolvedores tinham acessos a todos os códigos fontes dos projetos da companhia;

- A não utilização de uma estratégia de controle de versão impedia a geração de versões emergenciais, uma vez que só era possível liberar uma nova versão quando todos os desenvolvedores concluíssem suas atividades;
- Muitas vezes as versões eram construídas com *commits* parciais dos desenvolvedores, levando para o ambiente de produção códigos incompletos;
- Falta de gestão de mudanças de versões.

3.7. Considerações Finais

Este capítulo mostrou o ambiente onde foi desenvolvido o estudo de caso, apresentando, a arquitetura do Sistema Comercial PRAX, a composição equipe de desenvolvimento, os processos de desenvolvimento e de teste existentes, a estratégia praticada para controle de versão, o processo de gestão de mudanças utilizado, e por fim, os problemas e os desafios para implantação das práticas selecionadas.

Conhecer o ambiente cenário deste estudo de caso é fundamental para o entendimento da metodologia utilizada para a implementação dos testes automatizados e implantação da integração contínua, que será apresentada no capítulo a seguir.

4. ESTUDO DE CASO: ESTRATÉGIA DE INTEGRAÇÃO CONTÍNUA E METODOLOGIA DE TESTES

Neste capítulo, será apresentada a solução implementada para a realização de integração contínua e utilização de testes automatizados durante a geração de versões do Sistema Comercial PRAX.

4.1. Tecnologias utilizadas

Toda a solução foi implementada com a utilização de 12 ferramentas *open-source*. As ferramentas foram distribuídas em quadro grupos de acordo com o propósito. Os grupos e ferramentas apresentados a seguir na Figura 14 serão detalhados adiante.

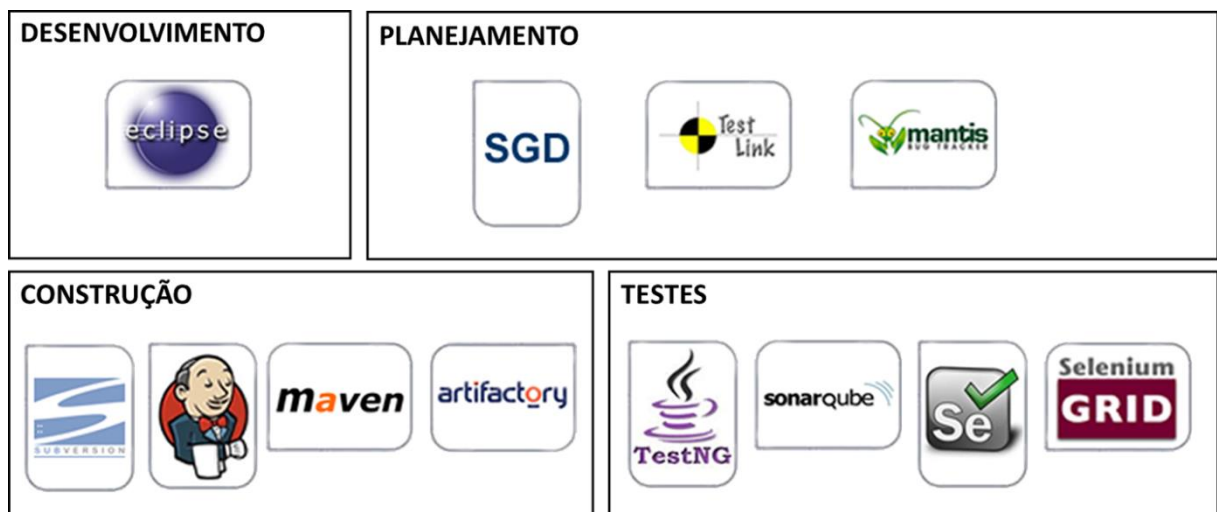


Figura 14 - Ferramentas utilizadas

Fonte: Elaborada pelo autor.

O primeiro grupo, Desenvolvimento, contém apenas a ferramenta de desenvolvimento IDE Eclipse, auxiliada por dois *pluguins* importantes que foram utilizados: o *plugin Subversive* como cliente SVN e o *plugin TestNg* para realização de teste a partir da ferramenta TestNG.

O segundo grupo, Planejamento, é composto pelas ferramentas:

- Sistema de Gestão de Demandas - SGD: utilizada no planejamento das *sprints*;

- *Testlink*: utilizada no planejamento dos testes; e
- *Mantis Bug Tracker*: utilizada na realização da gestão de incidentes.

No terceiro grupo, Construção, foram enquadradas quatro ferramentas:

- *Subversion*, como servidor de controle de versão;
- Jenkins, como servidor de integração contínua;
- Maven, ferramenta de build; e,
- *Artifactory*, como repositório para os artefatos criados e compartilhados com os demais projetos.

O quarto e último grupo, Testes, foi formado pelas ferramentas:

- TestNG, *framework* utilizado para construção e execução de testes unitários e funcionais;
- SonarQube utilizado para realização de testes estáticos;
- *Selenium*, *framework* utilizado para construção de testes funcionais; e,
- *Selenium Grid*, como servidor para realização em paralelo dos testes funcionais criados a partir do *Selenium* e executados pelo TestNG.

4.2. Macroprocessos

A Figura 15 apresenta uma visão macro dos processos definidos na GETIC, elaborados com o propósito de orientar a produção de softwares. É importante compreender como eles estão relacionados.

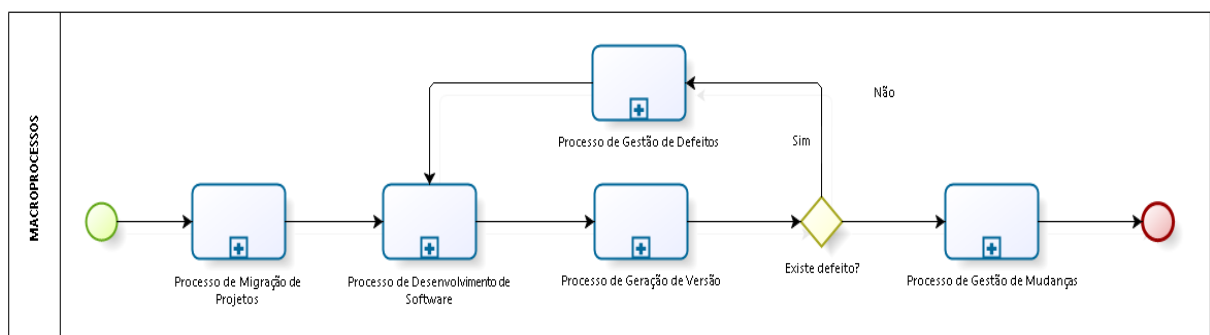


Figura 15 - Macroprocessos da GETIC

Fonte: Elaborada pelo autor.

Os processos disponibilizam um conjunto de atividades que disciplinam a produção e entrega de softwares da GETIC. Os processos definidos são:

- Processo de Migração de Projetos: Mecanismo criado para disciplinar a importação de projetos para o controle de versão e integração contínua.
- Processo de Desenvolvimento de Software: Mecanismo criado para garantir a produção de software com qualidade. Reúne atividades que, ordenadas, obtém um produto de software.
- Processo de Geração de Versão: Mecanismo responsável por planejar, gerar e disponibilizar versões dos produtos de softwares, incluindo atividades de testes e integração contínua.
- Processo de Gestão de Defeitos: Mecanismo responsável pela qualidade dos produtos produzidos pela GETIC, que é realizada através do monitoramento de *bugs*, falhas, erros provenientes dos ambientes de desenvolvimento, testes ou produção.
- Processo de Gestão de Mudanças: Mecanismo responsável pela liberação e publicação das versões;

Foi estabelecido um processo para a migração dos projetos da GETIC para a utilização da solução. Como o PRAX já estava em fase de desenvolvimento, fez se necessário o processo de migração.

