

UNIVERSIDADE DE CAXIAS DO SUL  
CENTRO DE COMPUTAÇÃO E TECNOLOGIA DA INFORMAÇÃO  
BACHARELADO EM SISTEMAS DE INFORMAÇÃO

**Eduardo José Cauzzi**

Proposta de plano de garantia da qualidade de software para o laboratório de  
criação e aplicação de software

Eduardo José Cauzzi

ejcauzzi@ucs.br

Iraci Cristina da Silveira de Carli

Caxias do sul

2015

**Eduardo José Cauzzi**

Proposta de plano de garantia da qualidade de software para o laboratório de  
criação e aplicação de software

Trabalho de Conclusão de  
Curso para obtenção do  
Grau de Bacharel em  
sistemas de informação da  
Universidade de Caxias do  
Sul.

Iraci Cristina da Silveira de Carli

Orientadora

Caxias do sul

2015

Dedico este trabalho à minha  
família e aos meus amigos e  
colegas que me auxiliaram  
em toda a jornada  
acadêmica.

## **RESUMO**

A garantia de qualidade de software é indispensável para qualquer processo de desenvolvimento de software. Com ela é possível garantir o atingimento da qualidade necessária para o que se desenvolve, atingir os objetivos acordados com as partes interessadas, reduzir custos com retrabalho e diminuir o tempo de projeto. Os métodos de verificação e validação permitem que a solução desenvolvida tenha sua qualidade avaliada durante todo o processo de desenvolvimento, garantindo que cada fase seja entregue conforme o esperado. Com projetos com qualidade elevada, é possível esperar uma satisfação maior do cliente pois estará recebendo uma aplicação conforme o que havia solicitado. Além dele, a própria organização terá ganhado pois trabalhará mais com atividades construtivas e menos com atividades corretivas.

Neste trabalho propôs um plano de garantia da qualidade de software para o laboratório de criação e aplicação de software da Universidade de Caxias do sul, utilizando também ferramentas de apoio da gestão do plano.

## LISTA DE ILUSTRAÇÕES

Ilustração 1 - Pilares do processo de garantia da qualidade de software .....	13
Ilustração 2 – Custo de alteração em um projeto de software .....	22
Ilustração 3 – Incidência de erros por fase.....	23
Ilustração 4 – Testes de caixa-branca.....	44
Ilustração 5 – Testes de caixa-preta .....	46
Ilustração 6 – Macroprocesso atual.....	55
Ilustração 7 – Manutenção de software.....	55
Ilustração 8 –Processo de desenvolvimento de novos projetos .....	56
Ilustração 9 – Subprocesso de criação de plano de projeto .....	57
Ilustração 10 – Subprocesso de planejamento de sprints .....	57
Ilustração 11 – Subprocesso de execução de sprints .....	58
Ilustração 12 – Subprocesso de testes .....	58
Ilustração 13 – Subprocesso de homologação.....	59
Ilustração 14 – Subprocesso de implantação.....	60
Ilustração 15 – Subprocesso de encerramento de projeto .....	60
Ilustração 16 – Macroprocesso proposto .....	62
Ilustração 17 - Processo de criação e planejamento do projeto proposto .....	63
Ilustração 18 - Processo de desenvolvimento proposto .....	73
Ilustração 19 – Processo de homologação proposto.....	76
Ilustração 20 - Processo de implantação proposto.....	83
Ilustração 21 –Processo de encerramento proposto .....	84
Ilustração 22: Usuários criados com seus perfis. ....	102
Ilustração 23: Configuração da integração entre Testlink e Redmine. ....	103
Ilustração 24 – Criação do projeto no Redmine .....	104
Ilustração 25 – Usuários criados e seus papéis no projeto .....	104
Ilustração 26 – Cadastro campo customizado para requisitos no Testlink.....	105
Ilustração 27 – Palavras-chave para classificação dos casos de teste .....	106
Ilustração 28 – Cadastro do projeto de teste no Testlink .....	106
Ilustração 29 – Plataformas cadastradas no Testlink.....	107
Ilustração 30 – Cadastro do plano de teste no software Testlink .....	108
Ilustração 31 – Plano de teste com suíte de verificação no Testlink .....	109

Ilustração 32 – Caso de teste para verificação de requisitos e análise e seus passos .....	110
Ilustração 33 – Atribuição dos testes de verificação ao responsável .....	112
Ilustração 34 – Simulação de execução dos testes de verificação.....	113
Ilustração 35 – Criação do bug no Redmine .....	114
Ilustração 36 – Relação entre teste de verificação no Testlink e bug no Redmine ..	114
Ilustração 37 – Status atualizado do <i>bug</i> no Testlink .....	115
Ilustração 38 – Histórico de verificações do artefato .....	115
Ilustração 39 – Organização dos requisitos no software Testlink.....	116
Ilustração 40 – Criação dos requisitos no software Testlink.....	117
Ilustração 41 – Estrutura do plano de testes .....	118
Ilustração 42 – Estrutura do caso de teste no Testlink.....	119
Ilustração 43 – Atribuição dos casos de teste às plataformas.....	120
Ilustração 44 – Atribuição dos casos de teste aos seus executores .....	120
Ilustração 45 – Histórico de passos da verificação da documentação do projeto ...	121
Ilustração 46 – Execução do caso de teste de integração .....	122
Ilustração 47 – Métricas gerais de casos de teste.....	123
Ilustração 48 – Métricas gerais de testes de verificação .....	123
Ilustração 49 – Métrica por plataforma: Teste automatizado.....	123
Ilustração 50 – Histórico de execução do caso de teste criar usuários .....	125
Ilustração 51 – Bug criado no Redmine para o erro ao criar o usuário .....	126
Ilustração 52 – Bugs por Caso de Teste .....	127
Ilustração 53 – Resultados parciais por suíte de teste .....	127
Ilustração 54 – Resultados parciais por plataforma.....	128
Ilustração 55 – Resultados parciais por palavra-chave .....	128
Ilustração 56 – Gráfico de resultados parciais por palavra-chave .....	128
Ilustração 57 – Bugs totais por palavra-chave.....	129
Ilustração 58 – Histórico de execução do teste de performance .....	130
Ilustração 59 – Não conformidades por caso de teste e por status.....	131
Ilustração 60 – Não conformidades por plataforma.....	132
Ilustração 61 – Métrica parcial por palavra-chave .....	133
Ilustração 62 – Métrica parcial por entrega .....	134
Ilustração 63 – Relatório de bugs por caso de teste .....	134
Ilustração 64 – Relatório de bugs por palavra-chave .....	135

Ilustração 64 – Gráfico de bugs por palavra-chave .....	135
---	-----

## **LISTA DE TABELAS**

Tabela 1 - Requisitos das ferramentas a serem analisadas.....	87
Tabela 2 - Verificação do atingimento dos requisitos na ferramenta Testlink.....	95



# SUMÁRIO

<b>1</b>	<b>INTRODUÇÃO .....</b>	<b>12</b>
1.1	O PROBLEMA DE PESQUISA .....	13
1.2	QUESTÃO DE PESQUISA .....	14
1.3	OBJETIVO DO TRABALHO .....	14
1.4	METODOLOGIA .....	15
1.5	ESTRUTURA DO TRABALHO.....	15
<b>2</b>	<b>QUALIDADE DE SOFTWARE .....</b>	<b>17</b>
2.1	QUALIDADE .....	17
2.1.1	Qualidade de software .....	18
2.1.2	Planejamento da qualidade.....	19
2.1.3	Controle de qualidade .....	19
2.2	<b>GARANTIA DA QUALIDADE DE SOFTWARE .....</b>	<b>19</b>
2.2.1	Custo da Qualidade .....	20
2.2.2	Onde deve ser aplicada a garantia.....	22
2.2.3	Adequação do modelo de qualidade ao custo.....	24
2.3	PROCESSO DE GARANTIA DA QUALIDADE DE SOFTWARE .....	25
2.4	VERIFICAÇÃO E VALIDAÇÃO .....	26
2.4.1	Verificação .....	26
2.4.2	Validação .....	27
2.4.3	Revisões de requisitos, análise e projeto .....	28
2.5	TESTES DE SOFTWARE .....	33
2.5.1	Testes de unidade .....	34
2.5.2	Testes de integração .....	35
2.5.3	Testes de sistema.....	37
2.5.4	Testes de aceite .....	38
2.6	CATEGORIA DE TESTES .....	39
2.6.1	Teste de funcionalidade.....	39
2.6.2	Teste de desempenho .....	39
2.6.3	Teste de usabilidade .....	40
2.6.4	Teste de estresse.....	41
2.6.5	Teste de segurança .....	41
2.6.6	Teste de configuração.....	42

2.7	ESTRATÉGIA DE TESTE .....	43
2.7.1	Teste caixa-branca .....	43
2.7.2	Teste caixa preta.....	45
2.7.3	Qual estratégia deve ser utilizada? .....	46
2.8	DOCUMENTAÇÃO DOS TESTES.....	47
2.8.1	Casos de teste .....	47
2.8.2	Plano de teste .....	48
2.9	MÉTRICAS.....	49
2.10	Considerações .....	52
3	<b>CENÁRIO ATUAL DO LABORATÓRIO DE CRIAÇÃO E APLICAÇÃO DE SOFTWARE.....</b>	<b>53</b>
3.1	PROCESSO ATUAL .....	55
3.2	PROBLEMA .....	61
3.3	CONSIDERAÇÕES.....	61
4	<b>PROPOSTA DE NOVO PROCESSO .....</b>	<b>62</b>
4.1	PROCESSO DE CRIAÇÃO E PLANEJAMENTO DO PROJETO PROPOSTO.....	63
4.1.1	Reunião com o cliente e análise inicial .....	63
4.1.2	Do levantamento de requisitos ao planejamento das tarefas e envolvidos 64	
4.1.3	Verificação da documentação de análise e requisitos.....	65
4.1.4	Definição do plano de teste .....	68
4.1.5	Definição dos casos de teste.....	70
4.1.6	Criação do projeto de implantação até documentação do projeto .	71
4.1.7	Verificação da documentação de projeto .....	71
4.1.8	Validação do projeto com o cliente e realização de ajustes .....	72
4.2	PROCESSO DE DESENVOLVIMENTO .....	72
4.2.1	Testes de unidade .....	74
4.3	PROCESSO DE HOMOLOGAÇÃO .....	75
4.3.1	Executar os testes de integração.....	76
4.3.2	Executar os testes de sistema – Equipe de testes .....	78
4.3.3	Executar os testes de sistema - GTI .....	80
4.3.4	Executar os testes de aceite.....	81
4.4	processos de implantação e encerramento.....	83
4.5	CONSIDERAÇÕES FINAIS .....	84

<b>5</b>	<b>ESCOLHA DA FERRAMENTA E APLICAÇÃO EM UM PROJETO PROTÓTIPO</b>	<b>86</b>
5.1	ESCOLHA DAS FERRAMENTAS.....	86
5.1.1	Metodologia de escolha .....	87
5.1.2	Requisitos das ferramentas.....	87
5.1.3	Projeto da avaliação .....	93
5.1.4	Avaliação, pesquisa e definição de ferramentas .....	94
<b>6</b>	<b>IMPLANTAÇÃO DAS FERRAMENTAS E APLICAÇÃO EM UM PROJETO PROTÓTIPO .....</b>	<b>100</b>
6.1	INSTALAÇÃO E CONFIGURAÇÃO DAS FERRAMENTAS.....	100
6.2	PROJETO BOLSA DE VALORES.....	101
6.3	APLICAÇÃO DO PROJETO .....	102
<b>7</b>	<b>Conclusão .....</b>	<b>137</b>
<b>8</b>	<b>REFERÊNCIAS .....</b>	<b>140</b>

## 1 INTRODUÇÃO

Vive-se em um contexto em que a tecnologia está cada vez mais presente na vida das pessoas. Com a evolução da tecnologia, não só no trabalho é possível a utilização de sistemas informatizados, mas a vida pessoal e social das pessoas está cada vez mais ligada com a informática.

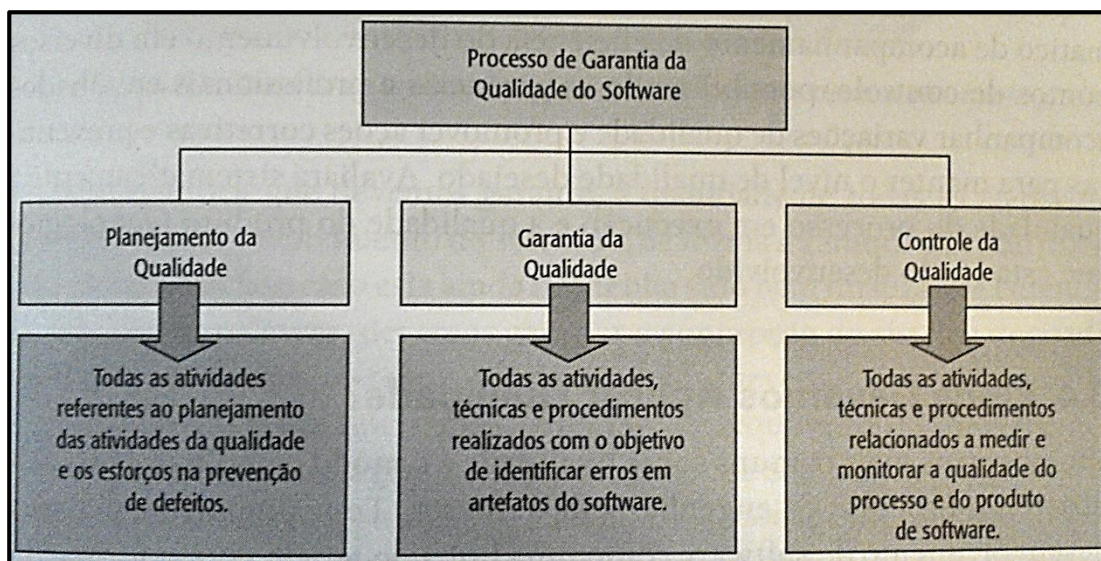
Conforme Pressman (2011), o clamor por maior qualidade de software começou realmente quando o software passou a se tornar cada vez mais integrado em todas as atividades de nossas vidas. Diante da evolução da tecnologia, que ocorre no intuito de atender estas crescentes inovações, e estas criam expectativas cada vez mais avançadas nas pessoas, a exigência de softwares mais confiáveis e que cumprem fielmente o que prometem evolui também, tornando indispensável a utilização de processos que qualifiquem mais o produto final por parte da entidade desenvolvedora da aplicação. Identificando isto, as organizações procuram evoluir na qualidade de seus produtos, tentando garantir a confiabilidade dos usuários.

O aumento na complexidade das aplicações traz consigo um aumento do número de variáveis envolvidas. Segundo Bartié (2002), “Nesse cenário todas as variáveis envolvidas no processo de desenvolvimento de software têm um nível crescente de complexidade. Com isso, os riscos de mau funcionamento aumentam proporcionalmente à complexidade desse ambiente, tornando-se mais difícil produzir softwares com um nível de qualidade desejado”.

Como será demonstrado ao longo do trabalho e conforme afirmação de Tsukumo, Rêgo, Salviano e outros, a qualidade de software é determinada muito pela qualidade dos processos utilizados para o desenvolvimento. Deste modo, a melhoria da qualidade de software é obtida pela melhoria da qualidade dos processos.

O PMI (*Project Management Institute*), elaborador do PMBOK (*Project management body of knowledge*), definiu que a garantia da qualidade do software é subdividida em três pilares, conforme figura a seguir.

Ilustração 1 - Pilares do processo de garantia da qualidade de software



Fonte: Teste de software, Bartiè (2002).

Ainda segundo Bartiè (2002), ter um processo de garantia da qualidade em todo o ciclo de desenvolvimento do software permite que um número maior de defeitos sejam descobertos antecipadamente, evitando a migração destes para as fases seguintes. Sendo assim, existirá um processo e um produto mais estáveis, reduzindo o retrabalho sobre o projeto, aumentando a satisfação da equipe interna e do cliente.

### 1.1 O PROBLEMA DE PESQUISA

A UCS – Universidade de Caxias do Sul é uma organização acadêmica com mais de 25 anos em cursos de tecnologia da informação. Nos últimos anos, procura atuar também no desenvolvimento de ferramentas que auxiliem os alunos nas interações com a universidade além de realizar desenvolvimentos em parceria com outros cursos.

Em março de 2015 estabeleceu-se um novo setor dentro do centro de tecnologia, visando o desenvolvimento de software para a universidade. Diante de sua recente criação, o laboratório de software, denominado “laboratório de criação e aplicação de software”, necessita de um plano de qualidade ainda não definido para

tornar o processo mais confiável, trazendo maiores garantias ao resultado final dos projetos implementados pelas equipes e que seja aderente à forma de trabalho do setor.

## 1.2 QUESTÃO DE PESQUISA

Como garantir um nível satisfatório de qualidade de software para o laboratório de criação e aplicação de software de forma a garantir uma maior confiabilidade ao resultado final dos projetos?

## 1.3 OBJETIVO DO TRABALHO

O objetivo deste trabalho é analisar o processo atual do laboratório de criação e aplicação de software e propor um plano de garantia de qualidade de software fundamentado e aderente à realidade. Assim espera-se que os projetos sejam desenvolvidos seguindo fielmente os requisitos propostos pelos usuários, para que as ferramentas sejam disponibilizadas com o menor número de falhas possíveis além de diminuir os custos com retrabalho por antecipação das situações em que as ferramentas podem ser colocadas.

O plano deve ser simples e aplicável à realidade e características do laboratório. Para que a gestão do plano de garantia da qualidade de software seja aprimorada, foram propostas ferramentas que auxiliem os responsáveis e os executores da gestão das tarefas do plano de qualidade. Estas ferramentas devem atender às tarefas propostas e propiciar uma melhora na gestão do projeto de software. Esta melhora deve ser perceptível tanto na atribuição das tarefas quanto no acompanhamento do projeto e na coleta dos resultados.

## 1.4 METODOLOGIA

Através da bibliografia do gênero, realizar-se-á um estudo profundo das características de um processo de qualidade de software, dos indicadores de qualidade e dos diferentes processos que fazem o desenvolvimento de sistemas se tornar melhor no que tange a qualidade.

Estudo das práticas sugeridas por entidades de engenharia de software, como no caso da CMMI, MPS.BR, IEEE, a fim de se determinar os processos fundamentais que a organização necessita.

Através de entrevistas e análises presenciais, determinar o processo atual do laboratório de software da universidade além da expectativa dos colaboradores com a implantação de um processo de qualidade. Diante disso, identificar os indicadores que tornarão a análise da evolução dos projetos de desenvolvimento possível, além dos indicadores que demonstrarão a evolução da equipe com a adoção do processo proposto.

Através de pesquisas em bibliografia e casos de implantação semelhantes, estudar e descrever os processos de qualidade de software mais aderentes à realidade e às expectativas da organização.

Por meio de artigos acadêmicos e trabalhos correlatos, pesquisar e sugerir as ferramentas necessárias para que cada tarefa sugerida possa ser gerida de forma satisfatória.

Por meio de um projeto protótipo, demonstrar a efetividade das ferramentas sugeridas na gestão do plano de garantia da qualidade de software em um cenário muito próximo do real.

## 1.5 ESTRUTURA DO TRABALHO

O capítulo 2 descreve o que é qualidade de software, em que momento deve ser aplicada, de que forma ela interfere no custo de um projeto e o custo-benefício de se implantar um plano de garantia de qualidade de software.

No capítulo 3 são apresentadas e discutidas as características necessárias e importantes em um processo de garantia da qualidade de software bem como o seu método de execução e utilização e o momento em que deve ser aplicado tal processo.

Já o capítulo 4 descreve o cenário atual do laboratório, evidenciando o processo atual e as características próprias do setor e do processo e assim definindo algumas premissas para o processo proposto.

Por fim, no capítulo 5 é evidenciada a proposta de novo processo, incluindo um plano da garantia da qualidade de software, baseado em todo o estudo das características necessárias e das características próprias do laboratório de criação e aplicação de software.

O capítulo 6 traz a bibliografia utilizada como base deste trabalho a fim de evidenciar as obras citadas ao longo do texto.



## 2 QUALIDADE DE SOFTWARE

Com a crescente expansão da utilização de sistemas informatizados na maioria das atividades das pessoas a qualidade deixou de ser um diferencial dos produtos e passou a ser uma exigência. Conforme Pressman (2011) cita, a preocupação com a qualidade de sistemas de software cresceu à medida que o software passou a se tornar cada vez mais integrado em cada aspecto da vida cotidiana.

### 2.1 QUALIDADE

Definir qualidade não é simples. Conforme Pressman (2011) descreve, sabe-se o que é qualidade ao vê-la e, mesmo assim, pode ser algo difícil de definir. Crosby (1980) afirma que qualidade é a conformidade aos requisitos. Já David Garvin (1984), traz uma visão diferente, afirmando que qualidade é um conceito complexo e multifacetado e pode ser descrito segundo cinco pontos de vista diferentes.

- A visão transcendental, que sustenta que qualidade é algo que se reconhece imediatamente.
- A visão do usuário, que verifica se o produto atende a metas específicas. Se o faz, então demonstra qualidade.
- A visão do fabricante que define qualidade em termos da especificação original do produto.
- A visão do produto que sugere que a qualidade está ligada a características inerentes de um produto e, finalmente, a visão baseada em valor, que mede a qualidade tomando como base o quanto um cliente estaria disposto a pagar por um produto.

### 2.1.1 Qualidade de software

Trazendo esta definição para o desenvolvimento de software, Pressman (2011) define como uma gestão de qualidade efetiva aplicada de modo a criar um produto útil que forneça valor mensurável para aqueles que produzem e para aqueles que o utilizam. Bartiè (2002) segue a mesma linha, assim como Koscianski e Soares(2006), afirmando que qualidade de software é um processo sistemático que focaliza todas as etapas e artefatos produzidos, dependendo do correto emprego das boas metodologias pelos colaboradores, com o objetivo de garantir a conformidade de processos e produtos, prevenindo e eliminando defeitos.