Para que as atividades do processo de geração de versão possam ser iniciadas, ele depende totalmente do processo de desenvolvimento. Por outro lado, os processos de gestão de mudança e gestão de defeitos dependem dos resultados do processo de geração de versão.

4.2.1. Processo de Migração de Projetos

O processo de migração de projetos foi criado para facilitar a padronização dos projetos da GETIC, que passarão a utilizar o sistema de controle de versão e integração contínua.

Este processo possui 7 atividades que estão distribuídas entre as equipes: Equipe do projeto, Equipe de qualidade e Equipe de Segurança

A Figura 16 apresentada a seguir traz o processo de migração que será detalhado a partir da sua implementação no PRAX.

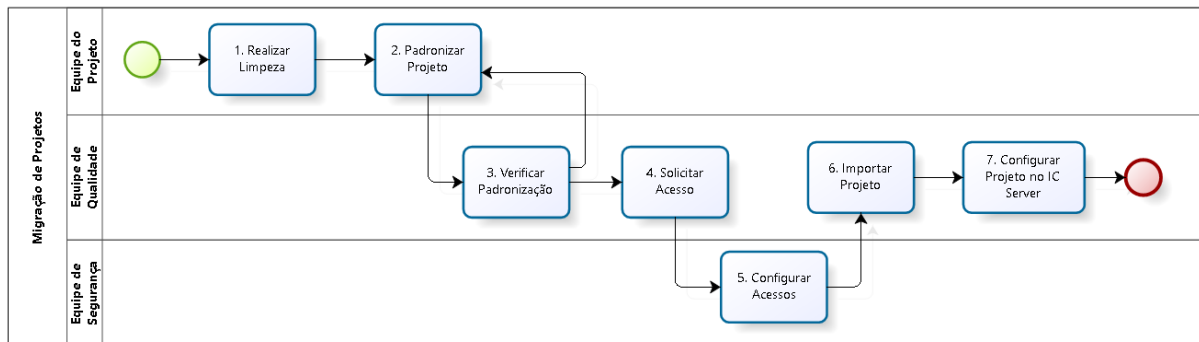


Figura 16 - Processo de migração de projetos

Fonte: Elaborada pelo autor.

O PRAX precisou passar por uma varredura em busca de todo e qualquer artefato dispensável para a construção da aplicação. A análise do que poderia ser retirado do projeto foi realizada pela equipe do projeto. Essa foi a primeira atividade do processo.

São exemplos de artefatos dispensáveis: Documentação/requisitos, arquivos temporários, duplicados, *templates*, imagens não utilizadas, etc.

Uma vez mantidos estes arquivos como parte do projeto, eles serão removidos durante a realização de manutenções do servidor SVN.

Para dar continuidade na migração, o PRAX precisou atender a um conjunto de requisitos/padrões para depois ser importado para o servidor SVN. A padronização do projeto também foi realizada pela equipe do projeto.

Os requisitos para a conclusão da segunda atividade do processo são:

- O sistema deverá exibir, na tela de abertura do sistema, como exemplificado abaixo na Figura 17, a versão da aplicação de acordo com o valor da chave **versao** existente em um arquivo **versao.properties**.

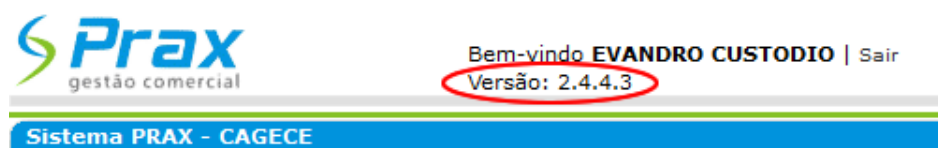


Figura 17 - Exibindo versão da aplicação

Fonte: Elaborada pelo autor.

- O projeto deverá estar configurado para utilização da ferramenta de build Apache Maven 3.x.
- O projeto precisará ter todas as bibliotecas (jarlibs) configuradas como dependência no pom.xml. As bibliotecas utilizadas pela aplicação serão disponibilizadas pelo servidor Artifactory.
- O projeto deverá estar cadastrado como um sistema no Sistema de Gestão de Demandas - SGD;

Na terceira atividade do processo, o PRAX passou por uma verificação, que confirmara o atendimento dos requisitos para migração do projeto. Caso o projeto não atendesse aos requisitos, uma lista de requisitos não atendidos seria repassada à equipe do projeto para realizar as readequações pendentes. A verificação foi realizada pela equipe de qualidade a partir da entrega, por parte da equipe do projeto, de uma cópia do projeto compactado no padrão .zip, .tar, ou .gz.

Após aprovação, a equipe de qualidade solicitou à equipe de segurança, através de e-mail, a criação de um grupo de acesso no servidor AD e inclusão dos membros da equipe do projeto no grupo criado.

Somente após a equipe de segurança configurar o acesso de cada integrante da equipe do projeto, a equipe de qualidade realizou a inclusão do projeto no repositório central do sistema de controle de versão.

Uma vez que o projeto foi importado para o servidor de controle de versão, a equipe de qualidade realizou a configuração do ambiente de integração contínua a partir da ferramenta Jenkins.

Esta configuração consiste em definir o processo de construção da aplicação, a periodicidade da construção, os testes que serão realizados, o servidor de teste onde a aplicação será publicada e os scripts que disponibilizarão a versão construída para a equipe de suporte.

Todos os projetos importados são submetidos a testes estáticos realizados pela ferramenta SonarQube.

4.2.2. Estrutura do Projeto

Uma vez importado para o repositório do SVN, o projeto passa a ter uma estrutura semelhante à apresentada na Figura 18.

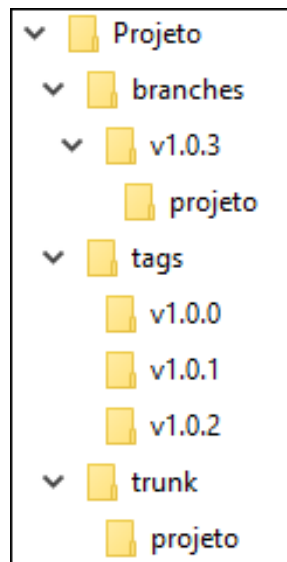


Figura 18 - Estrutura do Projeto no repositório SVN

Fonte: Elaborada pelo autor.

As pastas *trunk*, *branches* e *tags* desempenham papéis importantes na utilização do SVN e consequentemente na integração contínua. São elas:

- TRUNK - Nesta pasta se encontra a última versão funcional da aplicação, que não poderá, em hipótese alguma, conter erros, pois o servidor de integração contínua se baseará nela para realizar a próxima *build* da aplicação.
- BRANCHES - Nesta pasta se encontra a versão em desenvolvimento da aplicação. Poderá haver mais de uma versão em desenvolvimento.
- TAGS - Esta pasta armazenará, sempre, cópias de todas as versões da aplicação geradas e já liberadas. São cópias de segurança que, geralmente, são utilizadas para lançamento de versões emergenciais.

Cada desenvolvedor cria sua cópia de trabalho, a partir da versão disponível na pasta *branche*. Todas as alterações no projeto, acontecem primeiramente na cópia

de trabalho do desenvolvedor, e em seguida são compartilhadas com os outros membros da equipe do projeto através da operação *commit* do SVN.

Cada *commit* realizado deve corresponder a uma demanda no SGD, portanto, as informações a seguir são indispensáveis e devem ser adicionadas ao comentário.

O comentário contendo as informações sobre o *commit* deve:

- Iniciar com a sigla SGD seguida de um espaço;
- O número do SGD relacionado à atividade, seguido de um espaço;
- Entre colchetes, o percentual (%) que represente o andamento da atividade, seguido de um espaço;
- Se o *commit* se referir a uma correção, deve ser adicionado a palavra “MANTIS” seguida do símbolo “#” antes do número do relato; e,
- Uma descrição da atividade realizada.