Pressman (2011) e Sommerville (2011), afirmam que a qualidade de software não aparece simplesmente do nada. Ela é o resultado de uma prática consistente de engenharia de software. Diante disto, atribuem o sucesso a aplicação de quatro grandes atividades:

- Métodos de engenharia de software
- Técnicas de gerenciamento de projeto
- Ações de controle de qualidade
- Garantia da qualidade de software.

Se a expectativa é construir software de alta qualidade, é dever da equipe desenvolvedora ser capaz de criar um projeto que seja adequado ao problema e, ao mesmo tempo, apresente características que levem a um software com as dimensões e fatores de qualidade.

O impacto das decisões de gerenciamento inadequado sobre a qualidade de software é grande. As datas devem ser plausíveis, as dependências de cronograma devem ser adequadas ao projeto e a equipe, o planejamento de riscos deve ser coerente, dentre outros fatores determinantes. Devido a isto, a capacidade de gerenciamento do projeto é determinante na qualidade do software proposto.

Portanto, evidencia-se que qualidade de software só é alcançada com um processo que seja voltado para ela. A forma de alcance desta qualidade será descrita durante os próximos capítulos.

### **2.1.2 Planejamento da qualidade**

Um processo adequado de desenvolvimento de software deve planejar suas atividades de modo a deixar claro para todos os envolvidos os objetivos do projeto e o que se deve realizar para garantir que as metas sejam atingidas conforme os parâmetros estabelecidos. Bartiè (2002) afirma que planejamento da qualidade é o processo destinado a identificar quais padrões de qualidade são relevantes para o projeto e determinar como satisfazê-los. É realizado juntamente com outros processos de planejamento e tem como objetivo um plano que garanta uma estratégia de qualidade que atinja os objetivos propostos.

### **2.1.3 Controle de qualidade**

Para que se garanta a qualidade de software, deve existir um controle de qualidade. Para Bartiè (2002), controle de qualidade se concentra no monitoramento e desempenho dos resultados do projeto para determinar se ele está atendendo aos padrões de qualidade no processo de desenvolvimento.

Pressman (2011) descreve que o controle de qualidade engloba um conjunto de ações de engenharia de software que ajudam a garantir que cada produto resultante atinja suas metas de qualidade. Os modelos devem ser revistos de modo a garantir que sejam completos e consistentes. O código deve ser inspecionado de modo a revelar e corrigir erros antes dos testes iniciarem, entre outras técnicas que serão descritas neste trabalho.

## **2.2 GARANTIA DA QUALIDADE DE SOFTWARE**

Como é percebido, qualidade de software é algo que se garante ao longo de todo o processo de desenvolvimento de um sistema. Devido a isto, foi necessária a criação de um novo conceito, a garantia da qualidade de software.

Garantia da qualidade de software é o processo que engloba a estruturação, sistematização e execução das atividades que terão como objetivo garantir o adequado desempenho de cada etapa do desenvolvimento, satisfazendo os padrões de qualidade definidos no processo, conforme afirma Bartiè (2002).

Pressman (2011), de uma forma mais detalhada, acrescenta que a garantia de qualidade estabelece a infraestrutura que suporta métodos sólidos de engenharia de software, gerenciamento racional de projeto e ações de controle de qualidade. Além disso, a garantia da qualidade consiste em um conjunto de funções de auditoria e de relatórios que possibilita uma avaliação da efetividade e da completude das ações de controle da qualidade. Isso, pois, conforme Sommerville (2011) afirma, qualidade de software é diretamente relacionada à qualidade do processo de desenvolvimento de software.

A ISO, através da família de normas 9000, adiciona aos conceitos já apresentados que garantia de software é a aplicação de qualidade aos produtos de forma a assegurar ao cliente que a empresa fornecedora tem capacidade de atender aos requisitos com qualidade. Portanto, já fica evidenciado, para se alcançar qualidade de software deve-se ter qualidade no processo inteiro de desenvolvimento do sistema.

### **2.2.1 Custo da Qualidade**

Sabe-se que a qualidade possui um custo financeiro e de tempo. De qualquer forma a sua ausência também. Conforme Sommerville (2011) relata, o custo da qualidade inclui todos os custos necessários com a busca de qualidade ou para execuções relacionadas à qualidade, assim como os custos causados pela falta de qualidade. Bartiè (2002) ainda cita que fazem parte também deste tipo de visão os custos da detecção de defeitos, os custos relacionados a todas as atividades que visam garantir e detectar problemas nos diversos produtos gerados durante o ciclo de desenvolvimento e os custos de prevenção de defeitos.

Conforme Sommerville (2011) e Pressman (2011) detalham, os custos podem ser divididos em custos associados à prevenção, avaliação e falhas. De maneira

geral, custos de prevenção incluem o custo de atividades de gerenciamento necessárias para planejar e coordenar todas as atividades de controle e garantia da qualidade. Além disso, incluem o custo de atividades técnicas adicionais para desenvolver modelos completos de requisitos e de projeto, custos de planejamento de testes e o custo de todo o treinamento associado a essas atividades.

Os custos de avaliação incluem atividades para a compreensão aprofundada da condição do produto em cada fase do processo. Portanto podem incluir o custo de realização de revisões técnicas, o custo para coleta de dados e avaliação de métricas e o custo de testes e depuração do processo.

Os custos de falhas são os custos que ocorrem pela falta da identificação dos problemas em um momento anterior do processo fazendo com que erros ocorram depois da entrega do produto ao cliente. Diante disso, os custos de falhas são compostos pelo custo para se realizar retrabalhos para correção de erros, os custos que os retrabalhos geram e os custos relacionados ao gerenciamento dos retrabalhos, conforme Sommerville (2011) cita. Além disso, Pressman (2011) cita ainda os custos de falhas externas, que estão associados às consequências associadas à falha. Alguns exemplos disto são reclamações, devolução e substituição dos produtos, além do suporte do produto. Além disso, uma falha ou a sucessão de falhas pode causar uma má reputação e a consequente queda de negócios, que apesar da dificuldade de quantificá-los, deve estar presente na análise para determinação da validade da aplicação de um processo de qualidade de software.

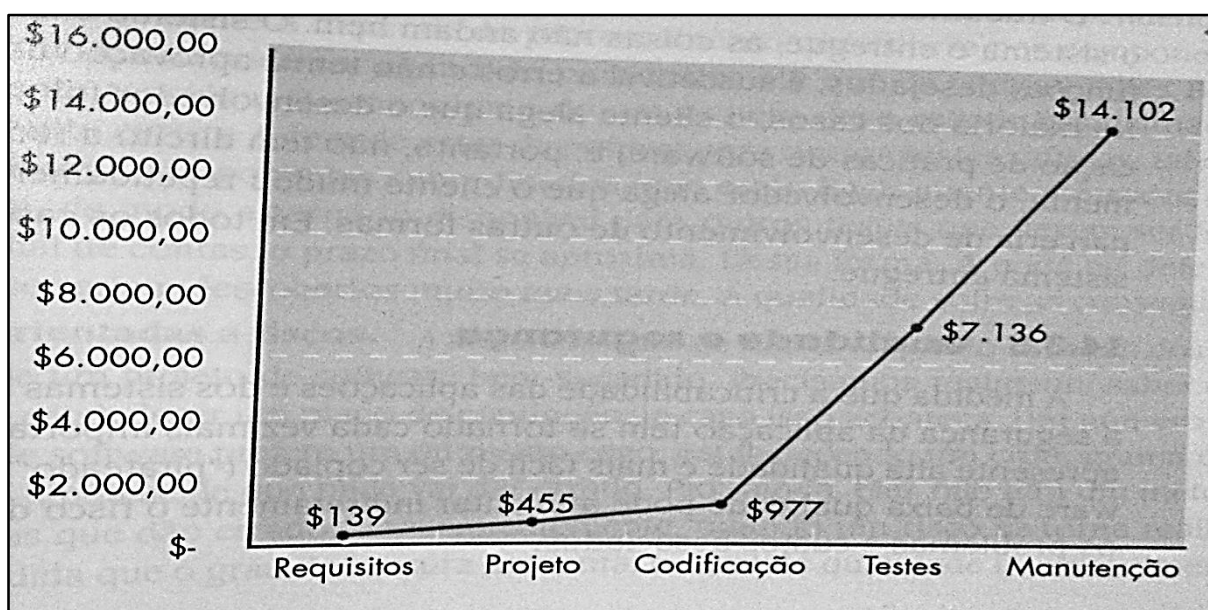
Sommerville (2011) evidencia que os custos relativos para descobrir e corrigir problemas aumentam conforme o processo do desenvolvimento de software progride. Pressman (2011), em sua bibliografia, traz estatísticas da Cigital Inc., que compara a média de custo para correção de erros em cada fase com uma quantidade limitada de erros a fim de evidenciar a importância da identificação de falhas no começo do processo.

De acordo com os dados médios das empresas de desenvolvimento utilizadas como base, o valor de correção de problemas na fase de codificação é de US\$ 977. Se em determinado projeto forem identificados 200 problemas nesta fase, o custo total para correção seria de aproximadamente US\$ 195.400. Já, se identificado um

erro na fase de testes, o valor médio por correção é de aproximadamente US\$ 7.136. Tomando como base o mesmo número de erros, o custo para correção deste montante seria de US\$ 2.472.100.

Isto torna evidente como a existência de métodos de averiguação de erros durante todo o processo, do começo ao final, reduz consideravelmente o custo com a resolução de problemas. Abaixo segue gráfico com o custo relativo de correção de erros e defeitos.

Ilustração 2 – Custo de alteração em um projeto de software



Fonte: Engenharia de software, Pressman (2011)

### 2.2.2 Onde deve ser aplicada a garantia

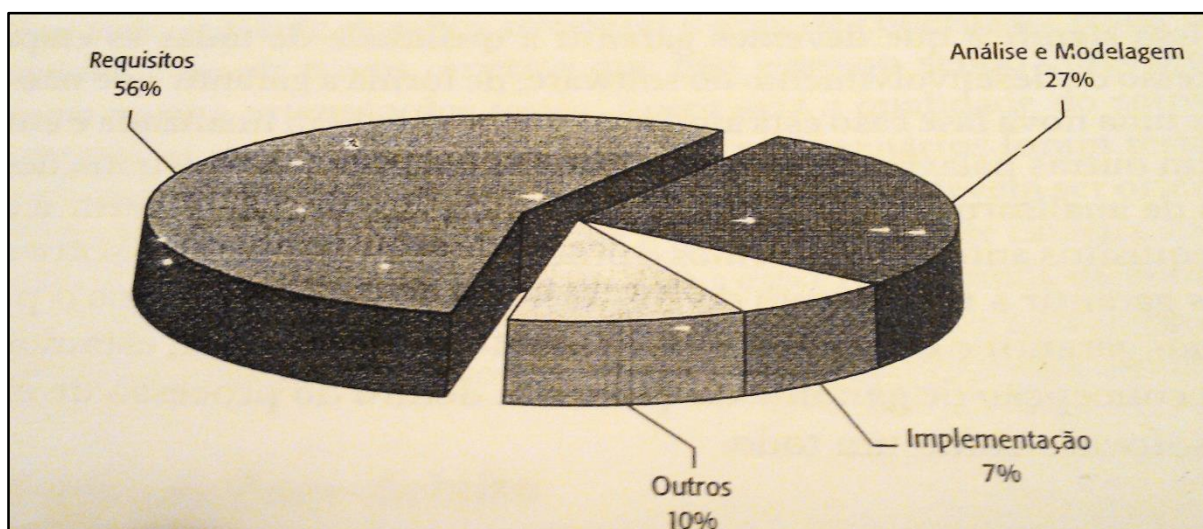
Em um primeiro momento, pessoas não muito familiarizadas com os métodos e processos existentes em engenharia de software, tendem a determinar que a qualidade em um processo de garantia da qualidade deva existir apenas na fase de testes do projeto, ou seja, após a conclusão de toda a codificação estipulada. Isto costuma ocorrer pois o processo é pensado como uma linha do tempo e está incorreto, já que fere a ideia de identificar os erros nas fases iniciais do processo de desenvolvimento de software, evitando que os erros sejam encontrados apenas no

final do processo e assim aumentando os custos de sua identificação e correção, como foi já mencionado anteriormente.

Isto significa que deve existir qualidade em cada etapa do processo para que a próxima etapa não herde erros de fases anteriores. Conclui-se então que a redução dos erros somente ocorrerá se em todas as etapas do processo de desenvolvimento existir procedimentos que avaliem a qualidade do que foi produzido.

Sabendo que erros ocorrem nas mais diversas partes do processo de desenvolvimento de software, elaboraram-se alguns estudos para se determinar quais fases eram responsáveis pela maior incidência de erros. O estudo, demonstrado por Bartiè (2002), comprovou que, ao contrário do que muitos pensam, a maior quantidade de erros não é causada pela fase da codificação e sim pelas fases iniciais. Isto ocorre pois geralmente o erro está na interpretação imperfeita dos requisitos e especificações pelos responsáveis, desencadeando falhas ao longo do projeto. Abaixo se evidencia o gráfico do estudo citado por Bartiè (2002), ilustrando e determinando as fases de maior incidência de problemas.

Ilustração 3 – Incidência de erros por fase



Fonte: Teste de software, Bartiè 2002.

Portanto, assim como Bartiè (2002), Konscianski e Soares (2006) também afirmam que ter um processo de garantia da qualidade em todo o processo de desenvolvimento permite que um maior número de falhas sejam descobertas antecipadamente, reduzindo os custos dos projetos em vários aspectos. É fácil também concluir que antecipando a identificação das falhas, também haverá uma redução no tempo total de desenvolvimento de cada projeto, já que existirá um menor tempo para retrabalhos. Cliente e equipe interna ficarão mais satisfeitas. O primeiro por ter uma aplicação mais estável e receber seus projetos de forma mais rápida e a segunda por trabalhar menos com retrabalhos e mais com atividades construtivas, que agregam valor às soluções, fortalecendo assim a organização como um todo.

### **2.2.3 Adequação do modelo de qualidade ao custo**

Conforme já evidenciado, um processo de qualidade completo é de suma importância para o processo de desenvolvimento de software. Porém implantar um processo completo em uma organização pode ser muito custoso, já que envolve certo número de recursos profissionais e tempo para isto, conforme Bartiè (2002).

Alguns aspectos devem ser levados em consideração para se determinar o nível de aprofundamento do processo na organização, como a maturidade da organização, ou seja, o nível de estruturação do processo de desenvolvimento é suficientemente bom para termos um grande processo de qualidade? Além destes aspectos, deve-se verificar a existência de documentos e produtos em cada uma das etapas, o nível de informalidade e de especificações.

Portanto, para a definição de um processo adequado e aderente ao processo e que não onerará o desenvolvimento com atividades dispensáveis, deve-se verificar as características do processo e dos recursos envolvidos e dar mais ênfase nos softwares e etapas mais críticos e problemáticos, minimizando assim os riscos no desenvolvimento e utilizando um processo de qualidade efetivo.



## 2.3 PROCESSO DE GARANTIA DA QUALIDADE DE SOFTWARE

Como é notável e como Glenford J. Myers (2011) afirma, zero-defeito é algo inatingível, ou seja, pela complexidade envolvida e pelo número altíssimo de situações existentes, torna-se impossível imaginar um produto de software livre de erros. Portanto, sabe-se que qualquer software produzido gerará problemas e retrabalhos passíveis de correções. Mesmo assim, conforme já foi evidenciado anteriormente, quanto mais se posterga a identificação de inconsistências, maior será o custo de manutenção. E conforme o próprio gráfico de Bartiè (2002) evidenciou na ilustração 1, a maior incidência de erros ocorre no levantamento de requisitos, na análise e modelagem, processos iniciais e essenciais para a correta conceituação de um sistema informatizado.

Konscianski e Soares (2006) afirmam que construir certo da primeira vez é objetivo fixado por todas as engenharias. Diante disso, conforme Bartiè (2002) afirma, garantia da qualidade de software é o processo que engloba a estruturação, sistematização e execução das atividades que terão como objetivo garantir o adequado desempenho de cada etapa do desenvolvimento, satisfazendo os padrões de qualidade definidos no processo. Na garantia da qualidade de software, qualidade não é uma fase do ciclo de desenvolvimento de software, é parte de todas as fases. A redução dos erros somente ocorrerá se em todas as etapas do processo de desenvolvimento existir procedimentos que avaliem a qualidade do que foi produzido.

Não é possível conceber um processo de garantia da qualidade de software sem integrá-lo com o ciclo de desenvolvimento de software. Um bom processo de qualidade é aquele que cria uma relação direta entre as fases de desenvolvimento e as atividades a serem desempenhadas pela equipe de qualidade, conforme descrito por Bartiè (2002). Nos próximos itens deste capítulo, será demonstrado, portanto técnicas e processos para que seja possível aumentar a qualidade dos artefatos produzidos no planejamento e análise do processo bem como no decorrer do desenvolvimento.

## 2.4 VERIFICAÇÃO E VALIDAÇÃO

As atividades de verificação e validação servem para assegurar que o software funcione de acordo com o que foi especificado e atenda aos requisitos dos stakeholders, conforme afirma Konscianski e Soares (2006). Bartiè (2002) complementa que como cada etapa deve produzir um conjunto de documentos, é possível estabelecer a qualidade dos documentos produzidos. Esses testes são conhecidos como testes de verificação.

Sommerville (2011) destaca que as atividades de verificação e validação constituem um processo iniciado com as revisões dos requisitos, passando pelas revisões da análise e do projeto até chegar às verificações e testes. Além disso, afirma que o objetivo final dos processos de verificação e validação é estabelecer a confiança de que o software está pronto para seu propósito. Isto significa que o sistema deve ser bom o suficiente para o seu intuito. O nível de confiança exigido depende do propósito do sistema, das expectativas dos usuários do sistema e do atual ambiente de marketing.

A seguir, serão demonstrados de forma mais detalhada e específica a definição do que é verificação e validação e em que momento do processo eles devem ser utilizados.

### 2.4.1 Verificação

Konscianski e Soares (2006) em sua definição, afirmam que verificação é o processo de determinar se a saída de uma fase está de acordo com os requisitos especificados na fase anterior. Como cada etapa deve produzir um conjunto de documentos, é possível estabelecer a qualidade dos documentos produzidos através de um tipo determinado de teste. Esses testes são conhecidos como testes de verificação, conforme definição de Bartiè (2002) que prossegue: testes de verificação podem ser entendidos como um processo de auditoria de atividades e avaliação de documentos gerados durante cada uma das fases elencadas no

processo de engenharia de software. A principal característica dos testes de verificação é o fato de não envolver o processamento de softwares.

Ainda segundo Bartiè (2002), os testes de verificação devem ser sistematizados, planejados e devidamente aplicados, de forma a se tornarem parte integrante do processo de engenharia de software. Estes testes devem ser conduzidos por uma equipe independente da área de desenvolvimento, de forma a garantir maior eficiência e nível adequado de profundidade nesses trabalhos. Uma área ou pessoas independentes conduzirão este processo com total autonomia, sendo mediador entre o cliente e o fornecedor.

Portanto, através dos processos e atividades de verificação se alcança a qualidade dos artefatos produzidos antes do desenvolvimento do software, conseguindo a esperada antecipação de falhas e reduzindo os custos de uma definição incorreta ou incompleta.

#### **2.4.2 Validação**

Já a validação, conforme definição de Konsciński e Soares (2006), é o processo de confirmar que o sistema completo é apropriado e consistente com os requisitos dos stakeholders.

Para isto deve-se possuir um processo para determinar o que não está conforme a expectativa do cliente. Para Bartiè (2002) os testes de validação têm por objetivo identificar o maior número possível de erros tanto nos componentes isolados quanto na solução tecnológica como um todo e portanto conseguem determinar o que não está conforme as especificações.

O sucesso do processo de validação está apoiado diretamente em um forte planejamento de todas as atividades de testes, nas quais a concentração dos trabalhos de validação nos componentes mais complexos e nos requisitos mais críticos contribui para aumentar a eficiência dos procedimentos de detecção de defeitos, uma vez que esses componentes concentram a maior quantidade de erros. Contrário aos testes de verificação trazidos no item anterior, que avaliavam as documentações e atividades com o objetivo de detectar erros ou quebras do

processo, os testes de validação possuem agora um produto computacional que pode ser executado.

Konscianski e Soares (2006) complementam, afirmando que a validação consiste, então, em identificar defeitos e possíveis problemas de um componente pronto.

### **2.4.3 Revisões de requisitos, análise e projeto**

A revisão é um processo essencial para garantia de continuidade de um processo bem realizado. Conforme Bartiè (2002) define, a revisão é um processo humano de análise de determinado documento e portanto fazem parte do processo de verificação. Esse processo de revisão requer pessoas adequadamente treinadas para desempenhar seus papéis durante a condução das atividades de verificação. Esta etapa tem importância fundamental no processo de qualidade, pois focaliza suas ações nas etapas iniciais do projeto, naquelas que geralmente ocorre maior incidência de defeitos e que estes são mais custosos para uma correção posterior, geralmente associados a definições erradas ou a falhas nas decisões realizadas.

Nem sempre a fase de modelagem de negócios é convenientemente explorada nos projetos de desenvolvimento e devido a este fato a revisão dos artefatos produzidos ao longo desta fase é tão importante. Conforme Paula Filho afirma, a revisão dos requisitos tem por objetivo assegurar que a especificação dos requisitos do software esteja conforme com o respectivo padrão, e com outros padrões aplicáveis ao projeto em questão, atenda aos critérios de qualidade dos requisitos, forneça informação suficiente para o desenho do produto, de seus testes de aceitação e do seu manual de usuário. Inicialmente, os requisitos são revistos informalmente pelos participantes do levantamento.

Conforme Bartiè (2002) relata, revisões podem ser individuais ou em equipes. Vale salientar que independentemente do tipo de revisão utilizado, revisões devem ser realizadas preferencialmente por uma pessoa diferente daquela que desenvolveu o artefato sendo examinado já que, conforme complementam

Konscianski e Soares (2006), o autor do trabalho tem mais dificuldades para encontrar erros.

Cada fase do projeto de software cumpre uma importante etapa do entendimento do problema e da definição de uma solução mais adequada às necessidades do cliente. Diante deste tipo de constatação Bartiè (2002) afirma que existe uma série de verificações principais e diferentes entre si a serem realizadas em cada fase do projeto, visto que cada uma delas difere em suas características. Estas verificações serão mais bem evidenciadas a partir do próximo parágrafo.

Para o processo de verificação do modelo de negócios, deve-se revisar o contexto do mercado e as necessidades do cliente, revisar os riscos do projeto, auditar as alternativas de execução do projeto e revisar o estudo de viabilidade do projeto. Para isto deve-se avaliar se todas as necessidades metas e exigências foram listadas, além de verificar se a modelagem de negócios cobre todas as necessidades. Confere-se também se as projeções realizadas são baseadas em métricas e indicadores confiáveis, além de avaliar a existência de alternativas para esta mesma solução, o retorno sobre o investimento do projeto e as opções de investimento.

Já para o caso da especificação de requisitos, os revisores deverão concentrar-se em identificar se todos os requisitos funcionais e não funcionais do software estão listados e completos, tornando o entendimento dos mesmos e em sua completude possível. Conforme Sommerville (2011) afirma, os requisitos funcionais descrevem o que um sistema deve fazer. Já os requisitos não funcionais são requisitos não diretamente relacionados com os serviços específicos de um sistema, como são os casos de desempenho, proteção ou disponibilidade.

Pressman (2011) afirma que os objetivos primários da análise de requisitos são:

- Descrever o que o cliente solicita;
- Estabelecer uma base para a criação de um projeto de software;
- Definir um conjunto de requisitos que possa ser validado assim que o software for construído.

Sommerville (2011), como complemento, escreve que os requisitos devem descrever o comportamento externo do sistema e suas restrições operacionais. Diante disso, Bartiè (2002) afirma que na verificação de requisitos, devem ser verificadas as consistências dos requisitos funcionais, verificar os requisitos não funcionais e verificar a rastreabilidade entre requisitos e necessidades.

Durante a verificação de requisitos deve-se atentar se cada requisito está claro o suficiente para que não produza uma interpretação errada sobre sua real intenção, o que provocaria falha na concepção do produto. Além disso, deve-se revisar a priorização de requisitos e a sua rastreabilidade. Os autores costumam evidenciar algumas características críticas que devem ser consideradas pelos revisores. A definição das características abaixo é de Bartiè (2002):

- Completo: Todos os requisitos do projeto devem estar documentados;
- Claro: Cada requisito deve expressar seu propósito em relação ao projeto;
- Simples: Cada requisito deve ser exato, não dar margens a dúvidas;
- Consistente: Cada requisito não deve possuir conflitos com outros requisitos existentes.
- Relevante: Cada requisito deve pertencer ao escopo do projeto em questão;
- Testável: Cada requisito deverá fornecer informações que possibilitem quantificar se um determinado item foi adequadamente implementado;
- Factível: Cada requisito deve ser viável em sua implementação, avaliando as condições financeiras, humanas e tecnológicas disponíveis no projeto.

Konscianski e Soares (2006) relatam que para cada requisito elencado devem existir parâmetros claros para atingimento da meta. Por exemplo: na afirmação “As consultas a pedidos deverão ter uma boa performance”, deve ser papel do autor do artefato indicar quanto tempo compreende esta boa performance. Caso o autor não o tenha feito, é papel do revisor relatar a ausência de informação. Estes parâmetros devem estar claros para que os testadores possam compará-los aos resultados reais e produzir as métricas do software.

Para a análise e a modelagem, conforme Bartiè (2002) revisa-se a arquitetura da aplicação, o modelo estático do projeto de software, o modelo dinâmico do projeto de software, o nível de componentização e o nível de reutilização.

Modelagem de sistema é o processo do desenvolvimento de modelos abstratos de um sistema, em que cada modelo apresenta uma visão de perspectiva diferente do sistema, conforme definição de Sommerville (2011). Ele continua, com a afirmação de que geralmente esta representação é gráfica, através de diagramas UML. Os cinco principais tipos de diagramas produzidos por analistas que utilizam a UML como forma de análise são:

- Diagramas de atividades, que mostram as atividades envolvidas em um processo ou no processamento de dados;
- Diagramas de casos de uso, que mostram as interações entre um sistema e seu ambiente;
- Diagramas de sequência, que mostram as interações entre os atores e o sistema, e entre os componentes do sistema;
- Diagramas de classe, que mostram as classes de objeto no sistema e as associações entre elas;
- Diagramas de estado, que mostram como o sistema reage aos eventos internos e externos.

Para verificação destes diagramas, Bartiè (2002) afirma que na etapa de revisão de análise o objetivo é definir uma solução tecnológica que suporte não somente os requisitos estabelecidos pelo cliente, mas também requisitos de qualidade que deverão ser atendidos pela arquitetura do software a ser modelada. A modelagem de uma boa arquitetura deve contemplar características como flexibilidade, escalabilidade, ser reutilizável. Então é essencial que, além disso, exista consistência na arquitetura da solução, que sejam aderentes os requisitos funcionais com a solução, assim como os não funcionais também.

Conforme Pressman (2011) descreve, o gerenciamento de projetos de software tem o foco nos 4 P's: Pessoas, produto, processo e projeto. Por isto a preocupação com ele deve ser grande. Sommerville (2011) diz que o gerenciamento de projetos é uma parte essencial da engenharia de software. Os projetos precisam ser gerenciados, pois a engenharia de software profissional está sempre sujeita a orçamentos organizacionais e restrições de cronograma. Os critérios para o sucesso do gerenciamento de projetos variam de um projeto para outro. Mas para a maioria dos projetos as metas mais importantes são:

- Fornecer o software ao cliente no prazo estabelecido;
- Manter os custos gerais dentro do orçamento;
- Entregar software que atenda às expectativas do cliente;
- Manter uma equipe de desenvolvimento que trabalhe bem e feliz.

Para isto sugere-se que sejam verificados os pontos imprescindíveis de projeto, que conforme a definição de Pressman (2011) podem ser:

- Definição adequada do plano de projeto;
- Definição e documentação dos marcos do projeto;
- Definição e documentação de recursos;
- Definição e documentação de riscos;
- Definição e documentação do cronograma;
- Definição de comunicação entre os recursos envolvidos.

Segundo diferentes autores, como Pressman (2011) e Koscianski e Soares uma importante abordagem de verificação é a reunião de verificação. Nela participarão um grupo de pessoas aptas a julgar se cada requisito está corretamente listado e abordado. Bartiè (2002) recomenda que este grupo seja heterogêneo, com profissionais de diferentes áreas, habilidades e especialidades, permitindo que as análises e discussões sejam aplicadas sobre várias perspectivas diferentes. No planejamento, deve ser considerado o número de profissionais que desempenharão os papéis de revisores dos documentos. O dimensionamento será em função da complexidade, importância e impacto que esse documento poderá trazer a determinadas áreas da organização, além do custo previsto para execução do projeto.

Realizado o planejamento das reuniões, deve-se preparar os participantes, fazendo com que recebam o máximo de informações possíveis sobre o tema a ser discutido na reunião e cabe ao organizador de tal fase este tipo de responsabilidade.

Durante as revisões, o profissional que conduz as reuniões deve garantir que todos os pontos dos documentos foram questionados e avaliados pelos revisores. A condução de um processo de verificação segue uma sequência lógica e simples. As principais atividades a serem executadas são:



- Um tópico é definido e será escopo das discussões;
- Uma questão é levantada por um revisor;
- A questão é discutida e avaliada;
- Os revisores confirmam a existência do defeito;
- O defeito é registrado e detalhado para que seja corrigido pelos autores;
- Outras questões são levantadas até que todas tenham sido analisadas;
- Um novo tópico é identificado até que todos tenham sido discutidos.

Sugere-se a utilização de checklists nesta fase do projeto para as verificações, conforme Konscianski e Soares (2006). Conforme Bartiè (2002), para cada documento diferente verificado por um processo de qualidade deve ser efetuado um checklist diferente. Neles devem estar uma identificação da fase a ser verificada, o critério a ser verificado e se está conforme ou não conforme, além de demais informações gerenciais necessárias, como projeto em questão, data, revisores, entre outros itens.

## 2.5 TESTES DE SOFTWARE

Constituintes do processo de validação de software e parte fundamental de qualquer processo de qualidade de software, os testes devem ser realizados para que se obtenha um produto confiável. Conforme Pressman (2011) define, teste é um conjunto de atividades que podem ser planejadas com antecedência e executadas sistematicamente e é destinado a mostrar que um programa faz o que é proposto a fazer e para descobrir os defeitos do programa antes de seu uso. Os resultados do teste são verificados à procura de erros, anomalias ou informações sobre os atributos não funcionais do programa.

IEEE define testes como o processo de avaliar um sistema ou um componente de um sistema por meios manuais ou automáticos para verificar se ele satisfaz os requisitos especificados ou identificar diferenças entre resultados esperados e obtidos. Os testes podem ser direcionados para alguns casos. Konscianski e Soares (2006) afirmam que o objetivo do teste é encontrar defeitos, revelando que o funcionamento do software em uma determinada situação não está

de acordo com o esperado. Um teste bem-sucedido identifica defeitos que ainda não foram descobertos e que podem ser, então, corrigidos pelo programador. Já para Sommerville (2011), os testes têm dois objetivos distintos: demonstrar ao desenvolvedor e ao cliente que o software atende seus requisitos e descobrir situações em que o software se comporta de maneira incorreta, indesejável ou de forma diferente das especificações.

Conforme Konschanski e Soares (2006), o esforço para o teste de um software deve ser compensatório, ou seja, deve haver um balanceamento entre o tempo e o custo de testes em um software e a quantidade de defeitos encontrados. Os recursos alocados para testes (pessoas, tempo e ferramentas) fazem parte da definição do projeto e não devem ser decididos logo antes ou durante a realização dos testes. Uma base para definição dos recursos pode ser um histórico de experiências anteriores da empresa.

Para Sommerville (2011), o sistema de software deve passar por três estágios de teste:

- Teste de desenvolvimento, em que o sistema é testado durante o desenvolvimento para descobrir bugs e defeitos. Projetistas de sistemas e programadores podem estar envolvidos no processo de teste.
- Teste de release, em que uma equipe de testes independente testa uma versão completa do sistema antes que ele seja liberado para os usuários. O objetivo deste teste é verificar se o sistema atende aos requisitos dos stakeholders.
- Testes de usuário, em que os usuários ou potenciais usuários de um sistema testam-no em seu próprio ambiente.

### **2.5.1 Testes de unidade**

Um dos métodos mais comuns e utilizados no processo de desenvolvimento é o teste de unidade, também conhecidos como teste unitário. Testes de unidades são os mais básicos e costumam consistir em verificar se um componente individual do software foi implementado corretamente. Este componente pode ser um método ou

procedimento, uma classe completa ou, ainda, um pacote de funções ou classes, Conforme Rios e Filho (2013).

Bartiè (2002) afirma que os testes unitários são, essencialmente, orientados à estrutura interna de implementação do componente que estaremos validando. Nesta estratégia de testes, o objetivo é executar o software de forma a exercitar adequadamente toda a estrutura interna de um componente, como os desvios condicionais, os laços de processamento e todos os possíveis caminhos alternativos de execução.

Sommerville (2011) afirma que quando uma pessoa está testando as classes de um objeto, deve projetar os testes para fornecer uma cobertura de todas as características do objeto. Isso significa que se deve: Testar todas as operações associadas ao objeto, definir e verificar o valor de todos os atributos associados ao objeto e colocar o objeto em todos os estados possíveis, o que significa simular todos os eventos que causam mudanças de estado.

Para evitar qualquer tipo de despesa maior com gerenciamento, Rios e Filho (2013) evidenciam que os testes de unidade costumam ser realizados pelo próprio programador e não pela equipe de testes, uma vez que este já conhece a estrutura interna e os desvios condicionais, sendo desnecessário o envolvimento de outras pessoas.

Ao final dos testes unitários, espera-se que os componentes alterados estejam funcionando conforme as especificações técnicas, seguindo as condições internas desenvolvidas conforme os parâmetros de entrada e gerando a saída que se espera do componente.

### **2.5.2 Testes de integração**

Como uma natural sequência lógica dos testes unitários, os testes de integração são executados em uma combinação de componentes para verificar se eles funcionam juntos de maneira correta, ou seja, assegurar que as interfaces funcionem e que os dados estão sendo processados de forma correta, conforme as especificações. Componentes podem ser pedaços de código, módulos, aplicações

distintas, clientes, servidores, entre outros, conforme definição de Rios e Filho (2013).

Os testes de integração são necessários visto que, conforme Bartiè (2002) relata, à medida que as unidades vão sendo construídas ou modificadas, estas devem manter a compatibilidade com outros componentes já existentes. Esses testes são orientados pela própria arquitetura do projeto tecnológico, na qual são colocados diversos componentes para que sejam processados e tenham seu funcionamento correto.

O próprio Bartiè (2002) relata que conforme um componente é alterado, todos os outros componentes que empregam as rotinas e atributos destes componentes são considerados automaticamente incompatíveis, visto que a compatibilidade existente anteriormente foi quebrada pela alteração. Todos os erros causados por estas alterações que causam incompatibilidade e não foram corretamente tratadas pela equipe responsável pelo desenvolvimento da alteração devem ser identificados nesta etapa do processo de qualidade.

Os pontos principais a serem verificados nestes testes, portanto, são:

- Deve-se garantir a perfeita interface entre os diversos componentes existentes na arquitetura tecnológica;
- Deve-se garantir a perfeita colaboração entre componentes;
- Deve-se garantir que determinados requisitos estejam implementados.

Seguindo na definição de Bartiè (2002), para realização destes testes, deve-se exercitar:

- Todas as dependências existentes entre os componentes;
- Todas as interfaces existentes entre os componentes;
- Todos os parâmetros de interfaces de um componente;
- Todas as propriedades públicas de um componente.

Para execução dos testes de integração existem algumas abordagens sugeridas. Dentre elas pode-se evidenciar a abordagem bottom-up onde os componentes de níveis mais básicos da arquitetura do aplicativo são conjuntamente testados de forma a simular as interações dinâmicas e suas interfaces. À medida

que se evolui nos testes de integração, os níveis de arquitetura vão crescendo para que sejam simuladas suas interfaces também. Este processo segue até que não existam mais níveis a serem alcançados, atingindo o nível máximo de integração que os componentes podem chegar.

Ao final dos testes de integração, espera-se que os componentes funcionem de forma integrada, mantendo o correto tratamento de comunicação entre as partes e considerando as diferentes variáveis tecnológicas existentes no software.

### **2.5.3 Testes de sistema**

Como sequência dos testes de integração, existem os testes de sistema, que abrangem uma parte maior do sistema. Os testes de sistemas são testes realizados pela equipe de qualidade visando a execução do sistema como um todo ou um subsistema, dentro de um ambiente operacional controlado, para validar a exatidão e perfeição na execução das funções, conforme Rios e Filho (2013) explicam. Conforme Bartiè (2002) explica, quando atingido este estágio a maior parte dos erros deve ter sido detectada nos testes unitários e nos testes de integração.

Conforme o próprio Bartiè (2002), os testes de sistemas devem contar com uma infraestrutura mais complexa de hardware, para que seja próximo do ambiente real de produção. Os testes de sistema devem concentrar-se, mais nos aspectos funcionais, de performance, segurança, disponibilidade, instalação, recuperação, entre outros, conforme necessidade levantada pelo planejador deste estágio. Entram também determinados aspectos de funcionalidade que não puderam ser testados e, portanto, deverão integrar os testes de sistemas. Para isto, Rios e Filho (2013) afirmam que a operação normal do sistema deve ser testada de forma mais próxima possível do que ocorrerá em ambiente de produção.

Ao final desta fase, espera-se que todas as funcionalidades desenvolvidas ou que possam ter sofrido alteração pelo desenvolvimento realizado tenham sido testadas, verificando-se a correta execução da aplicação quanto às definições de negócio, bem como as características de performance e de ambiente que a aplicação deve ser executada quando estiver em ambiente de produção.

#### **2.5.4 Testes de aceite**

Ao fim do processo de qualidade e aceitação temos os testes de aceite. Os testes de aceite são os testes finais de execução do sistema, realizados pelos usuários finais, visando verificar se a solução atende aos objetivos do negócio e aos seus requisitos, no que diz respeito à funcionalidade e usabilidade, antes da utilização no ambiente de produção, segundo Rios e Filho (2013).

Bartiè (2002) afirma que nessa etapa, o software é disponibilizado para clientes e usuários com o objetivo de estes validarem todas as funcionalidades requisitadas no início do projeto. Os usuários terão a oportunidade de interagir com um sistema completo, testado pela equipe de qualidade anterior.

Conforme o próprio autor Bartiè (2002), se ao chegar a esta etapa do processo de garantia a solução ainda apresentar falhas, é sinal que os processos de detecção de defeitos anteriormente executados não estão sendo efetivos, ou seja, não foram corretamente implementados, indicando falhas de processo. Nesta abordagem, existe o aceite formal, que consiste na execução dos testes pelos próprios usuários e clientes, aproveitando o ambiente de homologação. Ao final ocorre a aprovação ou não dos usuários, fazendo com que o projeto seja confirmado ou retorne para a devida correção.

O tipo de teste mais comum para este caso é teste funcional. Segundo Rios e Filho (2013), os testes funcionais são os que avaliam se o especificado foi implementado, verificando além da operação normal, se os valores extremos que examinam os limites dos sistemas foram considerados.

Portanto, ao se chegar ao final desta etapa o sistema não deve possuir falhas, ou se possuir, devem ser consideradas de baixa importância pelo próprio cliente, já que após isto será realizada a entrada do software em produção. Este é o último processo de qualidade que deve ser implementado, visto que nada pode ser alterado sem a validação do cliente e é uma grande oportunidade para a validação da qualidade do processo como um todo.

## 2.6 CATEGORIA DE TESTES

Para identificação de problemas, muitas são as possíveis abordagens. Conforme Bartiè (2002) relata, os testes têm por objetivo a identificação de erros, porém devemos entender que a localização de todo e qualquer tipo de defeito exige um esforço muito grande. Se não existe o tempo nem recursos suficientes para executar todos os procedimentos de testes, deve-se planejar uma estratégia na qual estabelecemos quais os tipos de erros queremos prioritariamente encontrar. Conforme Pressman (2011) afirma, uma série de testes especializados é necessária para encontrarmos tipos de erros diferentes. Diante disso, se elencará algumas das principais categorias de testes uteis para o processo estudado.

### 2.6.1 Teste de funcionalidade

Os testes de funcionalidade são muito comuns pois se verifica o resultado final da aplicação. Conforme Bartiè (2002), esta categoria de testes tem por objetivo simular todos os cenários de negócio e garantir que todos os requisitos funcionais sejam implementados. Os testes funcionais devem ser direcionados pelos documentos de especificação funcional. A ideia é garantir que não existam diferenças entre os requisitos funcionais e o comportamento do software construído. Os testes devem ser aplicados sem que exista uma preocupação direta com o algoritmo interno do software, empregando valores de entrada e esperando valores de saída como referencial para a avaliação da conformidade do requisito.

### 2.6.2 Teste de desempenho

Para situações em que o desempenho é determinante para se garantir que o sistema ou a funcionalidade está gerando os resultados conforme o esperado, deve-se garantir, antes da entrega do projeto, que o desempenho está adequado. Segundo Pressman (2011), o teste de desempenho é projetado para testar o desempenho em tempo de execução do software dentro do contexto de um sistema.

O teste de desempenho pode ser aplicado em cada uma das etapas de testes, do unitário ao aceite. Mas aconselha-se que seja realizado no teste de sistema visto que para se obter resultados confiáveis é necessário que se qualifique todos os elementos do sistema já integrados e se antecipe as situações, trazendo-o para antes dos testes de aceite.

Conforme Bartiè (2002), para realização dos testes de desempenho, deve-se especificar claramente o cenário que se deseja obter e apontar quais são os tempos de resposta considerados factíveis à realização de cada um.

Este tipo de teste, segundo o próprio Bartiè (2002), pode ser executado da seguinte forma:

- Validar todos os requisitos de desempenho e compará-los com o que está sendo possível obter no sistema;
- Simular vários usuários acessando a mesma informação, simultaneamente;
- Simular vários usuários processando a mesma transação, simultaneamente;
- Simular tráfego de rede em x por cento;

### **2.6.3 Teste de usabilidade**

Uma preocupação crescente com a difusão de diferentes plataformas de utilização de sistemas informatizados é a usabilidade do sistema. Para validação deste ponto, utilizam-se os testes de usabilidade, que, conforme Bartiè (2002), tem por objetivo simular a utilização do software sobre a perspectiva do usuário final.

A ideia deste teste é medir o nível de facilidade disponibilizada pela aplicação, de modo a deixar o software simples e intuitivo. Para avaliação deste tipo de teste, pode-se:

- Entrar em cada tela e avaliar a facilidade de navegação;
- Realizar várias operações e depois desfazê-las;
- Realizar procedimentos críticos e avaliar mensagem de alerta;



- Avaliar número de passos para realizar as principais operações;
- Avaliar a existência de ajuda em todas as telas;

#### **2.6.4 Teste de estresse**

Muitas vezes os erros ou inconsistências não estão nas operações dentro dos padrões normais de dia a dia e sim nas situações anormais. Para testar estas situações existe o teste de estresse ou carga. Segundo Pressman (2011), o teste de estresse usa um sistema que demande recursos em quantidade, frequência ou volumes anormais. Bartiè (2002) explica que esta categoria de testes deve provocar aumentos e reduções sucessivas de transações que superem os volumes máximos previstos para o software, gerando contínuas situações de pico e avaliando como o software e toda a infraestrutura está se comportando.

Segundo o próprio Bartiè (2002), estes testes podem ser executados da seguinte forma:

- Elevando e reduzindo sucessivamente o número de transações simultâneas;
- Aumentando e reduzindo o tráfego de rede;
- Aumentando o número de usuários simultâneos;
- Combinando cada um dos elementos acima.

#### **2.6.5 Teste de segurança**

Com a dependência dos processos organizacionais para com os sistemas informatizados, muitas informações sigilosas podem estar contidas nos sistemas. Conforme Pressman (2011) relata, qualquer sistema de computador que trabalhe com informações sensíveis ou que cause ações que podem inadequadamente prejudicar indivíduos é um alvo para acesso impróprio ou ilegal. As invasões abrangem uma ampla gama de atividades: hackers que tentam invadir sistemas por

diversão, funcionários desgostosos que tentam invadir por vingança, indivíduos desonestos que tentam invadir para obter ganhos pessoais, entre outros motivos.