A Figura 19, a seguir, apresenta como o *commit* deve ser comentado.

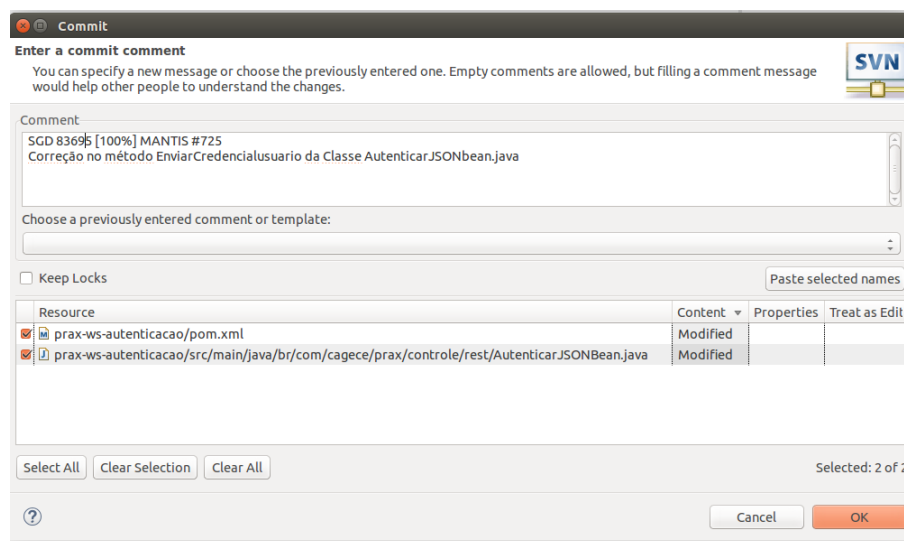


Figura 19 - Padrão da GETIC para realização de *commits*

Fonte: Elaborada pelo autor.

O SVN foi integrado com o SGD, de forma que, quando uma operação *commit* é realizada, o SVN envia ao SGD as informações sobre usuário, data, hora, número da release, comentário e classes/arquivos. O SGD disponibiliza uma tela com essas informações, como a apresentada na Figura 20.

Figura 20 - Tela do SGD contendo informações do *Commit* realizado

Fonte: Elaborada pelo autor.

4.2.3. Processo de Geração de Versão

O processo de geração de versão possui 10 atividades que estão distribuídas entre o Cliente, a Equipe de desenvolvimento, a Equipe de qualidade e o serviço de Integração contínua.

A Figura 21 apresenta a seguir o Processo de Geração de Versões.

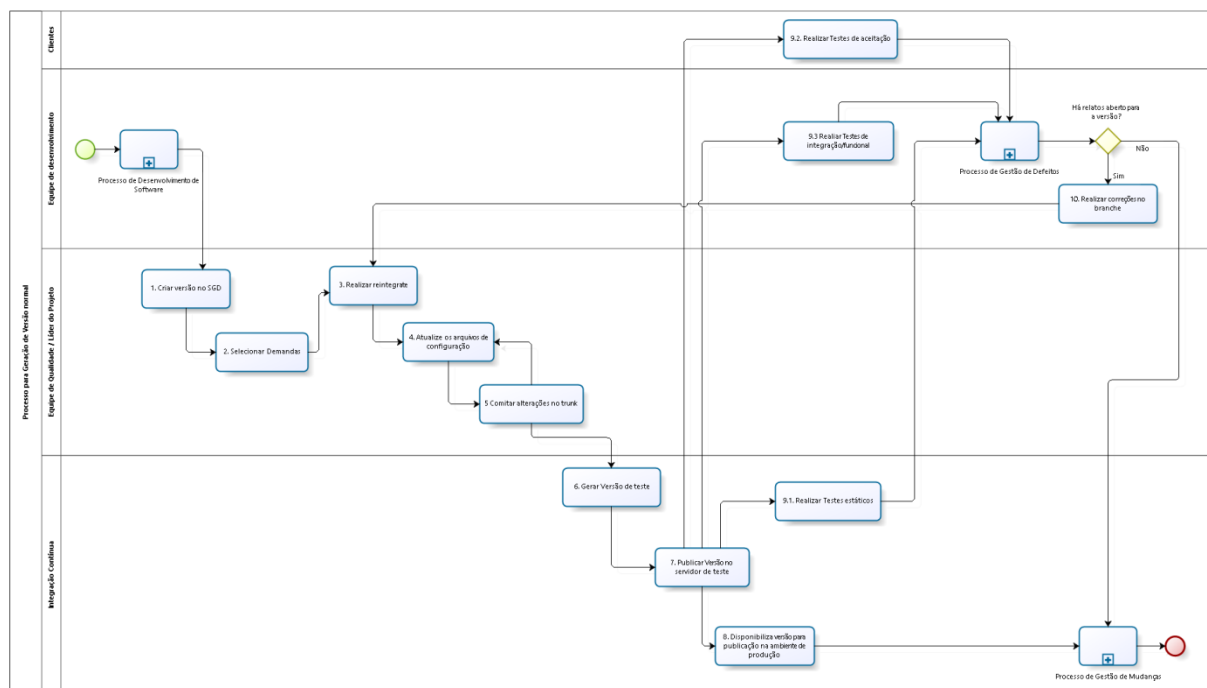


Figura 21 - Processo de Geração da Versão

Fonte: Elaborada pelo autor.

É importante lembrar que este processo está relacionado com três outros processos, dependendo, principalmente, do processo de desenvolvimento e interagindo com o processo de gestão de defeitos.

As necessidades dos clientes são repassadas para a equipe de desenvolvimento por meio de solicitações de demandas através do SGD. Essas demandas são agrupadas em *sprints* e implementadas pelas equipes dos projetos.

O código fonte das implementações é compartilhado no repositório do projeto no SVN, seguindo a estrutura apresentada no tópico anterior.

A primeira e segunda atividade do processo de geração de versão são realizadas pela equipe de qualidade e correspondem, respectivamente, ao cadastro da versão e seleção de demandas.

A geração de uma versão é iniciada quando, a partir do SGD, é criada uma nova versão e selecionadas as demandas que irão fazer parte dela. No processo de desenvolvimento, uma demanda tem o seu próprio ciclo de vida. Para que uma demanda possa fazer parte de uma versão, ela precisa estar com o *status* de “EM RELEASE”, e para isso, sempre que um desenvolvedor concluir uma atividade, ele modifica o status da demanda criada no SGD para “EM RELEASE”.

A tela de cadastro de uma versão no SGD é apresentada na Figura 22 a seguir.