Diante disso deve ser possível garantir que operações mal intencionadas não sejam possíveis de ocorrer dentro de um sistema informatizado. E para isto existe o teste de segurança. Segundo Bartiè (2002), o teste de segurança tem por objetivo detectar falhas de segurança que podem comprometer o sigilo e a fidelidade das informações, bem como provocar perdas ou interrupções de processamento. A ideia desta categoria de teste é avaliar o nível de segurança que toda a infraestrutura oferece simulando situações que provocam a quebra de protocolos de segurança, expondo quais são os pontos de maior fragilidade e risco de ocorrerem ataques.

Esses testes podem ser executados da seguinte forma:

- Validar todos os requisitos de segurança identificados;
- Tentar acessar funcionalidades e informações que requerem perfil avançado;
- Tentar extrair backups de informações sigilosas;
- Tentar descobrir senhas e quebrar protocolos;
- Tentar processar transações geradas de fontes inexistentes;

#### **2.6.6 Teste de configuração**

Em muitos casos, o software deve operar em uma variedade de plataformas e sob mais de um ambiente de sistema operacional. O teste de configuração, também conhecido como teste de ambiente ou teste de disponibilização, exercita o software em cada ambiente no qual ele deve operar. Além disso, o teste de disponibilização examina todos os procedimentos de instalação e software especializado de instalação que serão usados pelos clientes, conforme define Pressman (2011). A ideia, garante Bartiè (2002), é garantir que a solução tecnológica execute adequadamente sobre os mais variados ambientes de produção previstos nas fases de levantamento dos requisitos. Estes testes são recomendados para serem aplicados em softwares de missão crítica, pois requerem muito esforço para operacionalizá-los.

Para testá-los, pode ser realizado através das seguintes formas:

- Variando sistemas operacionais e suas versões;
- Variando navegadores, para sistemas de arquitetura *web*;
- Variando hardware que interage com a solução;
- Combinando os elementos.

Realizando portanto esta categoria de testes é possível esperar um sistema resiliente, adaptável às mudanças de meio em que podem ser submetidos.

## 2.7 ESTRATÉGIA DE TESTE

Para os testes de validação necessita-se uma estratégia de teste para se validar os requisitos propostos. Conforme Bartiè (2002) existem duas abordagens principais de teste, caixa branca ou caixa preta. A estratégia adotada determina o modo como irá se estabelecer o teste e como serão conduzidos os procedimentos de testes visando minimizar esforços e ampliar as chances de detecção de erros que estão inseridos no software.

### 2.7.1 Teste caixa-branca

Uma das estratégias de teste é o teste de caixa branca. Segundo Molinari (2008), o teste de caixa-branca usa uma estrutura de controle do projeto procedimental para derivar casos de teste. Ou seja, para realização do teste caixa-branca, o responsável deve conhecer a tecnologia empregada no sistema para determinar quais os caminhos que o sistema deve realizar e com isso determinar os parâmetros de entrada e os resultados esperados. Segundo ele, o objetivo macro do teste, quando utilizada esta estratégia, é garantir que todas as linhas de código determinantes e condições foram executadas e estejam corretas.

Segundo Bartiè (2002), os testes de caixa-branca são conhecidos por sua alta eficiência na detecção de erros, porém são também conhecidos por serem custosos

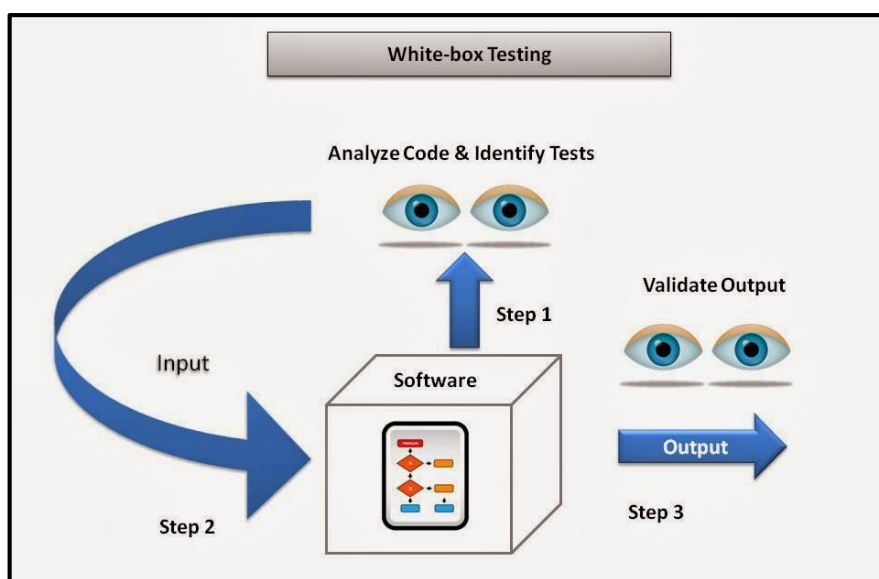
de implementar. Esta estratégia pode ser modelada e estruturada pelos próprios profissionais do desenvolvimento.

Molinari (2008) afirma que para a realização dos testes de caixa-branca, o responsável deve:

- Garantir que todos os caminhos independentes dentro de um módulo sejam exercitados;
- Exercitem as decisões lógicas para valores falsos ou verdadeiros;
- Executem os laços em suas fronteiras, inclusive dentro e fora de seus limites operacionais;
- Exercitem as estruturas de dados internas para garantir a validade.

A ilustração 4, colocada abaixo demonstra graficamente o que se espera dos testes de caixa branca, aonde o analista responsável verifica o código para então definir os testes. O passo 2 (*step 2*) é a reprodução dos parâmetros de entrada definidos na identificação dos testes no programa desenvolvido. Já no passo 3 (*step 3*), se valida a saída e se comparam os resultados obtidos com os resultados esperados.

Ilustração 4 – Testes de caixa-branca



Fonte: <http://www.softwaretestingsoftware.com>, consultado em 02 de junho de 2015

### 2.7.2 Teste caixa preta

A outra estratégia de teste é o teste de caixa-preta. Segundo Molinari (2008), o objetivo macro desta estratégia é garantir que todos os requerimentos ou comportamentos da aplicação ou de um componente estejam corretos. Os métodos de caixa-preta concentram-se nos requisitos funcionais do software.

Segundo Bartiè (2002), não é objetivo do teste de caixa-preta verificar como ocorrem internamente os processamentos no software, mas se o algoritmo inserido no software produz os resultados esperados. A grande vantagem da estratégia de caixa-preta é o fato de esta não requerer conhecimento da tecnologia empregada ou dos complexos conceitos empregados internamente no software. Diante disso os testes de caixa-preta são conhecidos por serem mais simples de se implantar e planejar do que os testes de caixa-branca.

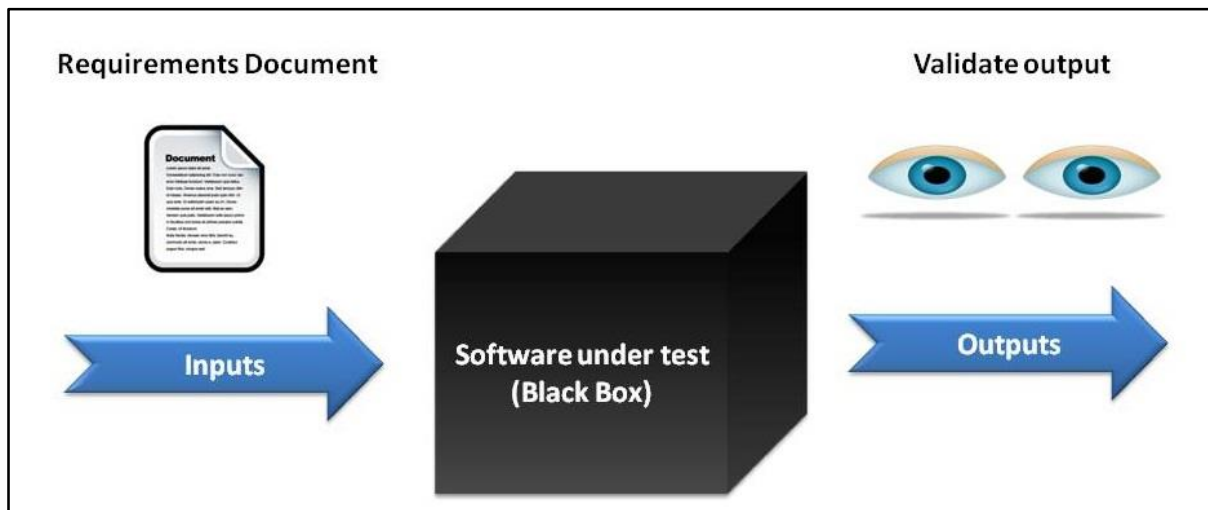
Conforme definição de Molinari (2008), o teste de caixa-preta tende a ser aplicado durante as últimas etapas da atividade de teste. Ele se concentra no domínio da informação. Portanto, os testes são projetados para responder às seguintes perguntas:

- Como a validade funcional é testada?
- Que entradas construirão bons casos de teste?
- O sistema é particularmente sensível a certos valores de entrada?
- Como as fronteiras de uma classe de dados são isoladas?
- Quais índices de dados e volumes de dados o sistema pode e deve tolerar?
- Que efeitos terão combinações específicas de dados sobre a operação do sistema?

A ilustração 5, abaixo colocada, procura demonstrar o que se espera dos testes de caixa preta. Verifica-se inicialmente o que se espera através dos documentos que descrevem os requisitos. A partir disto, coloca-se o software sob teste de funcionalidade, sem avaliar a codificação interna dele para estipulação dos

testes. Validam-se os resultados obtidos com os resultados descritos nos documentos de requisitos, objeto de definição das entradas dos programas.

Ilustração 5 – Testes de caixa-preta



Fonte: <http://www.softwaretestingsoftware.com>, consultado em 02 de junho de 2015

### 2.7.3 Qual estratégia deve ser utilizada?

Afinal, que estratégia de teste deve ser utilizada? A definição de Molinari (2008) é clara quanto a isto.

- Se o enfoque da aplicação é o negócio, então deve ser utilizada caixa-preta;
- Se os testes se referem à interface da aplicação, utiliza-se caixa-preta;
- Se o teste é sobre um componente ou aplicação não visível, utiliza-se caixa-branca;
- Se o ambiente exige que o código e o negócio tenham alta qualidade, utilizam-se ambas;
- Se a equipe de desenvolvimento deseja garantir a qualidade do código a ser enviado para a equipe de homologação, utiliza-se caixa branca.

## 2.8 DOCUMENTAÇÃO DOS TESTES

Todo projeto de software que passa por um processo de qualidade deve ser documentado. A principal norma para documentação de teste de software é a IEEE 829, que é tomada como base para este tópico do trabalho. As formas de documentação propostas serão apresentadas com maiores detalhes a seguir.

### 2.8.1 Casos de teste

Os casos de testes são determinantes para várias partes do processo. Além de direcionar a equipe responsável pela qualidade no que verificar, auxiliam os desenvolvedores e analistas a definir o que o sistema deve fazer.

Conforme Bartiè (2002), o documento de casos de teste registra todo o planejamento dos testes de determinados requisitos que foram estabelecidos durante o desenvolvimento do software. O documento estabelece o que será testado, sendo seu principal objetivo identificar o maior número de cenários e variações de determinado requisito de software.

Conforme o próprio Bartiè (2002), os casos de testes estabelecem quais informações serão empregadas durante os testes desses cenários e quais serão os resultados esperados, estabelecendo a massa crítica de testes necessária para validar os requisitos de software. Os principais componentes que devem estar nos casos de teste são:

- Identificação dos casos de teste (o que testar);
- Detalhamento dos parâmetros de entrada;
- Detalhamento da saída esperada;
- Detalhamento do comportamento esperado;
- Responsáveis pelo teste.

### 2.8.2 Plano de teste

O plano de teste é um artefato que acompanhará todo o processo de testes do projeto. Contudo ele inicia a ser realizado na fase de planejamento do processo de qualidade. O documento deve conter todas as informações necessárias para o correto desenvolvimento dos testes, a estratégia de testes, entradas e saídas esperadas, objetivos, critérios para aceitação, cronograma, envolvidos, entre outros tópicos a serem descritos de forma mais detalhada a seguir.

Após a identificação do plano de teste por sua capa, inicia-se o desenvolvimento do plano. O desenvolvimento possui alguns tópicos que devem ser considerados pelo planejador. Abaixo seguem as informações necessárias bem como o resumo de Rios e Filho (2013) sobre sua aplicação:

- Identificação do plano de testes;
  - Identificador único para reconhecimento do plano de teste. Este identificador é importante para que seja possível determinar que aquele arquivo é um plano de teste do projeto 'X'.
- Escopo;
  - Contém a visão geral do projeto, definindo limites do software que está sendo desenvolvido e tudo que será testado.
- Documentos relacionados;
  - Artefatos usados para elaboração, como autorização do projeto, plano de projeto de desenvolvimento, documento de análise de requisitos, análise de sistemas, entre outros.
- Itens a serem testados
  - Requisitos que serão testados;
- Níveis de testes;
  - Identifica quais níveis da sequência de testes este plano será situado.
- Abordagem do teste
  - Consequência dos níveis, informando as técnicas de testes que deverão ser utilizadas.



- Estratégia de teste
  - Alinhado aos itens anteriores, a estratégia define como os requisitos serão testados.
- Recursos a serem alocados
  - Recursos e seus tipos a serem alocados para execução de cada um dos testes estipulados.
- Cronograma de testes
  - O cronograma definido para a execução de cada um dos tipos de testes.

Com o nível de teste definido na primeira parte do documento, a segunda parte deve conter as funcionalidades a serem testadas. Isto serve para delimitar o que a equipe destinada a realizar este teste deve ou não realizar. Vale ressaltar que como esta parte é dependente do nível do teste as funcionalidades a serem testadas devem estar relacionadas com o nível. Por exemplo, em caso de testes de integração pode ser importante testar a conversão de dados entre banco de dados diferentes. Já em um teste de sistema, a geração de um relatório pode estar aqui destacada. Outro ponto importante é que neste tópico são listadas as funcionalidades que devem ser testadas e não como o teste deve ser realizado.

Além das funcionalidades, os critérios de liberação devem ser descritos, para que estejam claros quais são os critérios para que o projeto seja liberado e também se há algum nível de falha a ser aceito.

Ao final do documento, é recomendado explicitar os documentos a serem entregues ao final do período de qualidade, para posterior revisão caso seja julgado necessário.

## 2.9 MÉTRICAS

Métricas de software são essenciais para avaliação do processo de desenvolvimento. Segundo Molinari (2008) uma das razões-chave para utilizar métricas no desenvolvimento de software é a previsibilidade. O processo de tentar

prever problemas ou simplesmente permitir gerenciar o processo de desenvolvimento de software em todas as fases é fundamental.

Conforme Pressman (2011), medição é o processo pelo qual números ou símbolos são anexados aos atributos de entidades no mundo real para defini-los de acordo com regras claramente estabelecidas. Para Rios e Filho (2013), medição é a primeira etapa que leva ao controle e, eventualmente a melhoria. Se não se mede, não se pode entender o processo. Se não se entende então não se pode controlá-lo. Se não pode controlá-lo, não pode aperfeiçoá-lo e também não pode gerenciá-lo.

Para uma correta medição do processo, o momento de ser realizado conta muito. Conforme cita NITA (2002), as medições devem ser realizadas em todos os projetos ao final de cada iteração de cada fase para que a medida seja a mais aferida possível.

Segundo Sommerville (2011), as métricas podem ser divididas em duas classes.

- Métricas dinâmicas: Métricas coletadas por meio de medições efetuadas de um programa em execução, podendo ser coletadas durante os testes do software ou após o mesmo estar em uso pelos usuários finais;
- Métricas estáticas: Coletadas por meio de medições feitas de representações do sistema, como o projeto, o programa ou a documentação.

Para Pressman (2011), as métricas a serem utilizadas no produto de software devem levar em consideração as fases que a produção do software passa, dentro do ciclo de vida. Para se identificar quais as métricas a serem utilizadas devem-se analisar estas fases e determiná-las, levando em consideração especialmente as fases de análise, construção e testes.

Alguns tipos de indicadores são importantes para o processo de desenvolvimento de software. Os indicadores de cobertura fornecem o quanto, percentualmente, o produto de software foi adequadamente testado. Esta medida está relacionada a quantos requisitos de software foram submetidos a testes ou quanto das linhas de código foram efetiva e devidamente testadas em relação ao total de requisitos do projeto, conforme definição de Bartiè (2002). Os indicadores de

cobertura são aplicáveis a planejamento e execução dos testes. Portanto, quantos dos requisitos se planeja testar e quantos dos requisitos foram testados.

Como são necessários muitos esforços para alcançar o nível de qualidade esperado, torna-se fundamental a existência de indicadores que possibilitem avaliar o volume de erros detectados em cada etapa. Medir a eficiência da verificação é muito importante, pois as atividades de verificação são trabalhosas e introduzem custos e esforços adicionais no processo e por isso os testes de eficiência tem sua importância.

Conforme Bartiè (2002), a missão dos testes de verificação é identificar o maior número de erros, de forma a reduzir ao mínimo, o número de incidências nas fases de validação. Para se quantificar a eficiência dos testes de verificação, deve-se comparar o número de erros levantados pela equipe avaliadora ao número total de requisitos. Diante da subjetividade dos testes de verificação, é aconselhável verificar para determinar a qualidade dos processos de verificação, o número de problemas encontrados nos testes de validação somados ao número de erros em produção, comparando com o total de requisitos do projeto.

Já para determinar o nível de eficiência da validação, precisa-se monitorar o ambiente de produção. Erros em produção indicam que as atividades de validação não foram bem planejadas ou executadas. Diante disso, sugere-se que se compare para determinação deste indicador, o total de erros de validação com o total de erros em produção.

A qualidade do software determina o quanto esse software está próximo dos requisitos e será através dos defeitos encontrados em cada uma das fases que será possível determinar a distância entre o patamar desejado de qualidade e o que está sendo realizado. Será através das análises dos resultados destes indicadores que serão tomadas importantes decisões como a finalização da etapa, a implantação definitiva do produto no cliente ou a negociação por aumento de prazo ou recursos para o projeto, conforme comenta Bartiè (2002).

Um ponto muito importante a ser considerado sobre defeitos é a sua prioridade. Conforme relata Bartiè (2002), cada defeito de software produz impacto diferente no meio em que o cerca. Em sistemas de negócios, muitos erros de software podem levar a organização a perder grandes volumes de negócios apenas

por uma paralisação de poucos minutos. Outros sistemas podem ser interrompidos por períodos mais longos, porém também terão prejuízos significativos caso não exista uma resolução no dia seguinte. Outros sistemas podem permanecer dias e até semanas sem funcionar, porém não provocarão grandes prejuízos. Diante disso percebe-se que cada funcionalidade provoca um impacto diferente nos negócios. Diante disso é válido classificar os defeitos em prioridade. Sugere-se a adoção desta classificação por quatro níveis diferentes:

- Urgentes;
- Alta prioridade;
- Média prioridade;
- Baixa prioridade.

A correta abordagem dos defeitos quanto a sua prioridade pode fornecer um gerenciamento muito mais objetivo e evitar uma série de problemas administrativos mais graves.

## 2.10 CONSIDERAÇÕES

Existem muitos métodos, em diversas abordagens, para o alcance da qualidade de software. Os diferentes métodos podem ser aplicados ao longo do processo e de variadas formas e cada um será importante para se atingir qualidade em determinada parte do processo. A união destes métodos e sua aplicação efetiva ao longo dos projetos são a chave para o alcance de uma qualidade satisfatória no produto final entregue ao cliente, não aumentando de forma muito significativa os custos diretos de cada projeto.

### **3 CENÁRIO ATUAL DO LABORATÓRIO DE CRIAÇÃO E APLICAÇÃO DE SOFTWARE**

O laboratório de criação e aplicação de software é um laboratório de desenvolvimento de software que atua como parte do centro de ciências exatas e da tecnologia da Universidade de Caxias do Sul. O laboratório foi criado em 2015 por iniciativa dos professores do próprio centro com o objetivo de realizar projetos de desenvolvimento e inovação nas áreas de software e sistemas de informação, com foco na aplicação de novas tecnologias.

A idealização de um laboratório de software interno surgiu da existência de muitos alunos com necessidade e anseio de uma primeira oportunidade ou contato com projetos de softwares que não sejam de apenas acadêmicos, como os que são realizados em sala de aula, mas que também pudessem ver os programas sendo aplicados em um ambiente real. Combinado a isto, a existência de muitas demandas de cunho educacional, oriundas de diferentes centros da UCS, motivou a idealização do laboratório.

Outro fato motivador foi a constante necessidade dos próprios acadêmicos de outras áreas de conhecimento, que buscam apoio e consultoria de tecnologia da informação em projetos de trabalho de conclusão de curso e novos negócios.

No mesmo sentido, existe o desejo de capacitação dos alunos dos cursos da área de computação. As demandas internas da Instituição são oportunidades de situações reais que aproximam o acadêmico da realidade do mercado e permitem o desenvolvimento de aprendizagem ativa baseada em problemas. Este tipo de aprendizagem pode ser trabalhado em disciplinas, estágios e trabalhos de conclusão de curso.

Diante deste cenário, os objetivos do laboratório são a geração de produtos de software que sejam aderentes as diferentes áreas e segmentos da universidade, como são os casos de softwares para ensino, melhora na gestão da própria organização ou no atendimento dos alunos por exemplo.

O laboratório procura atender os clientes internos em projetos de desenvolvimento de software em arquitetura web, desktop, aplicativos móveis, jogos,

entre outros tipos, consultoria técnica e apoio a decisões sobre software para acadêmicos e unidades e manutenção dos produtos por ele desenvolvidos.

O laboratório possui um corpo permanente de professores que são responsáveis por diferentes áreas do instituto. Atualmente, destacam-se as seguintes áreas: Gerência de projetos, análise de sistemas, DBA e desenvolvimento, além de uma coordenação central que é de responsabilidade da Prof. Iraci Cristina da Silveira De Carli. A responsabilidade de cada uma das áreas citadas é de diferentes acadêmicos da própria universidade. Além destes, compõem a equipe do laboratório alunos e funcionários.

Uma particularidade do laboratório é o grande volume de mudanças que podem ocorrer na equipe. Isso se deve ao fato dos alunos possuírem ligação temporária com o projeto já que a motivação para este vínculo vem da obrigação de carga horária complementar nos cursos que realizam, busca por conhecimento e experiência inicial em suas carreiras profissionais, entre outros motivos. Outro ponto a ser observado é a limitada carga horária que os professores podem oferecer ao projeto visto que possuem outras atribuições principais na instituição.

Os stakeholders externos que afetam, aprovam e influenciam o projeto, mas não estão subordinados à gerência deste projeto são:

- Pró-reitoria de inovação e desenvolvimento tecnológico.
- Pró-reitoria acadêmica.
- Alunos de outras áreas demandantes de consultoria e apoio técnico.
- Gerência de Tecnologia da Informação.
- Unidades da UCS que poderão se tornar clientes.
- Clientes dos projetos
- Alunos de computação do CCET
- Professores de computação do CCET
- Direção do CCET.
- Empresa Junior da UCS.

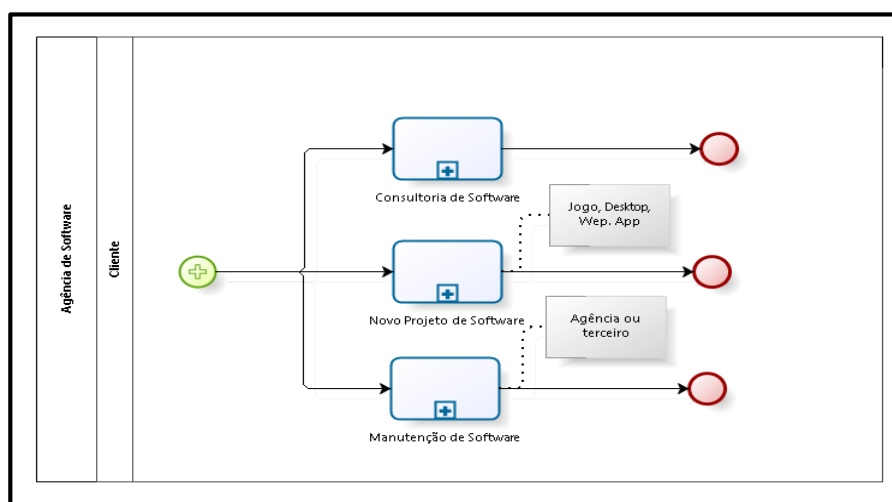
- NET Inovação.
- TecnoUCS.

Os custos de cada projeto de desenvolvimento são de responsabilidade da unidade demandante, os quais devem ser planejados pelo laboratório e aprovados nas instâncias pertinentes. O laboratório procura entregar projetos com qualidade igual ou superior às empresas comerciais que prestam serviços semelhantes.

### 3.1 PROCESSO ATUAL

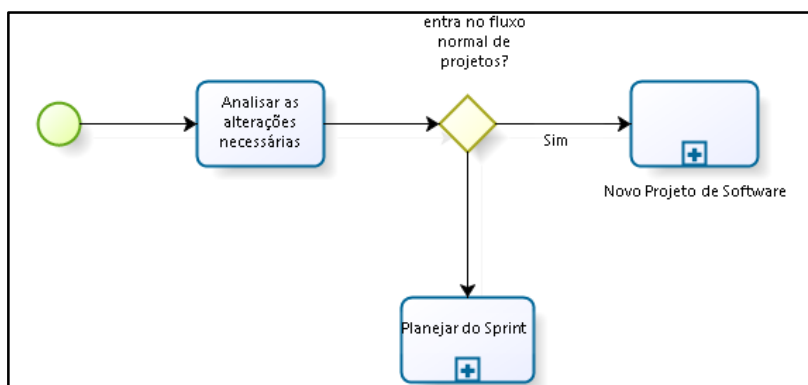
O processo atual inicia-se com a identificação de uma melhoria ou uma necessidade por parte do cliente. Esta necessidade pode ser um novo projeto de software, uma manutenção de software ou uma consultoria. Como o foco deste trabalho acadêmico é a implantação de um processo de garantia de qualidade de software, o foco na descrição do processo atual será no desenvolvimento e manutenção de software. A ilustração 6 demonstra as três opções de processo existentes no laboratório. A ilustração 7 demonstra como as possíveis alternativas quando uma manutenção é identificada no produto.

Ilustração 6 – Macroprocesso atual



Fonte: Equipe do laboratório.

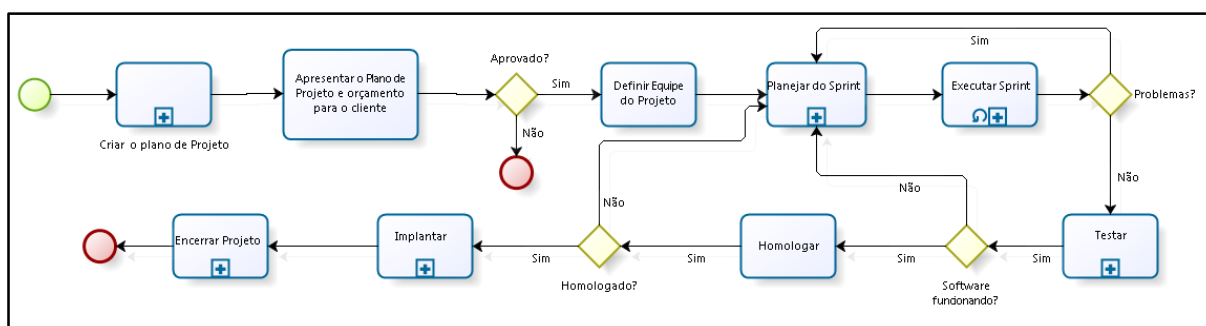
Ilustração 7 – Manutenção de software



Fonte: Equipe do laboratório.

A ilustração 8, abaixo colocada, demonstra o processo para melhor entendimento da descrição a seguir, que irá detalhar não só as tarefas contidas na ilustração, mas também seus subprocessos.

Ilustração 8 –Processo de desenvolvimento de novos projetos

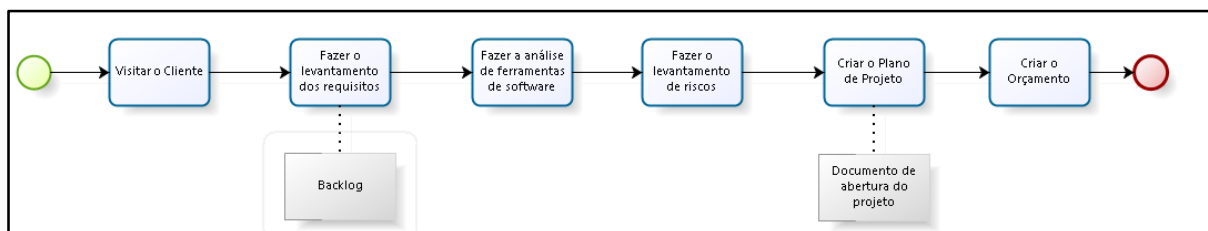


Fonte: Equipe do laboratório.

No processo de desenvolvimento de um novo projeto de software, a partir da identificação da necessidade, realiza-se uma visita de um analista da equipe do laboratório ao cliente da ferramenta. Realiza-se o levantamento de requisitos da ferramenta desejada, criando um documento de análise a partir desta conversa. A partir da análise de requisitos realiza-se a análise de sistemas e o levantamento de riscos do projeto. Cria-se um documento de plano de projeto que engloba as análises citadas. A partir deste documento e das análises cria-se então o orçamento do projeto. Estas tarefas fazem parte do subprocesso de criação de plano de projeto, que está demonstrado na ilustração 9 abaixo.



Ilustração 9 – Subprocesso de criação de plano de projeto

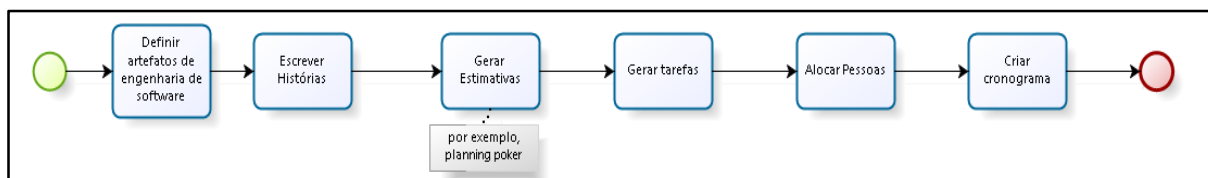


Fonte: Equipe do laboratório.

Diante da criação do plano de projeto e do orçamento, realiza-se uma nova reunião entre os interessados, laboratório e cliente, para apresentação do projeto. Caso o projeto não seja aprovado, pode ocorrer o término do projeto ou o ciclo pode recomeçar de sua fase inicial.

Quando ocorre a aprovação do projeto, os responsáveis pelo laboratório definem a equipe do projeto. O projeto então parte para uma nova fase de planejamento. Dentro desta nova fase de planejamento, definem-se os artefatos de engenharia de software que ainda não foram documentados na análise de sistemas citada no início do processo. Escrevem-se histórias e são definidas as tarefas do projeto. Geram-se as estimativas de tempo por história. Diante das estimativas, a equipe então é definitivamente alocada e o cronograma é definido. Estas tarefas juntas formam o subprocesso de planejamento de sprints, demonstrado na ilustração 10.

Ilustração 10 – Subprocesso de planejamento de sprints

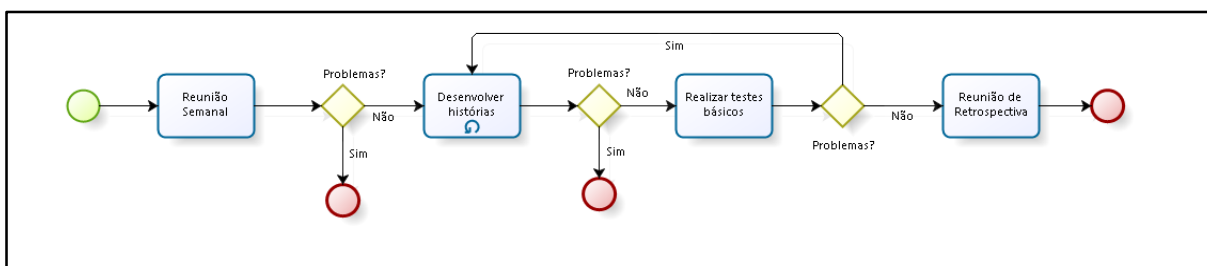


Fonte: Equipe do laboratório.

Inicia-se então o desenvolvimento do software. Os projetos são conduzidos com reuniões semanais, onde são relatados os problemas e soluções são propostas. A cada nova tarefa realizada são realizados testes básicos no desenvolvimento. Caso sejam identificados problemas o processo retorna ao

desenvolvimento. Isto ocorre ciclicamente até que os testes não identifiquem mais inconsistências. Ao final do subprocesso está prevista uma reunião de retrospectiva, com o objetivo de relatar as lições aprendidas do desenvolvimento do produto. As tarefas citadas ao longo deste parágrafo fazem parte do subprocesso de execução de sprints, exibido na ilustração 11.

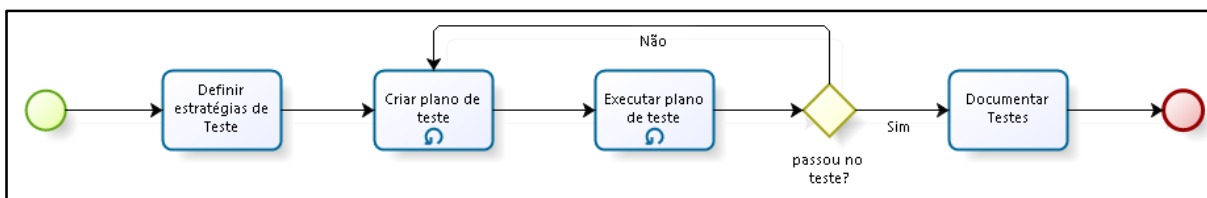
Ilustração 11 – Subprocesso de execução de sprints



Fonte: Equipe do laboratório.

Ao término do desenvolvimento, inicia-se a fase de testes. A fase de testes inicia-se com a definição da estratégia de testes. Diante disso cria-se o plano de testes, documenta-se e executa-o. Caso na execução do plano de testes algum problema seja identificado, este é encaminhado ao fluxo de planejamento do desenvolvimento, que deve analisar e desenvolver a correção. O processo de testes é executado pela própria equipe de desenvolvimento. O processo de testes é graficamente exibido neste trabalho através do subprocesso de testes, na ilustração 12.

Ilustração 12 – Subprocesso de testes

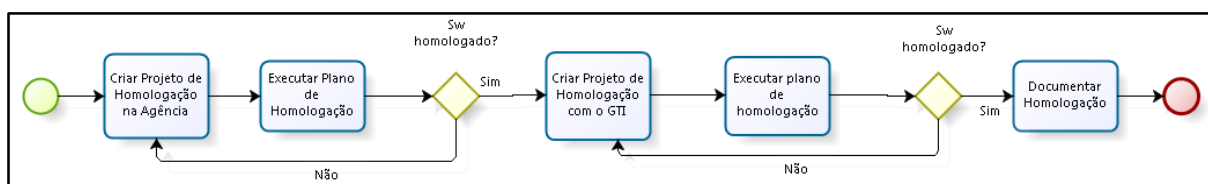


Fonte: Equipe do laboratório.

Após a identificação e correção destes problemas, o projeto é encaminhado para a fase de homologação. Esta fase de homologação é executada pelo cliente do

projeto. Novamente, caso algum problema seja identificado, o mesmo é encaminhado ao fluxo de planejamento do desenvolvimento, que deve analisar e desenvolver a correção. Este processo ocorre até a completa homologação da ferramenta por parte do cliente. Estipula-se que deve haver a definição de um plano de homologação, que será por este trabalho na definição do novo processo. Se analisado o macroprocesso, este conjunto de tarefas está disposto no subprocesso de homologação, que está realçado na ilustração 13.

Ilustração 13 – Subprocesso de homologação

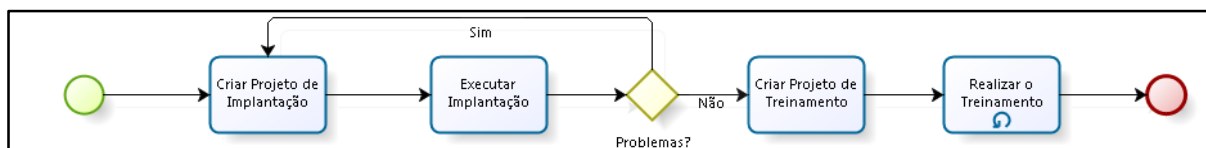


Fonte: Equipe do laboratório.

Ao término da homologação pelo cliente do software, o laboratório compreende que é necessário haver uma homologação por parte do GTI, equipe interna do centro de tecnologia da UCS. Para isto, define que deve haver também um plano para homologação. Diante disto, este trabalho deve abranger mais esta definição também.

Diante do término da homologação, inicia-se a fase de implantação com a criação de um plano de implantação. Este plano é executado. Ao término desta implantação, inicia-se o planejamento do treinamento com a criação de plano de treinamento. Após a criação, executam-se os treinamentos. Os treinamentos são dados pelos analistas ao grupo de usuários do cliente que estarão em contato com o software produzido pelo laboratório. Esta fase compõe o subprocesso de implantação, ilustrado na figura abaixo.

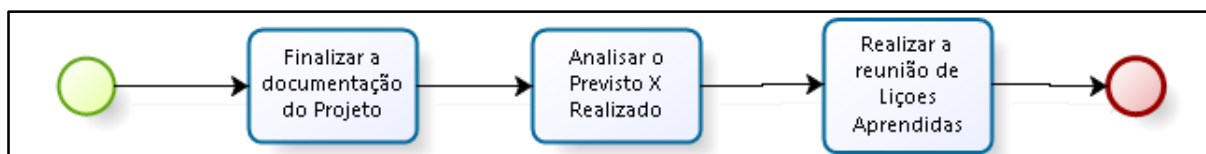
Ilustração 14 – Subprocesso de implantação



Fonte: Equipe do laboratório.

Diante da correta e bem sucedida implantação do software, finaliza-se a documentação, compara-se o previsto e o realizado, discutindo-se as lições aprendidas internamente pelo laboratório de software. Esta fase de encerramento é realizada pelos envolvidos no projeto por parte do laboratório de criação e aplicação de software e está demonstrada na ilustração 15.

Ilustração 15 – Subprocesso de encerramento de projeto



Fonte: Equipe do laboratório.

O processo de manutenção inicia-se com a identificação de algum problema por parte do cliente. Este relata ao laboratório seu problema. O laboratório por sua vez analisa a situação, verificando as alterações necessárias. Diante disso, dependendo da extensão e complexidade das alterações necessárias, define-se se um novo projeto será necessário ou se as alterações podem ser encaminhadas diretamente para a equipe de análise e desenvolvimento. Se um projeto novo é necessário este seguirá o padrão de projetos de desenvolvimento do laboratório de criação e aplicação de software. Caso não seja necessário, as tarefas e histórias são descritas e o processo continua desta fase.

### 3.2 PROBLEMA

A equipe do laboratório de criação e aplicação de software relata que o processo de qualidade deve ser repensado visto que está hoje focado apenas nos testes de software, fazendo com que o tempo disponível para as tarefas diárias seja destinado a correção de problemas, o que trata-se de um retrabalho. Percebe-se que, se comparado o processo colocado neste capítulo com os processos de qualidade de software citados no capítulo anterior, não existem tarefas de verificação no processo atual. Além disso, os processos de qualidade hoje definidos estão vagamente descritos, não demonstram quais as alternativas existentes para cada uma destas tarefas e quais os critérios para a escolha entre as opções. O processo atual não prevê documentação gerada a partir dos processos de qualidade e, portanto, também não prevê indicadores extraídos após cada atividade.

### 3.3 CONSIDERAÇÕES

O processo atual demonstra-se bem definido para o desenvolvimento em si, porém oferece poucas tarefas com o intuito de garantir a qualidade de software, o que é prejudicial principalmente em projetos de maior escopo ou complexidade. Obviamente para adição de processos de verificação e validação o processo que será proposto terá que se realocar algumas tarefas para que as documentações necessárias em cada atividade estejam disponíveis no momento em que forem executadas.

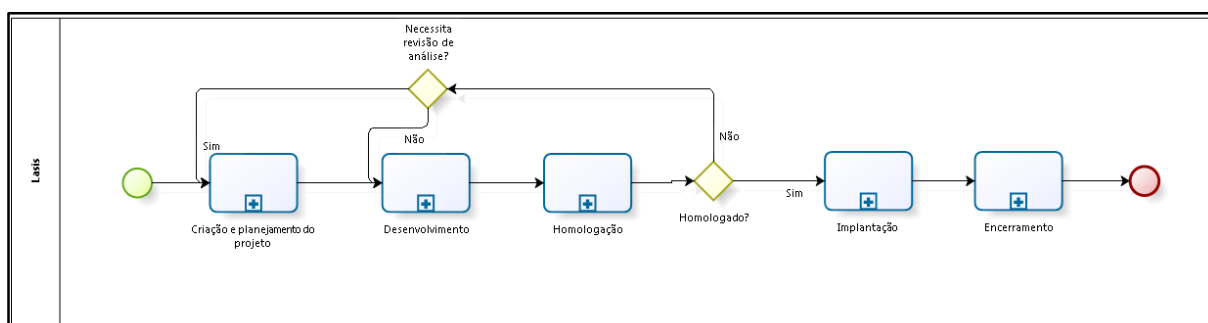
#### 4 PROPOSTA DE NOVO PROCESSO

Diante da necessidade de um plano de qualidade e da expectativa relatada pelos profissionais do laboratório, foi desenvolvida uma proposta de novo processo para ser implantada e validada em um projeto protótipo. Este capítulo realizará a explicação do processo como um todo. Para cada nova tarefa de qualidade sugerida, será detalhado através de um subcapítulo como ela será desempenhada. Algumas mudanças no próprio processo que não são diretamente relacionadas com qualidade foram necessárias para que a documentação, as validações e verificações necessárias ocorram no período adequado ao processo de qualidade proposto.

O processo proposto visa incluir tarefas de verificação e validação no processo dos projetos que façam com que o processo fique mais confiável, sem torná-lo oneroso, trazendo um bom custo-benefício e trazendo uma qualidade satisfatória ao produto final. Não serão considerados neste trabalho os processos de auditoria do processo, ficando em aberto para outro trabalho futuro. A decisão de não considerar a auditoria de processo neste trabalho foi feita pois para a criação de um novo processo de auditoria, associada à criação de processos de verificação e validação que será realizada, poderia tornar o trabalho inviável na questão de prazo.

O macroprocesso proposto para o laboratório é formado por cinco principais processos, sendo eles: Criação e planejamento do projeto, desenvolvimento, homologação, implantação e encerramento. A ilustração 16 demonstra este macroprocesso e a ordem em que os processos estão propostos.