Gerenc. de Desenvolvimento 2.096

Cagece Companhia de Água e Esgoto do Ceará

Portal Intranet Contato Genf Segunda-feira, 1 de Fevereiro de 2016

Administração Gerenciamento Movimentação Sprint Relatórios Monitoramento Logout

Versão - Cadastro

Sistema * PRAX - Sistema Comercial PRAX

Versão * 2 - H45

Base de dados Solicitações

Yoda	Num.	Sistema	Demanda	Previsão
<input checked="" type="checkbox"/>	74340	PRAX	SOLICITO QUE SEJA IMPLANTADO NA TELA ATENDIMENTO CENTRALIZADO E NA ATENDIMENTO CENTRALIZADO INTERNO NA PARTE DE GRANDES CLIENTE, O TELEFONE DOS EXECUTIVOS. SEGUER DADOS: MARCOS - 988955528 NADJA - 985633199 CLEUTON - 987321451 HELENA - 31011898/31012022 (PROVISÓRIO)	28/07/2015 a 28/08/2015
<input checked="" type="checkbox"/>	80866	PRAX	Solicitamos que na Aba "Atendimentos" da Tela "Atendimento centralizado interno" seja possível ordenar os serviços tanto pelo código do serviço quanto pela data de solicitação do serviço, em ordem ascendente ou descendente. Não há necessidade de listar a inscrição do imóvel, pois já é informado no cabeçalho da tela.	11/11/2015 a 29/01/2016
<input checked="" type="checkbox"/>	80944	PRAX	Bom tarde, solicitamos: 1) Inclusão de nome no relatório; 2) Acrescentar uma página com o resumo por loja dos valores informados ao final de cada loja; 3) Incluir nas colunas "Média" o valor médio do "Total por loja". Devido o tamanho do arquivo o mesmo não pode ser enviado neste pedido. Vou enviar por e-mail para o Aldivone. Atenciosamente,	11/11/2015 a 18/12/2015
<input type="checkbox"/>	81985	PRAX	1. Foi identificado que existe OS com o status[CANCELAMENTO DA SOLICITAÇÃO], porém a data de cancelamento não foi preenchida e sim a data de execução. EXEMPLO: Inscrição: 22513663, Atendimento: 112880687. 2. Embora tenha ocorrido o pagamento e a dívida baixa do atendimento 114313004, o mesmo está com a situação de [PENDENTE DE EXECUÇÃO]	27/11/2015 a 15/01/2016
<input type="checkbox"/>	82788	PRAX	NO RELATORIO DE ATENDIMENTO - SERVIÇOS REGISTRADO QUANDO INFORMA LOCALIDADE IGNORA TIPO DE ATENDIMENTO (CONFORME ORIENTAÇÃO RODOLFO/GETIC PARA CORRECAO)	10/12/2015 a 29/01/2016

Quantidade de solicitações marcadas para a release (3/29)

Confirmar Copiar Texto Voltar

Figura 22 - Tela do SGD para cadastro de versões

Fonte: Elaborada pelo autor.

A terceira atividade do processo diz respeito à reintegração do código fonte do *branch* da versão em desenvolvimento com a última versão funcional, que se encontra na pasta *trunk* do repositório do projeto. É de responsabilidade da equipe de qualidade realizar a reintegração.

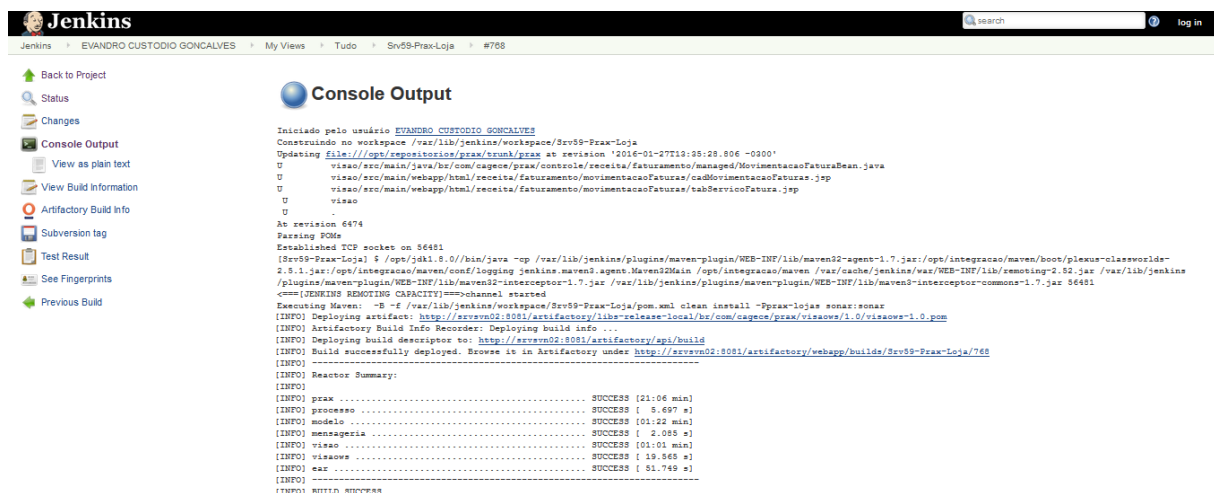
A próxima atividade consiste em atualizar os arquivos de configuração da versão do projeto. Os arquivos **pom.xml** e o **versao.properties** são os dois arquivos

de configurações que precisam ser atualizados, antes de ser disparado o processo de construção da próxima versão.

Após realizada a operação de reintegração do projeto ao *trunk* e atualizados os arquivos de configurações, é necessário enviar as alterações para o repositório do SVN. Essa é a quinta atividade do processo.

A sexta, sétima e oitava atividades do processo são realizadas pela ferramenta *Jenkins*, que é o servidor de integração contínua. Quando a construção da versão é disparada, o *Jenkins* lê a pasta *trunk* do repositório do SVN e aciona a ferramenta de build MAVEN, que por sua vez executa as instruções contidas no arquivo pom.xml do projeto.

A Figura 23, a seguir, apresenta o log do processo de construção da versão disparado pelo *Jenkins*.



```

Jenkins
EVANDRO CUSTODIO GONCALVES
My Views
Tudo
Srv59-Prax-Loja
#768

Back to Project
Status
Changes
Console Output
View as plain text
View Build Information
Artifactory Build Info
Subversion tag
Test Result
See Fingerprints
Previous Build

Console Output

Iniciado pelo usuário EVANDRO CUSTODIO GONCALVES
Construindo no workspace /var/lib/jenkins/workspace/Srv59-Prax-Loja
Updating file:///opt/repositorio/prax/trunk/prax at revision '2016-01-27T10:05:28.806 -0300'
U      visao/src/main/java/hr/com/cagece/prax/console/receita/faturamento/managed/MovimentacaoFaturaBean.java
U      visao/src/main/webapp/html/receita/faturamento/movimentacaoFaturas/cadMovimentacaoFaturas.jsp
U      visao/src/main/webapp/html/receita/faturamento/movimentacaoFaturas/tabServicoFatura.jsp
U
U
At revision 6474
Parsing POMs
Established TCP socket on 56481
[Srv59-Prax-Loja] $ /opt/sdki.8.0/bin/java -cp /var/lib/jenkins/plugins/maven-plugin/WEB-INF/lib/maven2-agent-1.7.jar:/opt/integracao/maven/boot/plexus-classworlds-2.5.1.jar:/opt/integracao/maven/conf/logging/jenkins.maven3.agent.Maven3Main:/opt/integracao/maven /var/cache/jenkins/war/WEB-INF/lib/remoting-2.52.jar /var/lib/jenkins/plugins/maven-plugin/WEB-INF/lib/maven2-interceptor-1.7.jar /var/lib/jenkins/plugins/maven-plugin/WEB-INF/lib/maven3-interceptor-commons-1.7.jar 56481
c=--[JENKINS REMOTING CAPACITY]--channel started
Executing Maven: -B -f /var/lib/jenkins/workspace/Srv59-Prax-Loja/pom.xml clean install -Pprax-loja sonar:sonar
[INFO] Deploying artifact: http://srv5902:8081/artifactory/libs-release-local/hr/com/cagece/prax/visao/1.0/visao-1.0.pom
[INFO] Artifactory Build Info Recorder: Deploying build info ...
[INFO] Deploying build descriptor to: http://srv5902:8081/artifactory/api/build
[INFO] Build successfully deployed. Browse it in Artifactory under http://srv5902:8081/artifactory/webapp/build/Srv59-Prax-Loja/768
[INFO] -----
[INFO] Reactor Summary:
[INFO]
[INFO] prax ..... SUCCESS (21:06 min)
[INFO] processo ..... SUCCESS ( 5.697 s)
[INFO] modelo ..... SUCCESS (01:22 min)
[INFO] mensageria ..... SUCCESS ( 2.055 s)
[INFO] visao ..... SUCCESS (01:01 min)
[INFO] ear ..... SUCCESS ( 19.565 s)
[INFO] ..... SUCCESS ( 51.748 s)
[INFO] BUILD SUCCESS
  
```

Figura 23 - Log do processo de construção da versão

Fonte: Elaborada pelo autor.