Ilustração 16 – Macroprocesso proposto

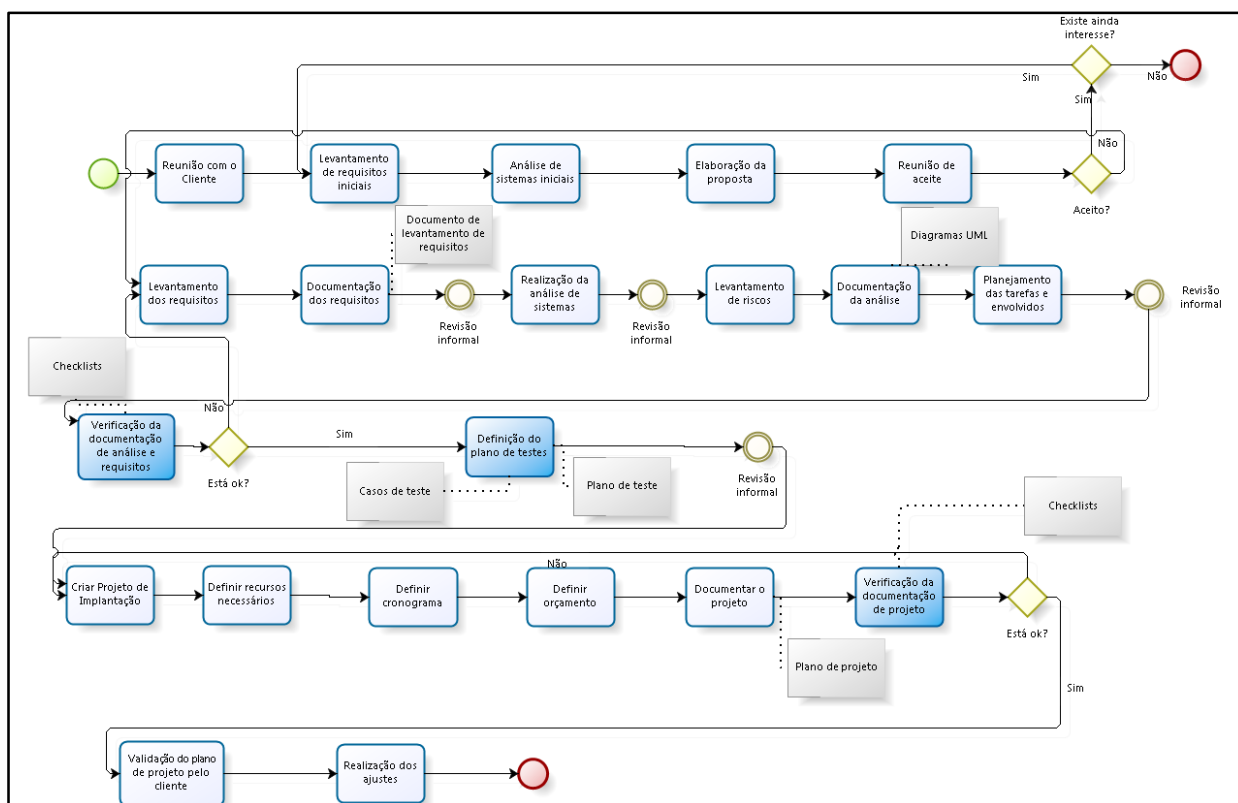


Fonte: Equipe do laboratório.

## 4.1 PROCESSO DE CRIAÇÃO E PLANEJAMENTO DO PROJETO PROPOSTO

Com base macroprocesso descrito, a sequência se inicia com o processo de criação e planejamento do projeto. A ilustração 17 traz o BPM deste processo proposto e ilustra o que será descrito ao longo desta sessão. As tarefas destacadas em azul escuro são as tarefas que estão sendo criadas por este trabalho.

Ilustração 17 - Processo de criação e planejamento do projeto proposto



Fonte: Próprio autor.

### 4.1.1 Reunião com o cliente e análise inicial

O novo processo inicia-se com a identificação de uma nova necessidade, melhoria ou manutenção de algum produto já existente. Após a manifestação ser

realizada pelo cliente da solução, se identifica se a alteração necessita de um novo projeto.

Na reunião realiza-se um levantamento de requisitos inicial, quando são identificadas as premissas e expectativas do projeto. Estes requisitos então são documentados nos documentos de análise, que seguem os padrões já estabelecidos. Com os requisitos documentados, realiza-se então uma análise de sistemas inicial. O objetivo destas duas primeiras etapas iniciais citadas é entender o que o cliente deseja e então realizar um desenho prévio da aplicação com um orçamento e resultante proposta, a fim de buscar a aprovação do cliente quanto à solução proposta e ao orçamento necessário. Os requisitos e análise de sistemas não são realizados de forma detalhada e completa neste momento visto que como ainda deve ser aprovado o projeto, recursos poderiam ser desperdiçados em caso de não aprovação. Por consequência nenhuma nova tarefa de qualidade está sendo proposta para esta fase do planejamento. O processo se repete até que o cliente aprove a proposta ou encerre o projeto, conforme sua decisão.

#### **4.1.2 Do levantamento de requisitos ao planejamento das tarefas e envolvidos**

Aprovada a proposta inicia-se o levantamento de requisitos detalhado, onde o objetivo é detalhar de forma completa tudo o que foi levantado no processo inicial. Estes requisitos então são documentados no documento de requisitos. É importante salientar que os documentos já existentes no processo de desenvolvimento atual não foram alterados visto que não faziam parte do foco do trabalho.

Com os requisitos documentados, realiza-se então a análise de sistemas. A análise de sistemas é realizada hoje por diagramas UML pelo analista de sistemas. Após a documentação da análise de sistemas realiza-se o levantamento de riscos, atualizando a documentação.

É necessário salientar que ao decorrer das tarefas de documentação dos requisitos, realização da análise de sistemas e documentação da análise foram propostas revisões informais que serão realizadas caso o executor acredite ser



necessário. As revisões informais são revisões realizadas por pessoas selecionadas pelo próprio executor da tarefa a fim de, com uma opinião de uma terceira pessoa, determinar se a tarefa está sendo cumprida de forma satisfatória ou se existem pontos a serem melhorados, alterados ou adicionados.

Inicia-se então o processo de planejamento das tarefas e envolvidos. Neste momento, o responsável pela gerência do projeto, que para o caso do laboratório também é o responsável pela qualidade do projeto, deve determinar além das tarefas de desenvolvimento quais serão os tipos de testes necessários para este projeto, tendo em vista o número de horas a serem destinadas para a completa execução do projeto. Este ponto de decisão é necessário visto que uma das premissas deste trabalho era a preocupação da elevação de horas nos projetos, o que pode diminuir o custo-benefício em projetos de pequeno porte, já que o número de horas necessárias para toda a elaboração do projeto é determinante no seu custo.

Quando os documentos citados são finalizados, realiza-se então o primeiro processo formal de qualidade, a revisão da documentação de análise e requisitos, a ser descrita em detalhes no próximo item.

#### **4.1.3 Verificação da documentação de análise e requisitos**

Conforme mencionado anteriormente, durante o levantamento e a definição da solução tecnológica, ações de verificação deverão ser realizadas para garantir a qualidade das atividades e documentos produzidos com a realização de revisões. Devido à importância do levantamento de requisitos e da análise de sistemas, estipulou-se uma tarefa exclusiva para sua verificação. Decidiu-se unir em uma única tarefa as duas revisões, de requisitos e de análise, pois pôde ser previsto que desta forma o tempo investido seja menor do que em duas tarefas separadas, atendendo a premissa de não elevar o número de horas de forma muito significativa.

Por definição do responsável pelo planejamento da qualidade do projeto, um grupo de pessoas, diferentes da pessoa responsável pelo levantamento de requisitos e da análise de sistemas, deve realizar uma reunião de revisão. O grupo

de pessoas pode ser composto por alunos, professores ou ambos, dependendo da natureza e das características do projeto.

Conforme comentado nos capítulos anteriores, revisão é um processo humano subjetivo de análise de determinados documentos. Portanto, a revisão de requisitos será realizada através de reunião de revisão. Reuniões de revisão foram discutidas no capítulo 3 e são compostas por pessoas com habilidades heterogêneas, permitindo que as análises e discussões sejam aplicadas sobre várias perspectivas diferentes. As reuniões devem ser planejadas, com os participantes já tendo recebido e previamente lido o material necessário para a reunião. É de responsabilidade do gerente de projetos garantir que a documentação necessária seja entregue aos participantes em tempo hábil para preparação para a reunião.

O dimensionamento das reuniões, a ser realizado também pelo gerente de projetos, será em função da complexidade, importância e impacto que o projeto possui, conforme Bartiè (2002) comenta em citação já apresentada anteriormente, que já reflete nesta fase do projeto no número de horas destinado a ele e o consequente orçamento destinado.

As características a serem avaliadas nos requisitos nesta tarefa seguirão a definição de Bartiè (2002), e, portanto, devem estar:

- Completos: Todos os requisitos do projeto devem estar documentados;
- Claros: Cada requisito deve expressar seu propósito em relação ao projeto;
- Simples: Cada requisito deve ser exato, não dar margens a dúvidas;
- Consistentes: Cada requisito não deve possuir conflitos com outros requisitos existentes.
- Relevantes: Cada requisito deve pertencer ao escopo do projeto em questão;
- Testáveis: Cada requisito deverá fornecer informações que possibilitem quantificar se um determinado item foi adequadamente implementado;
- Factíveis: Cada requisito deve ser viável em sua implementação, avaliando as condições financeiras, humanas e tecnológicas disponíveis no projeto.

A verificação de análise deverá verificar se os diagramas UML produzidos pela equipe de análise estão corretos conforme as normas de padrão destes documentos e se estão suportando todos os requisitos propostos pelo documento de levantamento de requisitos. Deverá ser verificado se a arquitetura contempla a aplicação a ser desenvolvida e se considera características como flexibilidade, escalabilidade e ser reutilizável. Além disso, os nomes de todos os objetos propostos devem estar corretos e claros assim como suas próprias descrições.

Ao decorrer da reunião, os participantes deverão debater cada um dos requisitos, verificando se os mesmos estão descritos conforme a definição estabelecida acima, o que é válido também para a arquitetura descrita através da análise de sistemas.

Caso todos os itens estejam de acordo o processo pode prosseguir, caso contrário o autor deve revisar os pontos levantados pelos revisores. Após esta edição, poderá ser realizada uma nova reunião de verificação. Vale salientar que nesta reunião os avaliadores devem verificar se os itens foram atendidos ou não, fazendo com que a reunião não se estenda muito e evitando a criação de conflitos entre os integrantes. Como os itens serão atendidos é de responsabilidade do autor do documento. A assertividade no levantamento de requisitos, constatada pelo número total de falhas no processo de verificação, poderá ser utilizada como métrica para avaliar a qualidade do levantamento de requisitos. Em caso de ocorrência de falhas, o gerente de projetos deve criar uma tarefa para que o executor do artefato em questão realize as alterações necessárias. Se, para padronizar as ações, for constatado que a melhor forma de formalizar esta tarefa for através de um documento de não conformidade, sugere-se a utilização do mesmo. O layout proposto para este tipo de tarefa será discutido no item 4.3.1.

A verificação do cumprimento dos itens citados será realizada por checklists. O checklist de verificação possuirá as seguintes informações:

- No cabeçalho:
  - Título;
  - Nome do projeto;
  - Fase do projeto;
  - Data da avaliação;

- Avaliadores.
- Nas linhas:
  - Cada uma das características de qualidade esperadas para o projeto;
  - Verificação do cumprimento ou não (Sim/Não).

Esta ação está sendo proposta para ser realizada dentro do subprocesso de criação e planejamento do projeto, após as fases de levantamento de requisitos e documentação da análise vista a importância destes dois processos. Como quase todas as tarefas subsequentes no processo de desenvolvimento são decorrentes destas, a qualidade destes artefatos é imprescindível.

#### **4.1.4 Definição do plano de teste**

O plano de testes é um documento que acompanhará todo o processo de testes do projeto. Por isso este deve ser realizado na fase de planejamento do processo de qualidade. No caso do laboratório de criação e aplicação de software, no processo de criação e planejamento do projeto. O documento deve conter todas as informações necessárias para o correto desenvolvimento dos testes, dentre eles a estratégia de testes, critérios para aceitação, cronograma, envolvidos, entre outros tópicos a serem discutidos. É de responsabilidade do analista de sistemas desenvolver este artefato visto que possui o entendimento de todos os requisitos relatados pelo cliente e de todas as funcionalidades esperadas do sistema.

A estrutura do artefato possui como informações de cabeçalho:

- Identificação principal, com título;
- Informações de escopo, contendo a visão geral do projeto;
- Definição dos limites do software;
- Se existirem, devem ser listados os documentos relacionados, como artefatos utilizados, documento de levantamento de requisitos, diagramas de análise de sistemas, entre outros.

A partir das informações gerais serem corretamente descritas, inicia-se o detalhamento do plano de teste. Este detalhamento buscará descrever as informações pertinentes a cada tipo de teste a ser empregado para o projeto em planejamento. Portanto o primeiro item a ser descrito é o tipo de teste (teste de integração, teste de sistema, etc.). Todos os tipos de testes devem ser listados nesta seção do artefato e para cada um dos tipos de testes serão descritos os seguintes itens:

- Categoria de testes
  - Categoria de testes a ser utilizada para este tipo de teste com as devidas justificativas cabíveis. Para determinar as categorias a serem testadas, o analista deve verificar levar em consideração as características do projeto, o orçamento disponível para o projeto e pelo nível de serviço esperado. Pode consultar o responsável pelo gerenciamento do projeto para discutir as questões de orçamento disponíveis.
- Itens a serem testados
  - Detalhamento de todos os itens a serem testados para a categoria relatada com as devidas justificativas cabíveis. Os itens são determinados dos parâmetros desejáveis de qualidade, que por sua vez, são extraídos dos requisitos funcionais e não funcionais do projeto.
- Abordagem do teste
  - Serão descritas as técnicas de testes que deverão ser utilizadas para a correta validação com as devidas justificativas cabíveis. As técnicas são determinadas a partir do tipo de teste a ser realizado e a categoria de teste em questão.
- Estratégia de teste
  - A estratégia define como os requisitos serão testados, ex: caixa-branca ou caixa-preta, com as devidas justificativas cabíveis. A estratégia de teste deverá ser estipulada pelo analista de sistemas no plano de teste levando em consideração o tipo de teste que se está fazendo e a categoria de teste.

- Itens a não serem testados
  - Itens que por definição do analista não deve ser testados com as devidas justificativas cabíveis.
- Recursos a serem alocados
  - Recursos a serem alocados para execução dos testes descritos.

O plano de testes possuirá na sua estrutura mais detalhada os casos de teste, a serem descritos no próximo item.

#### **4.1.5 Definição dos casos de teste**

Os casos de testes são essenciais para o processo de desenvolvimento como um todo e não somente para o processo de qualidade, já que além de servir como base para os testes, auxiliam também a equipe de desenvolvimento no entendimento da funcionalidade em desenvolvimento. O artefato de casos de testes proposto deve possuir as seguintes informações:

- Identificação dos casos de teste;
- Detalhamento dos parâmetros de entrada;
- Detalhamento da saída esperada;
- Detalhamento do comportamento esperado;
- Responsáveis pelo teste.

A responsabilidade do desenvolvimento dos casos de teste é do analista de sistemas responsável pela análise, visto que é ele que compila as informações reunidas pela análise de negócio e determina as ações que o software deve ter. O responsável pelo teste é determinado em conjunto com o responsável pela gerência do projeto.

Esta tarefa está proposta para ocorrer no processo de criação e planejamento do projeto, após a revisão da documentação de análise e requisitos visto que para definição dos casos de testes, o levantamento de requisitos e a análise de sistemas

são necessários e devem estar verificados para a correta elaboração. Os casos de teste deverão compor o plano de teste por ser parte integrante do artefato.

#### **4.1.6 Criação do projeto de implantação até documentação do projeto**

Na continuidade do processo de criação e planejamento do projeto proposto, após a conclusão da revisão do plano de testes, inicia-se a revisão informal dos planos de testes, que segue o mesmo padrão de operação das revisões informais já citadas ao longo do texto.

Cria-se o plano de projeto. Definem-se os recursos necessários, elencando os recursos humanos e de infraestrutura. Define-se cronograma, elencando os marcos, tarefas necessárias e seus responsáveis, o que vale também para os processos de qualidade propostos neste trabalho. A partir das definições citadas, define-se orçamento final e documenta-se o projeto com todos estes detalhes.

#### **4.1.7 Verificação da documentação de projeto**

Como uma nova atividade de revisão formal, a tarefa de revisão da documentação do projeto está sendo proposta no processo de criação e planejamento do projeto, após a documentação do projeto e antes da validação do projeto pelo cliente. Isto porque só após a documentação dos recursos, cronograma e riscos, existem os artefatos suficientes para este processo de qualidade. A tarefa deve ser verificada antes da reunião com o cliente para que o projeto apresentado seja o mais adequado possível, não negligenciando nenhum item que possa ter sido esquecido ou mal documentado.

A tarefa de revisão da documentação será uma tarefa de verificação, que procurará identificar se:

- Os recursos foram estipulados de forma correta;
- Os riscos existentes foram documentados e antecipados;
- Se as tarefas estão corretamente descritas;

- Se a comunicação entre as partes está prevista;
- Se os custos do projeto estão corretamente descritos e estão coesos com o que se espera praticar;
- Se os marcos do projeto foram estipulados e estão corretos;
- Se os custos estão dentro do orçamento;
- Se o prazo estabelecido pelo cliente será atendido;
- Se o cronograma está correto, levando-se em consideração o prazo acordado.

Esta tarefa de verificação será realizada por reunião de revisão, por uma equipe diferente da responsável pela elaboração dos artefatos e os resultados serão apurados por checklist. A forma de formalização destas tarefas será a mesma estipulada no item 4.1.3, já que o padrão para realização é o mesmo, bem como os indicadores gerados.

#### **4.1.8 Validação do projeto com o cliente e realização de ajustes**

Como continuação do processo, realiza-se uma reunião de validação com o cliente, que poderá apontar ajustes em cada um dos artefatos documentados até o momento. Diante disso, a próxima tarefa é a realização dos ajustes solicitados. A partir da conclusão desta tarefa, entende-se que o processo de criação e planejamento do projeto está concluído.

Após a conclusão do processo de criação e planejamento do projeto proposto, inicia-se o processo de desenvolvimento, conforme pode ser verificado no macroprocesso, na ilustração 17. Este processo segue as definições e diretrizes definidas ao longo subprocesso de criação e planejamento do projeto.

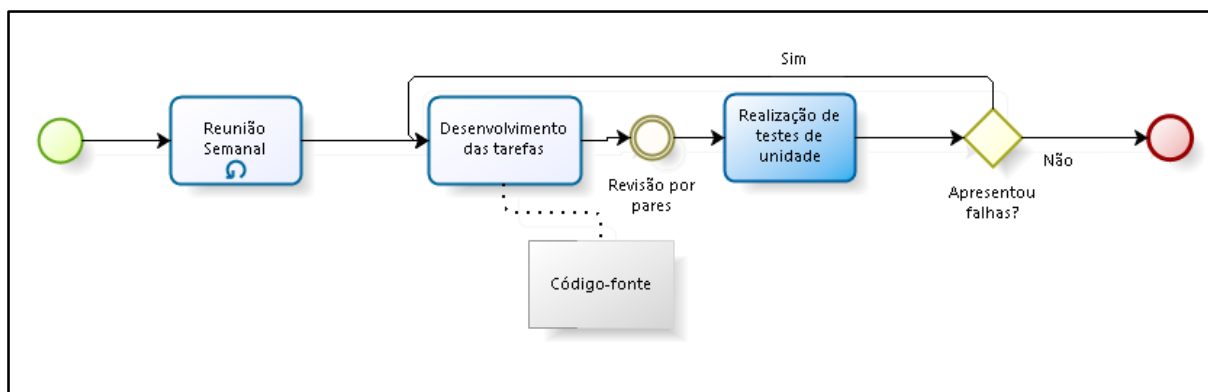
## **4.2 PROCESSO DE DESENVOLVIMENTO**

O processo de desenvolvimento é responsável pela produção de código-fonte para atingimento dos parâmetros e objetivos estabelecidos anteriormente ao longo



do planejamento. A ilustração 18 demonstra a sequência de atividades propostas para esta fase do projeto. A tarefa “Realização de testes de unidade” está destacada em azul mais escuro para indicar que esta foi uma tarefa criada por este trabalho.

Ilustração 18 - Processo de desenvolvimento proposto



Fonte: Próprio autor.

A equipe de desenvolvimento em si, realiza reuniões semanais a fim de discutir as dificuldades dos projetos em execução e os novos projetos a serem realizados na semana. Após esta reunião, inicia-se o desenvolvimento de cada um dos pontos especificados nos documentos de análise, realizados no processo de criação e planejamento do projeto. Conforme mencionado anteriormente, o desenvolvimento seguirá as definições realizadas no processo anterior. A responsabilidade de garantir que o desenvolvimento esteja fiel ao que está sendo proposto nos artefatos produzidos anteriormente é das tarefas de qualidades que estão sendo propostas e serão apresentadas com maiores detalhes na sequência.

Ao longo do desenvolvimento, pode-se solicitar uma revisão por pares, que se trata de uma revisão informal para código-fonte. Neste caso, o desenvolvedor solicita o auxílio de outra pessoa para revisão do código e auxílio para determinar se o comportamento final da aplicação está condizente com o especificado nos artefatos. Após o término do desenvolvimento, realizam-se os testes de unidade, tarefa de qualidade proposta por este trabalho e detalhada no item a seguir.

### 4.2.1 Testes de unidade

Propõe-se que a tarefa de realizar os testes de unidade seja realizada no processo de desenvolvimento. Isto pois a análise de sistemas e o plano de testes já existem e já foram verificados, o que faz com que se determine com maior facilidade e agilidade os resultados a serem esperados pelo executor para os casos em que o objeto será condicionado.

Normalmente a estratégia de testes utilizada nesta fase é a de caixa-branca. Isto se deve ao fato de que os testes unitários são executados sobre o código recém desenvolvido e, portanto, orientados a partir da estrutura interna de implementação do componente.

Propõe-se que a tarefa seja de responsabilidade do próprio desenvolvedor responsável pelo desenvolvimento ou manutenção de determinado objeto já que o momento adequado para fazer isto é ao final de cada alteração, pois o desenvolvedor estará com o código-fonte do objeto em seu poder. Os testes unitários tem como objetivo principal executar o software de forma a exercitar adequadamente toda a estrutura interna de um componente, como os desvios condicionais, os laços de processamento e todos os possíveis caminhos alternativos de execução, conforme a estratégia de caixa-branca determina.