Como parte do ciclo de vida da construção da versão utilizando a ferramenta MAVEN, o código fonte é compilado, testado (testes unitários), empacotado, a versão é publicada no servidor de testes e adicionado ao repositório da ferramenta Artifactory.

A nona atividade corresponde à realização de testes e é subdividida em três outras atividades, testes estaticos, testes funcionais e testes de aceitação, realizados, respectivamente, pelo SonarQube, *Selenium*/equipe de teste e pelo usuário final.

Uma vez publicada a versão gerada no servidor de teste, o Jenkins aciona a ferramenta SonarQube para dar início à realização dos testes estáticos.

A Figura 24, apresentada a seguir, mostra o resultado dos testes estáticos realizados no PRAX.

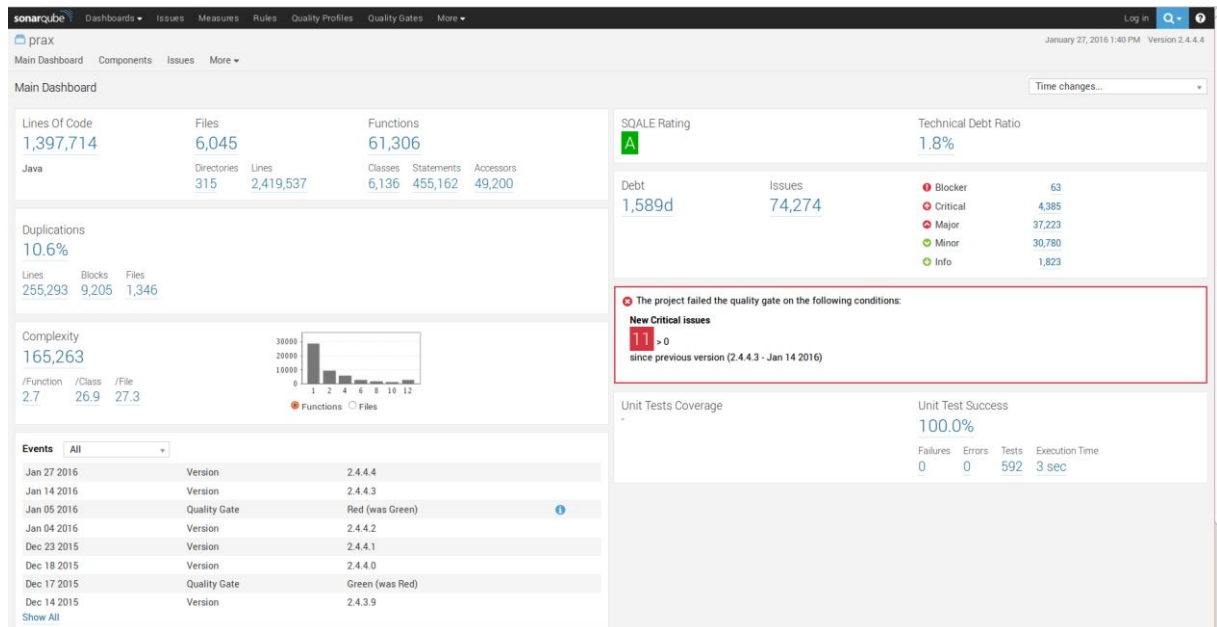


Figura 24 - Testes estáticos realizados pela ferramenta SonarQube

Fonte: Elaborada pelo autor.

Os testes funcionais são realizados em duas etapas. Na primeira, o Jenkins aciona a ferramenta TestNG para a execução dos testes automatizados através da ferramenta *Selenium*. São realizados 259 casos de testes, com duração média de oito horas. Caso sejam realizados em paralelo, utilizando os recursos das ferramentas TestNg e *Selenium Grid*, o tempo de execução dos testes é reduzido para em média de uma hora e meia.

Alguns casos de testes funcionais são realizados manualmente pela equipe de qualidade, tendo em vista a complexidade de cada caso. Os clientes, em algumas situações, também realizam os testes de aceitação.

Os testes automatizados são monitorados através de um painel onde são exibidas, em tempo real, informações sobre a execução dos testes. Caso haja alguma falha, um e-mail é enviado automaticamente para a equipe de qualidade e o teste é adicionado a uma lista de testes falhos no painel.

Os *feedbacks* dos testes falhos são as principais entradas para o processo de gestão de incidentes que será apresentado no tópico a seguir.

A Figura 25, apresentada a seguir, exibe a ferramenta *Selenium Grid* em ação.

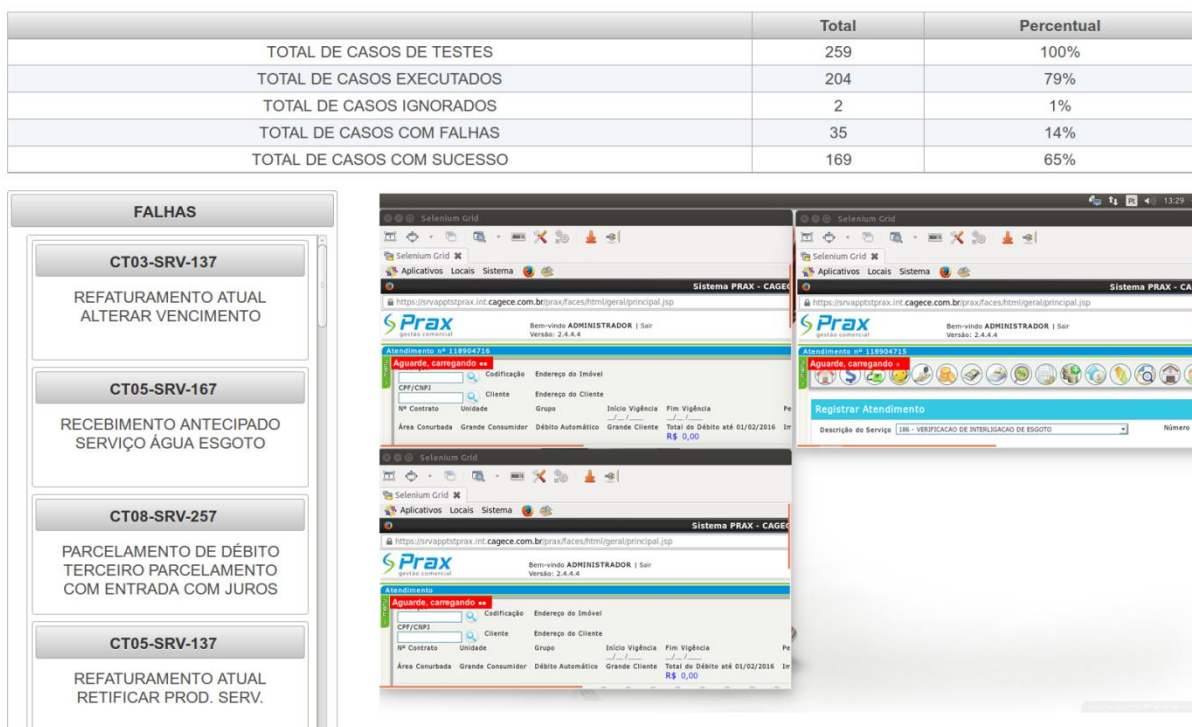


Figura 25 - Selenium Grid realizado testes funcionais em paralelo.

Fonte: Elaborada pelo autor.

Uma vez que sejam encontradas falhas, estas são reportadas pelo processo de gestão de defeitos e devem ser corrigidas pelo desenvolvedor. O código fonte das correções é reintegrado à versão, que, por sua vez, é reconstruída e uma nova bateria de testes é realizada.

Quando todos os incidentes forem resolvidos e todos os testes funcionais e testes de aceitação forem concluídos, entra em ação o processo de gestão de mudanças, que é o processo responsável pela troca de versão no ambiente de produção.

4.2.4. Processo de Gestão de Defeitos

O processo de gestão de defeitos, possui 6 atividades que estão distribuídas entre a equipe de desenvolvimento, o líder de equipe e a equipe de qualidade.

A Figura 26, a seguir, apresenta o Processo de Gestão de Defeitos.

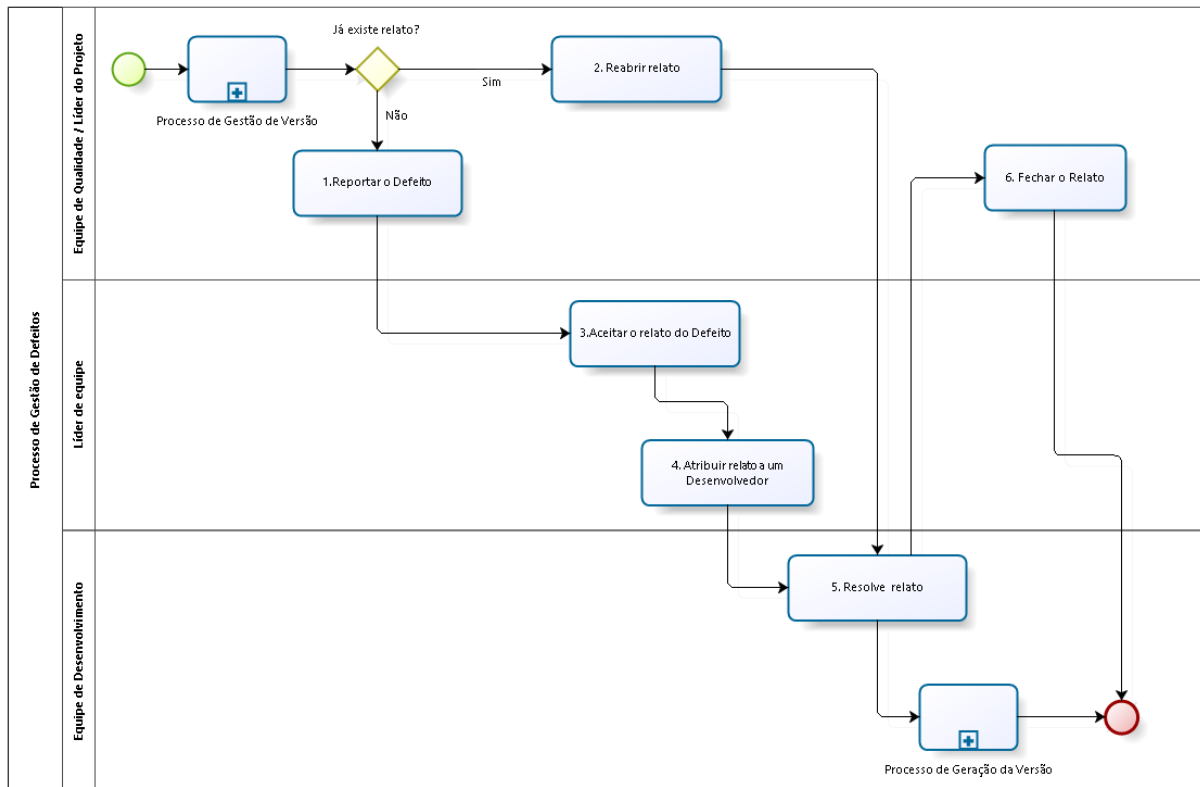


Figura 26 - Processo de Gestão de Defeitos

Fonte: Elaborada pelo autor.

O ciclo de vida de uma falha é acompanhado pela ferramenta *MantisBT* através de relatos. Assim, para cada falha encontrada durante a realização dos testes, é aberto um relato, que, em um segundo momento, deve ser avaliado pelo líder do projeto, que decide se aceita ou não o relato.

Caso seja aceito, o relato é atribuído a um desenvolvedor, que se encarregará de resolver o relato, ou seja, realizará a devida correção e, em seguida, submeterá o código corrigido ao repositório SVN do projeto.

Quando o relato for resolvido e a correção incorporada à versão em teste, a equipe de qualidade realiza os testes de regressão. Caso a falha persista, o relato é reaberto e encaminhado novamente para o desenvolvedor, do contrário o relato é fechado.

4.2.5. Processo de Gestão de Mudanças

O processo de gestão de mudanças possui 7 atividades que estão distribuídas entre a equipe de mudança, equipe de riscos e segurança, o líder de equipe e/ou equipe de qualidade.

A Figura 27, a seguir, apresenta o Processo de Gestão de Mudanças.

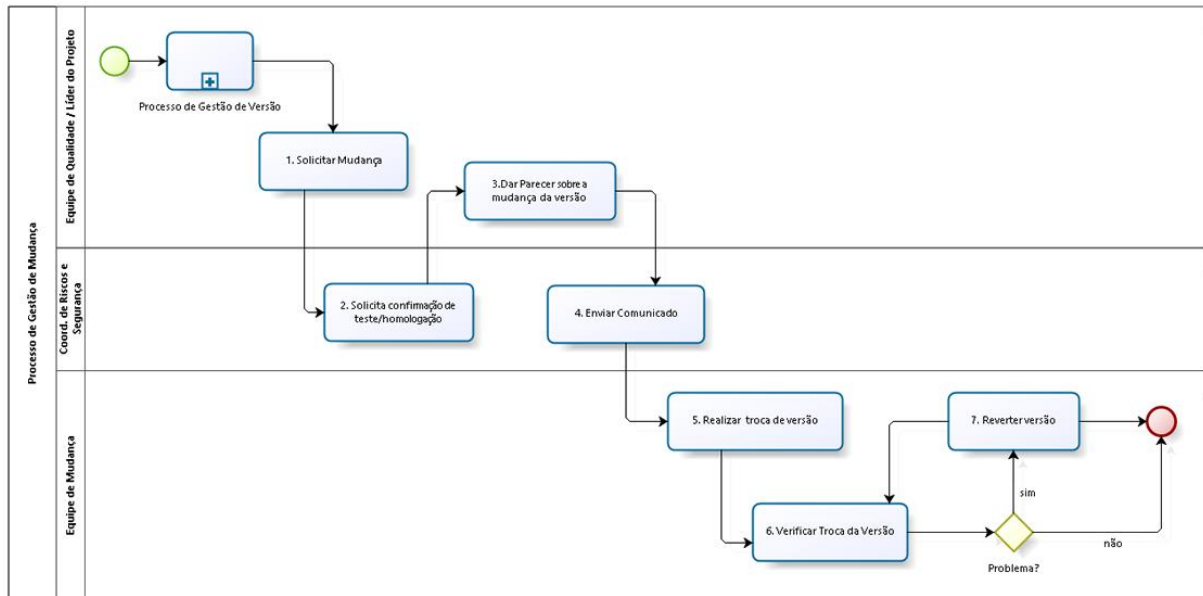


Figura 27 - Processo de Gestão de Mudanças

Fonte: Elaborada pelo autor.

Uma vez que a versão passou por todos os testes e foi devidamente aprovada pela equipe de qualidade, é necessário realizar a publicação da versão no ambiente de produção.

O processo de gestão de mudanças é iniciado pela solicitação formal de mudança, através do preenchimento do formulário de mudanças, onde constam informações relevantes para a correta execução de troca de versões.

O formulário de mudança contém informações sobre horários para troca de versão, um resumo sobre as mudanças implementadas na versão e define quais ativos serão afetados com a mudanças. Essas informações servirão para documentação das mudanças realizadas.

A equipe de segurança recebe a solicitação de mudança e solicita o parecer da equipe de qualidade quanto à homologação da versão. Dado o parecer favorável, é

enviado um comunicado a todos os interessados sobre o planejamento da troca da versão.

Após o comunicado, a equipe de mudança, no horário estipulado no formulário de mudança, realiza a troca da versão no ambiente de produção e, em seguida, realiza a verificação da mudança. Caso ocorra algum problema durante a troca ou após a troca da versão, a equipe de mudança realiza uma reversão e aborta a mudança, informando os motivos que impossibilitaram a sua realização.