Para exercitar corretamente a estrutura interna dos componentes utiliza-se ferramentas automatizadas. As ferramentas automatizadas não serão sugeridas por este trabalho pois este tem o intuito de tratar da parte gerencial e de processo de um plano de garantia da qualidade de software. De qualquer forma, a automatização de testes unitários por ferramentas está ligada à técnica que é utilizada. Normalmente o desenvolvedor optará por uma técnica voltada para os critérios estruturais, como grafo de fluxo de controle ou grafo de programa, que analisam os desvios condicionais do código e propõem os parâmetros de entrada de acordo com o grafo gerado.

Caso alguma inconsistência seja encontrada, o desenvolvedor tem condições de já corrigir o problema para que o processo de desenvolvimento entregue uma versão estável do objeto, não sendo necessário criar uma ocorrência para tal. Corrigido o problema, o desenvolvedor deve reiniciar os testes de unidade, a fim de

garantir que as funcionalidades já testadas não tenham sido influenciadas pela alteração realizada.

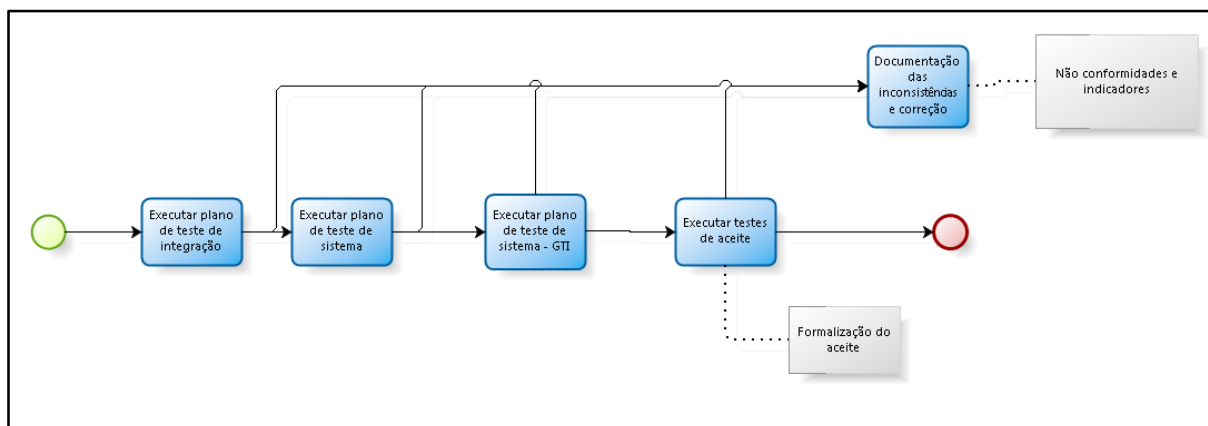
Devido às características citadas, portanto, não serão criados artefatos neste plano de qualidade para esta tarefa, bastando os casos de testes devem estar criados na ferramenta de automatização de testes a ser estudada futuramente por outro trabalho, conforme mencionado anteriormente.

Caso os testes unitários apresentem falhas, os objetos que causam a falha devem ser corrigidos pela própria equipe de desenvolvimento. Após os testes de unidade não apresentarem mais erros e todos os pontos necessários terem sido desenvolvidos, o processo de desenvolvimento é encerrado.

#### 4.3 PROCESSO DE HOMOLOGAÇÃO

Com o término do processo de desenvolvimento, inicia-se o processo de homologação. Esta fase do projeto é responsável por aplicar as regras e determinações de qualidade especificadas pelo processo de criação e planejamento do projeto, além das técnicas necessárias para que a aplicação seja entregue estável e de acordo com o que foi estabelecido com o cliente sobre o produto de software produzido no processo de desenvolvimento, garantindo que o cliente está recebendo o que manifestou no início do projeto. O processo de homologação possui algumas tarefas de qualidade propostas que serão descritas nos próximos itens e demonstrada na ilustração 19.

Ilustração 19 – Processo de homologação proposto



Fonte: Próprio autor.

Para cada um dos planos de testes, caso seja encontrada alguma inconsistência, é realizada a tarefa de documentação das inconsistências e correção. Vale salientar que se a execução de determinado plano de testes encontrar qualquer inconsistência e passar pelo processo de correção, após esta correção, o processo de homologação deve ser reiniciado, a fim de garantir que a alteração não interferiu nas demais funcionalidades do sistema, além de garantir que de fato corrigiu o problema.

#### 4.3.1 Executar os testes de integração

Os testes de integração são executados em uma combinação de componentes para verificar se eles funcionam juntos de maneira correta, ou seja, assegurar que as interfaces funcionem e que os dados estão sendo processados de forma correta, conforme as especificações.

Os pontos principais a serem verificados nestes testes, portanto, são:

- Deve-se garantir a perfeita interface entre os diversos componentes existentes na arquitetura tecnológica;
- Deve-se garantir a perfeita colaboração entre componentes;
- Deve-se garantir que determinados requisitos estejam implementados.

Devido a sua natureza técnica, os testes de integração devem ser realizados utilizando a estratégia de caixa-branca. Recomenda-se que os testes de integração sejam realizados pelo analista de sistemas, que pode delegar esta função a um desenvolvedor.

Os testes de integração devem seguir o plano de testes estabelecido, utilizando uma das seguintes abordagens: bottom-up ou top-down. A escolha da abordagem deve partir da arquitetura já existente da aplicação a ser testada, bem como a categoria de testes que está sendo verificada e ambas devem previamente documentadas no caso de teste, na composição do plano de teste. A categoria de testes é importante componente de decisão pois testes de desempenho tendem a utilizar a abordagem bottom-up, enquanto testes de usabilidade tendem a utilizar a abordagem top-down.

A abordagem bottom-up que avalia os componentes diretamente relacionados aos componentes alterados e, conforme os testes evoluem, sobe-se no nível da arquitetura do sistema até que todos os níveis de arquitetura de software tenham sido avaliados. Já na abordagem top-down, o que ocorre é o contrário. Primeiro é verificada a camada superior da aplicação e então os componentes que se relacionam com o item superior vão sendo validados até que os demais níveis de arquitetura passem por validação.

Devido a sua natureza, estes testes são relativamente complexos de serem estipulados e executados, visto que buscam identificar todos os objetos que interagem com os objetos alterados ou desenvolvidos e, então, aplica-se o teste colocando o objeto nos estados possíveis. Diante disso, cabe ao elaborador do plano de teste verificar a necessidade desta categoria.

Toda não conformidade encontrada em qualquer tipo de teste deverá ser documentada no documento de não conformidades. Este documento deverá possuir os seguintes itens:

- No cabeçalho:
  - Identificação própria;
  - Identificação do projeto a que se refere.
- Nas linhas deverão ser descritas as inconsistências encontradas, relatando as seguintes informações:

- Fase do teste em que foi identificada a inconsistência;
- Categoria em que foi identificada a inconsistência;
- Descrição do problema;
- Data da identificação;
- Prioridade para resolução;
- Responsável pela identificação do problema;
- Identificação de referência à ferramenta de controle da resolução (Ex: Bug 987321);
- *Status* atual da resolução da situação (Ex: Pendente, em desenvolvimento, corrigido).

Na descrição do problema devem estar evidenciados os objetos que ocasionaram a não conformidade e de que modo deve ser submetido o teste para que o problema seja simulado.

Após a documentação das não conformidades deverá ser feito um planejamento pelo responsável pela qualidade do projeto, neste caso o gerente de projetos, para que sejam corrigidos os problemas. Este planejamento deve levar em consideração também uma nova validação da integração.

Ao final, a documentação de não conformidades deve ser utilizada para quantificar o número de não conformidades comparado ao número total de cenários possíveis, criando-se assim as métricas do projeto quanto a defeitos. Aproveitando os dados informados nos documentos de não conformidades, alguns indicadores importantes poderão ser gerados. Alguns exemplos propostos são:

- O número de não conformidades por fase de testes;
- O número de não conformidades por categoria de testes.

#### **4.3.2 Executar os testes de sistema – Equipe de testes**

Os testes de sistemas propostos serão testes realizados pela equipe de qualidade visando a execução do sistema ou um subsistema como um todo, dentro de um ambiente operacional controlado, para validar a exatidão e perfeição na execução das funções. O teste de sistema compõe uma das atividades de validação

propostas neste trabalho. Diante disso estão relacionados para serem realizados no processo de homologação, após os testes de integração para que seja garantido que as funções básicas dos itens desenvolvidos foram testadas e as integrações dos mesmos com os seus objetos dependentes também.

Esta fase do processo de homologação deve ser realizada por uma equipe diferente da que participou do desenvolvimento do sistema, focada apenas na qualidade, podendo ser composta por alunos, professores ou ambos, conforme planejamento realizado pelo gerente de projetos. Sugere-se que seja uma equipe diferente porque, conforme foi evidenciado anteriormente, as pessoas responsáveis pelo desenvolvimento, quando realizam processos de qualidade, tendem a testar o que desenvolveram e não todas as características do sistema. Já se uma equipe diferente executa os testes, seu raciocínio não está direcionado e, portanto, existe uma maior probabilidade de encontrar falhas.

Os testes devem ser realizados em um ambiente de homologação com características semelhantes ao ambiente de produção, conforme disponibilidade. Em sua totalidade, deve ser executado utilizando a estratégia de caixa-preta, ou seja, validando o resultado conforme os requisitos estipulados sem considerar para isto diretamente o código fonte no momento de definir os casos de teste. Deve seguir as determinações do plano de teste, podendo utilizar para esta fase, as seguintes categorias de testes (conforme definição presente no próprio plano de testes ):

- Teste de funcionalidade;
  - Categoria responsável pela validação do resultado final da aplicação, comparando-o com o descrito no levantamento de requisitos;
- Teste de desempenho;
  - Validação do desempenho da aplicação comparado aos tempos estipulados no levantamento de requisitos;
- Teste de estresse
  - Categoria de teste responsável por levar o sistema a situações de carga anormais.
- Teste de usabilidade

- Categoria responsável por medir o nível de facilidade disponibilizada pela aplicação, de modo a deixar o software simples e intuitivo;

Toda não conformidade encontrada em qualquer tipo de teste deverá ser documentada no documento de não conformidades, cujo modelo já foi descrito nesta seção quando abordado o tema “testes de integração”.

Após a documentação das não conformidades deverá ser feito um planejamento pelo responsável pela qualidade do projeto, neste caso o gerente de projetos, para que sejam corrigidos os problemas. Este planejamento deve levar em consideração uma nova validação do software também.

Ao final, a documentação de não conformidades deve ser utilizada para quantificar o número de não conformidades comparado ao número de requisitos totais estipulados, criando-se assim as métricas do projeto quanto a defeitos. Os indicadores sugeridos para esta fase são:

- Indicadores por fase de testes;
- Indicadores por categoria de testes;
- Número total de erros nos testes de sistema com o número total de erros nos testes de integração.

Destaca-se aqui este último indicador, não mencionado ainda anteriormente nas tarefas propostas, que compara os testes entre as fases. Este tipo de indicador é útil na medição da acuracidade de cada fase.

#### **4.3.3 Executar os testes de sistema - GTI**

Se definida a necessidade deste processo de qualidade, são executados os testes de sistema por parte do GTI. A necessidade de execução é definida no plano de testes com base na criticidade do projeto, na natureza do projeto e no número de horas existentes para atividades de validação.

De características iguais aos testes de sistema estipulados para a equipe de testes, os testes de sistema no GTI (Governança de Tecnologia da Informação)



seguem as mesmas diretrizes já citadas para aquele processo com a diferença de que devem ser realizados pela equipe do GTI. A diferença é a especialidade da equipe, composta por indivíduos de natureza mais técnica que desempenham funções, entre outras, de manutenção dos ambientes em que os softwares serão instalados.

Devido à especialidade desta equipe, algumas categorias de teste podem ser desenvolvidas com maior precisão, como são os casos dos testes de:

- Desempenho;
  - Avaliação do desempenho da aplicação e comparação do mesmo com os parâmetros estipulados nos documentos de requisito.
- Estresse;
  - Submissão da aplicação a situações anormais de carga, tanto a maior quanto a menor, quanto à quantidade, frequência e volume de dados e informações processadas.
- Segurança;
  - Validação responsável por verificar se operações mal intencionadas são possíveis de ocorrer dentro do projeto realizado;
- Configuração
  - Categoria de teste que exercita o software em cada ambiente no qual ele deve operar.

Conforme descrito anteriormente, os testes de sistema executados pelo GTI possuem características de execução iguais aos executados pela equipe de testes e, portanto, seguem as mesmas diretrizes. Diante disso, a documentação das não conformidades, bem como o seu tratamento e métricas, seguirão as mesmas diretrizes dos testes realizados pela equipe de testes.

#### **4.3.4 Executar os testes de aceite**

Ao fim do processo de qualidade e aceitação propõe-se a existência dos testes de aceite. Os testes de aceite são os testes finais de execução do sistema,

realizados pelos usuários finais, visando verificar se a solução atende aos objetivos do negócio e aos seus requisitos.

A tarefa proposta de testes de aceite está localizada no processo de homologação, visto que é o último teste antes da entrada do sistema em produção. A característica principal deste teste é a validação das características funcionais do sistema. Por ser um teste funcional, fica claro que a estratégia a ser utilizada é de caixa-preta. Ele deve ser executado sobre todo o sistema pelos usuários finais do sistema, aproveitando o ambiente de homologação já preparado pela equipe do laboratório de criação e aplicação de software.

Os usuários finais deverão validar se as funcionalidades descritas aos analistas no começo do projeto foram atendidas de forma satisfatória. Entende-se como atendida a funcionalidade que estiver de acordo com o descrito nos documentos de análise de requisitos e que estejam também de acordo com a expectativa do usuário final. Não há como julgar apenas o que está descrito na documentação visto que a revisão da própria qualidade da análise deve estar sendo avaliada.

Ao término da tarefa, caso os testes não encontrem nenhuma falha, o cliente deve redigir um aceite formal, como forma de aprovação do projeto e liberando o sistema para que seja implantado em ambiente de produção. Este aceite é importante para que o laboratório tenha a garantia de que a funcionalidade foi homologada pelo cliente e que este aceita a implantação do projeto.

As não conformidades encontradas devem ser reportadas para a equipe do laboratório, que preencherá o documento de não conformidades conforme descrito anteriormente nesta seção. Com estas informações pode-se assim se verificar o seguinte indicador adicional não mencionado ainda ao longo do texto:

- Número de não conformidades encontradas pelo cliente *versus* o número de não conformidades encontradas no processo de homologação.

Através deste último indicador poderá ser verificada a qualidade no processo de homologação, visto que os testes de aceite são a tarefa final deste processo.

Após a documentação das não conformidades deverá ser feito um planejamento pelo responsável pela qualidade do projeto, neste caso o gerente de

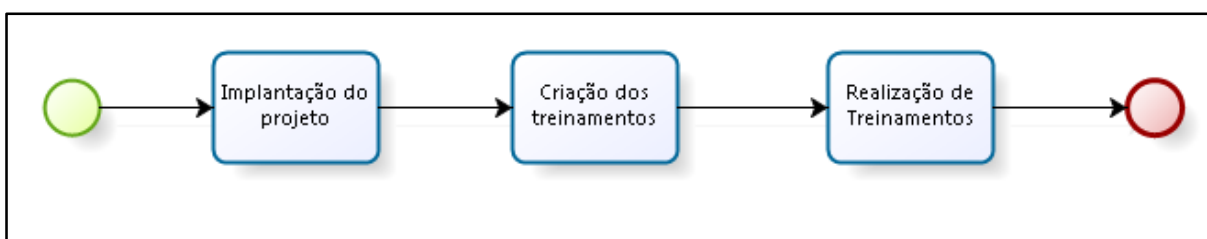
projetos, para que sejam corrigidos os problemas. Este planejamento deve levar em consideração uma nova validação do software também.

Para confirmação do correto teste de aceite, o cliente deve documentar as evidências de testes, através de texto, imagens ou vídeo, para garantir que a aplicação foi testada conforme o acordado. Um revisor da equipe de desenvolvimento deve verificar se os testes de fato ocorreram e de forma satisfatória. Aconselha-se que seja o analista de sistemas responsável pela análise visto que saberá os requisitos que deveriam ser testados e os resultados esperados. Ao final dessas tarefas, finaliza-se também o processo de homologação.

#### 4.4 PROCESSOS DE IMPLANTAÇÃO E ENCERRAMENTO

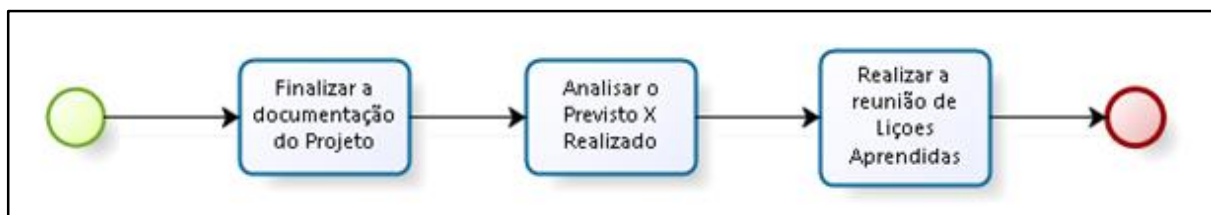
Terminado o processo de homologação, iniciam-se os processos de implantação e encerramento, já descritos na seção que descrevia o processo atual, recolocados aqui nas duas próximas ilustrações e que não tiveram nenhuma alteração na ordem das tarefas e no modo de executá-las, além de não terem sido adicionadas tarefas de qualidade, o que pode ser realizado em um trabalho futuro.

Ilustração 20 - Processo de implantação proposto



Fonte: Próprio autor.

Ilustração 21 –Processo de encerramento proposto



Fonte: Próprio autor.

#### 4.5 CONSIDERAÇÕES FINAIS

O plano de garantia proposto buscou contemplar as necessidades do laboratório e trazer informações suficientes para a escolha das ferramentas.

Para esta escolha, foram consideradas apenas as tarefas de qualidade de software propostas por este trabalho para serem cobertas por uma ferramenta, sendo elas:

- Verificação da documentação de análise e requisitos;
- Definição do plano de teste;
- Definição dos casos de teste;
- Revisão da documentação de projeto;
- Gerenciamento de testes de integração;
- Gerenciamento de testes de sistema;
- Gerenciamento de testes de sistema - GTI;
- Gerenciamento de testes de aceite.

As tarefas de qualidade propostas e não consideradas na abrangência das ferramentas, foram:

- Revisões informais;
- Testes de unidade.

A decisão de não deixar as revisões informais cobertas pelas ferramentas foi motivada pelo fato que este tipo de tarefa é de cunho informal, não havendo necessidade de documentação da atividade realizada nem ganhos substanciais com

a introdução de um processo de descrição da ajuda realizada. Já no caso dos testes de unidade, procura-se manter o dinamismo e a agilidade do desenvolvimento, não havendo a necessidade de documentação ou extração de métricas para este caso. Sabe-se que por muitas vezes esta tarefa é realizada com o auxílio de uma ferramenta para automação, no entanto como este trabalho foca na parte de definição e gerenciamento do plano de qualidade, a execução automatizada ou não desta tarefa não será tratada, ficando em aberto para algum trabalho futuro.

Diante da descrição das tarefas a serem cobertas pelas ferramentas de gerenciamento realizada neste item fica mais fácil, portanto, estabelecer uma ligação entre os requisitos levantados nos próximos itens com o processo detalhado ao longo do capítulo.

## 5 ESCOLHA DA FERRAMENTA E APLICAÇÃO EM UM PROJETO PROTÓTIPO

Processo indispensável para este trabalho, a escolha da ferramenta iniciou-se a partir da identificação das tarefas que necessitavam de ferramentas para a sua melhor execução e gerenciabilidade. Visto que o presente trabalho refere-se a uma proposta de plano de garantia da qualidade de software, os softwares escolhidos devem estar relacionados com o gerenciamento do processo, das tarefas, dos recursos e gerar informações passíveis de serem elencadas em métricas e indicadores.

A identificação ocorreu através da análise das tarefas elencadas no capítulo 4 e as características de cada tarefa demonstradas no próprio capítulo bem como no capítulo 2. Cada tarefa elencada possuía diferentes requisitos já que buscam objetivos diferentes e possuem passos e técnicas diferentes na sua. Outro ponto importante é que além de atender os requisitos propostos as ferramentas escolhidas deveriam manter a fluidez do processo, evitando a burocratização dispensável.

O método de escolha das ferramentas baseou-se em pesquisas em artigos e trabalhos correlatos. Os softwares deveriam estar de acordo com as práticas e preferências da Universidade de Caxias do sul. Estas incluem a utilização de ferramentas *open source* ou software livre, preferencialmente em plataforma web e extensíveis para uma utilização na nuvem computacional.

### 5.1 ESCOLHA DAS FERRAMENTAS

Para escolha das ferramentas buscou-se entender e descrever quais seriam os requisitos que as elas deveriam atender para que a implantação fosse possível. Com isso definido, buscou-se determinar quais ferramentas poderiam fazer com que as necessidades levantadas fossem atingidas de forma satisfatória. Nos próximos itens, serão descritos ambos os cenários com maiores detalhes.

### **5.1.1 Metodologia de escolha**

A metodologia de escolha utilizada seguiu a norma ISO 14598-1, que conforme JUNIOR, PRADELA e OLIVEIRA (2009), apresenta a visão do processo de avaliação. Aplica-se tanto à avaliação de componentes como do sistema, e a qualquer fase do ciclo de vida do produto. A norma separa a escolha em quatro fases, para a correta seleção.

Na primeira fase busca-se estabelecer os requisitos da avaliação, onde são considerados principalmente o objetivo da escolha e o tipo de produto avaliado. Ambas estas tarefas já foram atingidas, com a descrição realizada anteriormente neste trabalho.

Na segunda fase, especifica-se a avaliação. Nesta especificação, devem ser elencadas as métricas e os critérios de avaliação da ferramenta. Esta especificação será tratada pelo item 5.2.2 deste trabalho, onde serão elencados os requisitos que a ferramenta deve apresentar e determinar a obrigatoriedade dos mesmos.

A terceira fase da avaliação é composta pelo projeto da avaliação. Neste projeto deve ser determinada a forma que a avaliação será realizada, descrevendo, portanto, o método de escolha da ferramenta.

Por fim, a quarta fase da avaliação é composta pela aplicação da avaliação e determinação da ferramenta escolhida.