A Figura 28, exibida a seguir, apresenta o formulário de solicitação de mudança e o comunicado de troca de versão enviado aos interessados.

FORMULÁRIO DE SOLICITAÇÃO DE MUDANÇA

* Informe a finalidade da Mudança :

* Data de Início : * Data de Finalização :

* Qual o ativo afetado? * Qual o ativo afetado?

Itens impactados pelo ativo

* Complexidade da Mudança

* Informe mais detalhes

Cagece COMUNICADO GETIC

AVISO DE INDISPONIBILIDADE DE SERVIÇO

Comunicamos que o sistema **PRAX** ficará indisponível hoje (28/01/2016) no horário de **22:00h às 23:00hs** para implementação da versão 2.4.4.4

Na realização deste serviço serão aplicadas correções e melhorias realizadas.

Obs: pedimos aos usuários que efetuem a limpeza do cache do seu navegador para disponibilização correta da nova versão. Para saber como proceder [clique aqui](#). Ou acesse o link abaixo:

<http://srvotrs.int.cagece.com.br/otrs/customer.pl?Action=CustomerFAQZoom;ItemID=232>




Figura 28 - Formulário e comunicado de mudança

Fonte: Elaborada pelo autor.

Quando a mudança de versão é realizada com sucesso, a equipe de qualidade cria, a partir da pasta *trunk* do repositório SVN do projeto, uma *tag*, que nada mais é do que uma cópia da versão finalizada, que servirá para geração de versões emergências até que uma nova versão seja gerada e, conseqüentemente, uma nova *tag* seja criada.

4.2.6. Testes automatizados

Os testes funcionais automatizados são realizados sempre que o processo de *build* realizado pelo servidor de IC é finalizado. Já foram implementados 259 testes utilizando o *framework Selenium WebDriver*.

O PRAX tem mais de 600 testes unitários implementados que são executados durante o processo de construção da versão.

Os testes unitários objetivam, principalmente, proporcionar cobertura de teste no módulo de faturamento, enquanto os testes funcionais com o *Selenium* foram direcionados para o módulo de atendimento do PRAX.

4.2.7. Integração contínua

As principais aplicações geradas a partir do projeto PRAX se encontram configuradas e em plena utilização do serviço de Integração Contínua. Atualmente são 18 aplicações com o processo de construção dependentes da construção do PRAX.

Os aplicativos móveis, em um primeiro momento, não foram beneficiados pela solução, porém, todos os artefatos necessários para a construção dos aplicativos já são gerados por consequência da construção do módulo EAR do PRAX.

Todos os projetos construídos a partir do servidor de IC passam por testes estáticos, que são realizados pela ferramenta SonarQube.

Os testes funcionais com *Selenium* são realizados após o servidor de IC realizar a publicação do projeto PRAX no servidor de testes.

A Figura 29, a seguir, traz uma das visualizações dos projetos configurados e construídos automaticamente pelo servidor *Jenkins*.

The screenshot shows the Jenkins web interface. The top navigation bar includes the Jenkins logo, a search bar, and the user name 'EVANDRO CUSTODIO GONCALVES' with a 'log out' link. The breadcrumb trail shows 'Jenkins > Servidor 72 (Testes)'. The sidebar on the left contains various navigation links such as 'New Item', 'People', 'Build History', 'Edit View', 'Delete View', 'Project Relationship', 'Check File Fingerprint', 'Manage Jenkins', 'Credentials', 'My Views', and 'Disk Usage'. The main content area displays a table of build jobs for 'Servidor 72 (Testes)'. The table has columns for status (S), weather icon (W), name, last success, last failure, and last duration. The jobs listed are 'Sn59-Prax-Loja', 'Sn59-Prax-Testes-Automatizados', 'Sn59-SA2', 'Sn72-AtendimentoVirtual', 'Sn72-Prax-Webservice', 'Sn72-prax-ws-autenticacao', 'Sn72-prax-ws-imediato', 'Sn72-prax-ws-modelo', and 'Sn72-prax-ws-utils'. Below the table, there is a 'Build Queue' section showing 'No builds in the queue' and a 'Build Executor Status' section showing '1 Idle' and '2 Idle'. The footer of the page includes a 'Help us localize this page' link, a page generation timestamp 'Page generated: Feb 1, 2016 2:47:17 PM', and a 'REST API' link.

S	W	Name	Last Success	Last Failure	Last Duration
		Sn59-Prax-Loja	5 days 1 hr - #768	7 days 2 hr - #762	26 min
		Sn59-Prax-Testes-Automatizados	25 days - #51	N/A	6 hr 49 min
		Sn59-SA2	24 days - #35	N/A	5 min 40 sec
		Sn72-AtendimentoVirtual	2 mo 24 days - #347	N/A	19 sec
		Sn72-Prax-Webservice	1 mo 25 days - #101	N/A	16 sec
		Sn72-prax-ws-autenticacao	2 days 23 hr - #23	N/A	13 sec
		Sn72-prax-ws-imediato	2 days 23 hr - #22	N/A	14 sec
		Sn72-prax-ws-modelo	10 days - #4	N/A	1 min 41 sec
		Sn72-prax-ws-utils	2 days 23 hr - #22	N/A	12 sec

Figura 29 - Tela do Servidor *Jenkins*

Fonte: Elaborada pelo autor.

4.3. Implantação

À medida que, naturalmente, o Sistema Comercial PRAX evoluía, era inevitável não pensar em uma solução que viesse agilizar o processo de entrega e contribuir para um ganho significativo na sua qualidade.

O projeto foi iniciado em dezembro/2013, a partir da elaboração de um projeto que, em seu cronograma, contemplava o estudo sobre as ferramentas, reestruturação do servidor de controle de versão, configuração de servidores e integração entre as ferramentas, implementação de testes automatizados e implantação da integração contínua. Todo o processo para implementação da solução durou cerca de um ano e meio.

As principais atividades realizadas para implantação da solução foram:

- Estudo de ferramentas que poderiam ser utilizadas;
- Definição e documentação da arquitetura;
- Configuração do ambiente de testes, que consistia em instalar e integrar as ferramentas definidas na arquitetura da solução;
- Reestrutura do SVN e migração de projetos;

- Definição do processo de testes;
- Adequações no processo de desenvolvimento;
- Definição da arquitetura dos testes automatizados;
- Implementação dos testes automatizados;
- Capacitação da equipe de desenvolvedores;

Durante a implantação, todas as equipes foram envolvidas, tanto com o propósito de dar conhecimento, como para demonstrar a importância e os benefícios para a companhia como um todo.

Um marco da implantação foi a realização de um treinamento sobre testes de softwares para toda a equipe do Sistema Comercial PRAX, o que gerou uma valorização do trabalho, que até então vinha sendo desenvolvido pela equipe de teste, como também, despertou o interesse na solução.

4.4. Dificuldades para implantação

A maior dificuldade para tornar realidade a solução apresentada neste trabalho foi a limitação da equipe de qualidade no tocante ao seu número de profissionais, considerado o porte do sistema comercial. Os resultados significativos da solução foram mais demorados.

Outras dificuldades encontradas para implantação são apresentadas a seguir:

- Toda a instalação e configuração da solução foi feita pela equipe de qualidade;
- Implementação dos testes automatizados sempre era preterida pela necessidade de realização de testes manuais para liberação de versões;
- A falta de qualificação da equipe de desenvolvimento para utilização das ferramentas selecionadas;
- A demora para apresentação de resultados palpáveis da solução; e,
- A adequação constante dos servidores para suportarem a solução;

4.5. Benefícios

Depois de quase dois anos de utilização, podemos apresentar os principais benefícios proporcionados pela solução descrita neste trabalho:

- Maior segurança no processo de entregas;
- Melhoria na qualidade do produto;
- Melhoria do Processo de Desenvolvimento;
- Utilização de um Processo de Testes;
- Utilização de um Processo de Gestão de Mudanças;
- Utilização de um Processo de Gestão de Defeitos;
- Registro, rastreabilidade e controle das mudanças sofridas pelo projeto durante seu ciclo de vida;
- Redução dos riscos trazida pela visualização rápida do estado de integração do sistema;
- Geração de versões emergenciais, o que antes era impossível;
- Utilização plena de testes automatizados;
- Rastreabilidade de informações de mudanças para fins de auditoria.

4.6. Considerações Finais

Este capítulo apresentou as tecnologias utilizadas e os macroprocessos criados para nortear a implementação dos testes automatizados no Sistema Comercial PRAX e a implantação da integração contínua na GETIC, além de elencar as dificuldades e benefícios da utilização destas práticas.

A estrutura de controle de versão adotada e os processos de migração de projetos, geração de versão, gestão de defeito e gestão de mudanças foram apresentados à medida que a metodologia de implantação era relatada.

5. CONCLUSÃO

Este trabalho apresentou o estudo de caso realizado na Companhia Água e Esgoto do Estado do Ceará – CAGECE para descrever a utilização de testes automatizados e implantação da integração contínua, concomitantemente ao desenvolvimento do Sistema Comercial PRAX.

Para uma maior compreensão do estudo, foi fornecido um referencial teórico sobre testes de softwares, com os principais conceitos da área, as atividades, modelos, métodos, níveis, técnicas e estratégias de testes, incluindo ainda, integração contínua, seus principais componentes e benefícios da utilização desta prática. A gestão de defeitos, a gestão de mudanças e o controle de versão também foram explorados

Um panorama inicial de como a companhia lidava com o desenvolvimento do Sistema Comercial PRAX foi apresentado, trazendo as dificuldades existentes no cenário anterior à utilização dos testes automatizados e da integração contínua.

Em seguida, foram apresentadas as ferramentas e descritos os macroprocessos criados com o propósito de orientar a produção de softwares, objetivos deste trabalho. Os processos de migração de projetos, geração de versão, gestão de defeitos e gestão de mudanças, componentes da solução, tiveram suas atividades detalhadas em meio à apresentação da metodologia de implantação das práticas ágeis.

Foi feito um resumo do processo de implantação, expondo as dificuldades encontradas e os benefícios alcançados após a implementação das práticas na companhia.

Os pontos fortes que contribuíram para a realização deste trabalho foram:

- O envolvimento das equipes de desenvolvimento do PRAX, da qualidade, de infraestrutura, de segurança e suporte diante da importância da solução implementada;
- O apoio incondicional dos gestores da companhia e em especial da GETIC;
- O comprometimento da equipe de qualidade para superar as limitações e dificuldades.

As dificuldades que retardaram a utilização da solução foram:

- O número reduzido de profissionais da equipe de qualidade que desenvolverem as atividades de instalação e configuração da solução;
- A necessidade de liberação de versões com a realização de testes manuais;
- A complexidade da solução, o que exigiu muito estudo e tempo dedicado.

Por fim, a partir da utilização solução, os seguintes benefícios foram verificados:

- A possibilidade de geração de versões emergenciais;
- A rastreabilidade das mudanças, para fins de auditoria;
- A agilidade na realização de testes, a partir da automatização, reduzindo significativamente o número de testes manuais.

5.1. Trabalhos futuros

Como trabalhos futuros, são sugeridos os seguintes:

- Pesquisar e propor um conjunto de indicadores que permitam mensurar os benefícios da solução proposta a partir deste trabalho;
- Pesquisar e documentar os padrões na arquitetura dos testes automatizados através da solução proposta;
- Pesquisar e propor a incorporação de outras ferramentas com o intuito de ampliar a realização de testes, como os de performance, sobrecarga e testes de *webservices*;
- Pesquisar e propor a implementação de testes automatizados para as aplicações *mobile* auxiliares do Sistema Comercial PRAX.

REFERÊNCIAS

ANICHE, Mauricio. **Test-Driven Development: Teste e Design no Mundo Real**. São Paulo: Casa do Código, 2012. 167 p.

BENNERTZ, Cristiano. **Ferramenta de Apoio ao Processo de Gestão de Defeitos**. 2011. 68 f. TCC (Graduação) - Curso de Curso de Sistemas de Informação, Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau, 2011.

CORILLO, Mirilian Carla Araujo; JUBILEU, Andrea Padovan. **MODELO DE PROCESSO PARA MICRO E PEQUENAS EMPRESAS DE SOFTWARE COM BASE EM METODOLOGIAS ÁGEIS**. Disponível em: http://www.researchgate.net/publication/267786151_MODELO_DE_PROCESSO_PARA_MICRO_E_PEQUENAS_EMPRESAS_DE_SOFTWARE_COM_BASE_EM_METODOLOGIAS_GEIS>. Acesso em: 15 fev. 2014.

CRISPIN, Lisa; GREGORY, Janet. **Agile testing: A practical guide for testers and agile teams**. Boston, Ma, Usa: Pearson Education, Inc., 2009. 533 p. (The Addiso).

DUVALL, Paul M.; MATYAS, Steve; GLOVER, Andrew. **Continuous Integration: Improving Software Quality and Reducing Risk**. Boston, Ma, Usa: Pearson Education, Inc., 2007. 318 p. (The Addiso).

HIRAMA, Hechi. **Engenharia de Software: Qualidade e Produtividade com tecnologia**. Rio de Janeiro: Elsevier, 2011. 210 p.

LENZI, Thiago Fabian. **Ferramenta de suporte a gestão de defeitos com integração entre 0800net e mantis**. 2012. 57 f. TCC (Graduação) - Curso de Curso de Sistemas de Informação, Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau, 2012.

LUPPI, Eduardo José; NOGUEIRA, Marcelo. Implementação de teste de software. In: VII INTERNATIONAL CONFERENCE ON ENGINEERING AND COMPUTER EDUCATION, 7., 2011, Guimaraes, Portugal. **Engineering Education inspiring the next generation of engineers**. Guimaraes, Portugal: Icece, 2011. v. 7, p. 271 - 273.

MARTINS, Marcelo Carvalho. **Sistema web para gerenciamento de mudanças na governança de ti em conformidade com o framework information technology infrastructure library (ITIL)**. 2007. 86 f. TCC (Graduação) - Curso de Curso de Sistemas de Informação, Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau, 2007.

MOLINARI, Leonardo. **Testes de Software: Produzindo Sistemas Melhores e Mais Confiáveis**. 2. ed. São Paulo: Erika, 2003. 228 p.

PRESSMAN, Roger S.. **Engenharia de Software: Uma Abordagem Profissional**. 7. ed. Porto Alegre: Amgh, 2011. 771 p. Tradução de: Ariovaldo Griesse, Mauro Moro Fecchio.

ROSELI PINHEIRO (São Paulo). Ti Exames Consultoria e Treinamentos Ltda (Org.). **Fundamentos em Testes de Softwares: Formação essencial para analistas de testes**. 2014. Disponível em: <www.tiexames.com.br>. Acesso em: 14 jul. 2014.

SMART, John Ferguson. **Jenkins The Definitive Guide: Continuous Integration for the Masses**. Sebastopol: O'reilly Media, 2010. 441 p.

SOMASUNDARAM, Ravishankar. **Git: Version Control for Everyone: Beginner's Guide**. Birmingham: Packt Publishing, 2013. 180 p.

SOMMERVILLE, Ian. **Engenharia de Software**. 9. ed. São Paulo: Pearson, 2011. 529 p.

THOMAS MULLER (Belgica). International Software Testing Qualifications Board (Org.). **Certified Tester Foundation Level Syllabus**. 2011. Disponível em: <http://www.bstqb.org.br/uploads/docs/syllabus_ctfl_2011br.pdf>. Acesso em: 12 dez. 2014.