Nas próximas sessões deste trabalho serão tratadas as segunda, terceira e quarta fases citadas com o objetivo de escolha da ferramenta.

### **5.1.2 Requisitos das ferramentas**

Através da descrição das atividades e características de qualidade realizadas no capítulo 2, bem como a descrição da execução das tarefas e proposta de um novo processo no capítulo 4, uma série de requisitos funcionais e não funcionais pôde ser elencada a fim de determinar quais as ferramentas seriam escolhidas para

implantação no laboratório de criação e aplicação de software. Este processo era de suma importância visto que, através desta implantação e a posterior realização de um protótipo em um projeto em andamento no laboratório, poder-se-ia demonstrar que o processo proposto é viável e como ferramentas de mercado podem auxiliar no processo de gerenciamento e execução. As ferramentas de mercado sendo aderentes ajudam a corroborar o plano de garantia proposto, visto que ferramentas de mercado procuram seguir os processos científicos e processos de diversas naturezas e diversas organizações.

Os requisitos funcionais, que por definição de Rezende (2005) são as funções ou as atividades visando o pleno atendimento das necessidades do cliente ou do usuário, são essenciais em uma análise de software para que as ferramentas analisadas e escolhidas sejam aderentes às necessidades dos usuários que irão utilizá-las. Os requisitos não funcionais, que por definição de Wazlawick (2010) são requisitos não ligados diretamente com as funções que um sistema deve possuir mas sim os requisitos de ordem tecnológica da ferramenta como requisitos de performance, arquitetura, interface, entre outros, são importantes para que a escolha das ferramentas sejam realizadas conforme as políticas da organização quanto à arquitetura de software, por exemplo, além de evidenciar alguns parâmetros tecnológicos esperados do software.

Na tabela 1 são descritos os requisitos funcionais e não funcionais levantados a partir dos passos anteriores do presente trabalho que influenciarão na escolha e atenderão a segunda fase da avaliação citada na sessão 5.2.1. A obrigatoriedade do requisito na ferramenta será determinada com três possíveis classificações: obrigatório, recomendado e desejável.

Tabela 1: Requisitos das ferramentas a serem analisadas.

(Continua)

<b>Requisito</b>	<b>Descrição</b>	<b>Tipo de requisito</b>	<b>Classificação</b>
Permitir a criação requisitos	A ferramenta deve permitir a criação de requisitos de software para descrição do que	Funcional	Recomendado



(continuação)

	se espera do projeto em testes.		
Permitir a criação de plano de testes	Permitir a criação de um plano de testes, capaz de gerir o processo de validação.	Funcional	Obrigatório
Permitir a criação de casos de testes	A ferramenta deve proporcionar a criação de casos de testes para detalhar os passos de execução da validação do projeto.	Funcional	Obrigatório
Permitir a atribuição de responsáveis	Permitir que os casos de testes e tarefas de verificação sejam atribuídos a diferentes responsáveis, segregando a sua execução por pessoas.	Funcional	Recomendado
Permitir classificação do caso de teste quanto à verificação ou validação	Possibilidade de classificar entre verificação e validação um caso de teste.	Funcional	Desejável
Permitir categorização de testes	Categorização do teste a ser realizado para o caso de teste em questão.	Funcional	Recomendado
Permitir classificação por abordagem do teste	Abordagem do teste a ser utilizado para aquele caso de teste.	Funcional	Recomendado
Permitir informar a estratégia de teste	Permitir informar a estratégia de teste a ser verificada no momento da execução do caso de teste.	Funcional	Recomendado
Permitir a definição de	Permitir a definição do	Funcional	Desejável

(continuação)

cronograma	cronograma para o projeto de teste em questão.		
Permitir informar parâmetros de entrada	Permitir informar parâmetros de entrada a serem utilizados em um caso de teste.	Funcional	Obrigatório
Permitir informar parâmetros de saída	Permitir informar parâmetros de saída a serem utilizados em um caso de teste.	Funcional	Obrigatório
Permitir informar o comportamento esperado	Permitir informar o comportamento esperado em um caso de teste.	Funcional	Obrigatório
Permitir definir o estado de execução do caso de teste	Controle do estado atual do teste quanto a sua execução.	Funcional	Recomendado
Permitir informar e consultar o resultado do teste	Permitir informar e consultar o resultado da execução do caso de teste.	Funcional	Obrigatório
Permitir informar o tempo para execução do teste	Permitir informar o tempo utilizado para execução de um teste de validação ou para a verificação de um artefato.	Funcional	Desejável
Permitir arquivos em anexo	Permissão para usuários realizarem upload e download de arquivos via anexo para contemplar os documentos relacionados a um plano de teste, por exemplo,	Funcional	Desejável

(continuação)

	ou demonstrar o documento a ser verificado em um teste de verificação.		
Controle de não conformidades	Possuir um controle de não conformidades ocasionadas pelos testes de verificação e validação.	Funcional	Obrigatório
Possuir campo para descrição da não conformidade	Possuir campo para descrição da não conformidade encontrada no caso de teste.	Funcional	Obrigatório
Possibilidade de priorização para resolução	Possibilidade de priorização para resolução da não conformidade.	Funcional	Desejável
Permitir definir responsável pela não conformidade	Responsável por solucionar a não conformidade.	Funcional	Recomendado
Possuir controle de estado atual da não conformidade	Possuir controle de estado atual da não conformidade	Funcional	Desejável
Permitir atribuições diferentes por usuário	Cada usuário deve possuir atribuições de responsabilidades diferentes visto que testadores e gerentes de projetos, por exemplo, possuem responsabilidades diferentes em um projeto.	Funcional	Desejável
Possuir relatório para verificação do	Relatórios gerenciais para verificação do	Funcional	Obrigatório

(continuação)

andamento do projeto	andamento do projeto de testes.		
Permitir relacionamento entre caso de testes e requisito	Permitir relacionamento entre casos de testes de sistema e requisitos para que seja possível saber a cobertura e taxa de inconformidades por requisito.	Funcional	Desejável
Possuir funcionalidade para métricas	Possuir funcionalidade para métricas e indicadores da qualidade do projeto, conforme descritos ao longo do trabalho.	Funcional	Obrigatório
Arquitetura WEB	A arquitetura da ferramenta deve ser WEB, permitindo o acesso de qualquer computador ou dispositivo com acesso à rede computacional da universidade conforme definição da equipe interna da UCS.	Não funcional	Obrigatório
Multiusuário	O software deve permitir a criação de mais de um usuário e o acesso simultâneo concomitante de mais de um usuário.	Não funcional	Obrigatório
Integração entre ferramentas	Caso exista mais de uma ferramenta, deve existir integração entre elas.	Não funcional	Recomendado
<i>Open source</i> ou	As ferramentas devem	Não funcional	Obrigatório

(conclusão)

software livre	seguir as políticas de <i>open source</i> ou software livre.		
----------------	--	--	--

Ao todo, foram elencados 29 requisitos, sendo 25 funcionais e 4 não funcionais. Se analisada a classificação dos mesmos quanto a sua obrigatoriedade, totalizou-se 13 requisitos obrigatórios, 8 recomendados e 8 desejáveis.

### 5.1.3 Projeto da avaliação

No projeto da avaliação, terceira fase desta pesquisa de ferramentas, deve-se definir como a avaliação propriamente dita ocorrerá. Nesta definição, deve estar descrito como se planeja determinar as ferramentas que devem ser avaliadas, diante de que método será verificada a existência das características necessárias e como a obrigatoriedade dos requisitos será levada em consideração.

As ferramentas a serem avaliadas serão obtidas através de pesquisa em trabalhos correlatos e artigos acadêmicos. Através destes serão elencadas as ferramentas a serem avaliadas e que seguirão no processo de avaliação.

Para verificação da existência e do atingimento dos requisitos propostos anteriormente serão utilizados trabalhos correlatos, artigos acadêmicos e as documentações próprias das ferramentas. Desta forma deve-se determinar como a ferramenta atende ao requisito proposto. Para isto, será utilizada uma escala de quatro níveis, conforme abaixo.

- Atende de forma completa: O software possui a funcionalidade de forma a atingir o requisito de forma completa;
- Atende parcialmente: O software possui funcionalidade similar que pode ser adaptada ou possui a funcionalidade desejada porém esta não atende de forma completa o que se espera;
- Não atende: A ferramenta não atinge o requisito proposto através de suas funcionalidades e ou de funcionalidades similares adaptadas.

- Não se aplica: Em caso da ferramenta não necessitar atender o requisito por este ser tratado por outra ferramenta complementar participante do processo de escolha.

Determinou-se que, para avaliar se a ferramenta em análise, os requisitos obrigatórios devem estar atendidos de forma completa ou parcial. Preferencialmente, todos os requisitos recomendados devem estar tratados de forma completa e parcial. Por fim, quando avaliados os requisitos desejáveis, a ferramenta deve atender o maior número possível, mesmo que de forma parcial.

#### **5.1.4 Avaliação, pesquisa e definição de ferramentas**

Com base na definição dos requisitos da ferramenta e do plano traçado, iniciou-se a pesquisa das ferramentas que conseguiriam atender os requisitos da forma desejada. Procurou-se inicialmente chegar a uma lista ferramentas que poderiam alcançar os requisitos. Isto foi possível através de pesquisas realizadas em alguns trabalhos e artigos como: Um processo de verificação e validação para o middleware ginga, Caroca (2010), e Automação Em Testes Ágeis, Silva e Moreno (2011).

Verificou-se rapidamente que com apenas uma ferramenta não seria possível alcançar os requisitos especificados, da forma explicitada na sessão 5.2.2. Para alcançá-los se necessitaria de uma ferramenta para o gerenciamento do que tange os testes de verificação e validação e outra para tratamento das não conformidades. As principais ferramentas elencadas inicialmente foram:

- **Testlink:** Conforme descreve Caroca (2010), o Testlink é uma ferramenta Open Source, de arquitetura WEB, para gerenciamento de casos de teste e a sua execução. Esta ferramenta permite facilmente criar e gerenciar casos de teste, bem como organizá-los em planos de teste e associá-los a requisitos. Esses planos de teste permitem que membros da equipe de teste executem os casos de teste e rastreiem resultados de testes, além de permitir gerar relatórios, gráficos, priorizar e atribuir tarefas. Ela opera

nativamente com sistemas de gerenciamento de não conformidades conhecidos como o Bugzilla, Mantis, Redmine, entre outros (TESTLINK, 2010).

- **Salomé TMF:** Conforme a descrição de Silva e Moreno (2011), o Salomé TMF é uma ferramenta que apoia o processo de teste. Ele permite criar casos de teste, executar testes, registrar os resultados, gerenciar os requisitos e os defeitos do sistema e produzir relatórios de resultados em HTML. Possui integração configurável com o software de controle de não conformidades Mantis.
- **Mantis:** Conforme a descrição de Silva e Moreno (2011), o Mantis é uma ferramenta WEB para controle de não conformidades. Ele possui um controle de *bugs* por projetos e é possível controlar severidade, estado e prioridades. Para tratamento dos *bugs*, existe um controle de usuários e perfis de usuários, trazendo um bom controle para o gerenciador do projeto quanto a responsabilidades e atribuições.
- **Redmine:** Conforme descreve Caroca (2010), o Redmine é um software livre e de código aberto, gerenciador de projetos e ferramenta de gerenciamento de *bugs*. Para tratamento das não conformidades, existe um controle de usuários e perfis de usuários. Permite que documentos sejam anexados e, por ser um software de gerenciamento de projetos também, permite a criação de tarefas de em geral, *wiki*, notícias, entre outras funcionalidades. Ele contém calendário e gráficos de Gantt para ajudar na representação visual dos projetos.

Diante da lista de ferramentas obtida através da pesquisa, procurou-se definir as ferramentas prioritárias para análise, uma para o gerenciamento dos testes de verificação e validação e outra prioritária para o gerenciamento das não conformidades.

Verificou-se através do artigo de Fleischer (2012), que existe uma popularidade grande em organizações que possuem um plano de garantia da qualidade de software pela utilização da ferramenta Testlink. Além disso, esta ferramenta consegue ser integrada com muitas ferramentas de gerenciamento de não conformidades, fazendo com que exista um dinamismo maior no momento de determinar a ferramenta de gerenciamento de não conformidades, conforme

Testlink(2010). Diante disso, a prioridade de escolha para a ferramenta de gerenciamento de testes foi dada à Testlink.

A outra ferramenta necessária, para gerenciamento de não conformidades, seguiu o mesmo princípio de escolha. Ambas as ferramentas citadas acima são integráveis com as ferramentas de gerenciamento de testes elencadas, porém o Redmine apresentou inicialmente um maior número de funcionalidades, visto que além de uma ferramenta para controle não conformidades, é também uma ferramenta de gerenciamento de projetos. Sua interface propicia também uma curva de aprendizado maior ao usuário, visto que é mais amigável.

Seguindo o projeto de avaliação, iniciou-se a verificação dos requisitos e a forma como as ferramentas atendem as necessidades. A primeira ferramenta a passar pela avaliação foi a Testlink. Para a determinação do tratamento dos requisitos por parte do Testlink, utilizou-se principalmente a documentação da própria ferramenta, de Testlink (2010). A segunda ferramenta analisada foi a Redmine. A avaliação desta ferramenta buscou verificar o atingimento dos requisitos do software em uma situação de aplicação do mesmo como gerenciador de não conformidades. Basicamente se utilizou a documentação do próprio Redmine para determinar o atingimento.

O atingimento dos requisitos foi documentado na tabela 2, onde se compara o requisito à ferramenta que atende o requisito proposto.

Tabela 2: Verificação do atingimento dos requisitos na ferramenta Testlink. (continua)

<b>Requisito</b>	<b>Tipo de requisito</b>	<b>Classificação</b>	<b>Atingimento</b>	<b>Ferramenta</b>
Permitir a criação requisitos	Funcional	Recomendado	Completo	Testlink
Permitir a criação de plano de testes	Funcional	Obrigatório	Completo	Testlink
Permitir a criação de casos de testes	Funcional	Obrigatório	Completo	Testlink
Permitir a atribuição	Funcional	Recomendado	Completo	Testlink



de responsáveis				
Permitir classificação do caso de teste quanto à verificação ou validação	Funcional	Desejável	Completo	Testlink
Permitir categorização de testes	Funcional	Recomendado	Completo	Testlink
Permitir classificação por abordagem do teste	Funcional	Recomendado	Completo	Testlink
Permitir informar a estratégia de teste	Funcional	Recomendado	Completo	Testlink
Permitir a definição de cronograma	Funcional	Desejável	Parcialmente	Testlink
Permitir informar parâmetros de entrada	Funcional	Obrigatório	Completo	Testlink
Permitir informar parâmetros de saída	Funcional	Obrigatório	Completo	Testlink
Permitir informar o comportamento esperado	Funcional	Obrigatório	Completo	Testlink
Permitir definir o estado de execução do caso de teste	Funcional	Recomendado	Completo	Testlink
Permitir informar e consultar o resultado do teste	Funcional	Obrigatório	Completo	Testlink
Permitir informar o tempo para execução do teste	Funcional	Desejável	Completo	Testlink
Permitir arquivos em	Funcional	Desejável	Completo	Testlink e

(continuação)

anexo				Redmine
Controle de não conformidades	Funcional	Obrigatório	Completo	Testlink e Redmine
Possuir campo para descrição da não conformidade	Funcional	Obrigatório	Completo	Redmine
Possibilidade de priorização para resolução	Funcional	Desejável	Completo	Redmine
Permitir definir responsável pela não conformidade	Funcional	Recomendado	Completo	Redmine
Possuir controle de estado atual da não conformidade	Funcional	Desejável	Completo	Redmine
Permitir atribuições diferentes por usuário	Funcional	Desejável	Completo	Testlink e Redmine
Possuir relatório para verificação do andamento do projeto	Funcional	Obrigatório	Completo	Testlink
Permitir relacionamento entre caso de testes e requisito	Funcional	Desejável	Completo	Testlink
Possuir funcionalidade para métricas	Funcional	Obrigatório	Completo	Testlink
Arquitetura WEB	Não funcional	Obrigatório	Completo	Testlink e Redmine
Multiusuário	Não funcional	Obrigatório	Completo	Testlink e Redmine
Integração entre	Não	Recomendado	Completo	Testlink e

(conclusão)

ferramentas	funcional			Redmine
<i>Open source</i> ou software livre	Não funcional	Obrigatório	Completo	Testlink e Redmine

Diante da avaliação realizada através da tabelas 2, percebeu-se que ambas as ferramentas atenderam aos critérios definidos no plano de avaliação, estando aptas a serem aplicadas em um projeto protótipo para o laboratório de criação e aplicação de software. Ficou constatado também que nenhum requisito ficou descoberto. Todos os requisitos funcionais são alcançados através de uma das duas ferramentas e os não funcionais por ambas. A partir deste atendimento, descartou-se a avaliação das demais ferramentas elencadas anteriormente, estando assim o autor apto a iniciar a implantação das ferramentas.

## **6 IMPLANTAÇÃO DAS FERRAMENTAS E APLICAÇÃO EM UM PROJETO PROTÓTIPO**

A partir da definição das ferramentas, realizadas na sessão anterior, iniciou-se a implantação das mesmas e a aplicação em um projeto protótipo. A implantação foi baseada em instalação do ambiente e configuração da aplicação para estar preparadas para uma situação semelhante a uma real, vivenciada pelo laboratório. A aplicação consistiu em analisar o projeto escolhido em conjunto com a equipe do laboratório através da documentação disponível e aplicar em um ambiente de homologação as ações propostas nos capítulos anteriores.

No decorrer dos próximos itens deste trabalho, serão descritos a instalação e configuração das ferramentas, brevemente descrito o projeto escolhido e posteriormente demonstrado como as ferramentas foram utilizadas e como os requisitos elencados no projeto de avaliação foram alcançados.

### **6.1 INSTALAÇÃO E CONFIGURAÇÃO DAS FERRAMENTAS**

Verificou-se que ambas as ferramentas possuíam máquinas virtuais preparadas disponibilizadas pela comunidade das próprias ferramentas. Optou-se por utilizar estes ambientes pois possuíam ambientes atualizados, pré-configurados com as configurações sugeridas e de forma gratuita.

A máquina virtual da ferramenta Testlink possuía um sistema operacional Ubuntu 14.04 e a versão da ferramenta pré-instalada era a 1.9.14, versão estável mais recente liberada pela comunidade do Testlink no momento do início deste projeto.

A máquina virtual da ferramenta Redmine possuía também um sistema operacional Ubuntu 14.04 e a versão da ferramenta pré-instalada era a 3.1.1, versão estável mais recente liberada pela comunidade do Redmine no momento do início deste projeto.

Ambas as aplicações foram colocadas em uma estação de homologação. Foram configurados dois IPs diferentes, um para cada uma. O Testlink ficou com o

IP 192.168.168.160 e o Redmine ficou com 192.168.168.161. Esta configuração é importante para que ambas as ferramentas trabalhem em paralelo corretamente e para que seja possível configurar a integração entre ambas. As imagens `tl-logo-transparent-12.5` e `tl-logo-transparent-25` foram alteradas para os logos da Universidade de Caxias do Sul, para se adequar ao padrão visual da organização.

Após as configurações citadas, as ferramentas estavam preparadas para o início da implantação do projeto.

## 6.2 PROJETO BOLSA DE VALORES

O projeto a servir de protótipo para a implantação das ferramentas deste trabalho é uma demanda de cunho educacional, que visa o desenvolvimento de um software de investimentos para promover a percepção de mercado e a capacidade crítica dos alunos em relação ao que é apresentado e percebido como valor. O objetivo principal do software é premiar grupos de trabalho em avaliações feitas pelos professores no processo do semestre e dos grupos durante a rodada de negócios da bolsa de valores. A avaliação busca premiar não somente os melhores trabalhos bem como aqueles que fizeram o melhor investimento. Todos os trabalhos terão um índice atribuído em função do seu desempenho ao longo do semestre.

O projeto serviu como base pois estava com seu desenvolvimento em curso durante a elaboração deste trabalho e possuía mais de uma entrega planejada. Outro ponto crucial na determinação do projeto foi que ele está aplicado ao principal objetivo do laboratório de criação e aplicação de software, que é a elaboração de projetos educacionais que propiciem uma melhora nas ferramentas de aprendizado dos alunos.

No início da implantação, o projeto possuía documentação de análise requisitos e análise de sistema já realizada. A documentação do projeto ainda não estava completa, tornando a aplicação deste projeto ideal também conforme a ordem das tarefas de qualidade criadas para o trabalho.

### 6.3 APLICAÇÃO DO PROJETO













Iniciou-se a implantação das ferramentas pelo Testlink, visto que será responsável pelas primeiras atividades do plano de garantia da qualidade proposto e também pelo fato de possuir maior número de tarefas e requisitos a serem atendidos.

#### a) Criação de usuários

Após a instalação e configuração do servidor desta ferramenta, iniciou-se a configuração dos usuários e suas atribuições. Foram criados seis usuários. Cada usuário, por possuir atribuições diferentes no projeto, adotou um perfil diferente. Os usuários criados foram:

- admin: Usuário administrador da ferramenta. Perfil: admin;
- analistaDeSistemas: Usuário responsável pela execução nos testes de verificação. Perfil: tester;
- gerenteDeProjetos: Usuário para gerenciar os testes. Perfil: leader;
- testador: Usuário responsável pela execução dos testes de validação. Perfil tester.
- testadorGTI: Usuário responsável pela execução dos testes de validação do GTI. Perfil tester.
- usuarioFinal: Usuário final responsável pelos testes de aceite. Perfil: Tester.

Ilustração 22: Usuários criados com seus perfis.

Login	Nome	Sobrenome	E-mail	Perfil	Localização	Ativo
analistaDeSistemas	analistadesistemas	lasis	analistadesistemas.lasis@ucs.br	tester	pt_BR	 
admin	admin	admin	admin@ucs.br	admin	pt_BR	 
gerenteDeProjetos	gerentede projetos	lasis	gerentede projetos.lasis@ucs.br	leader	pt_BR	 
testador	Testador	Test	testador@ucs.br	tester	pt_BR	 
usuarioFinal	Usuário	Final	usuario@ucs.br	tester	pt_BR	 
testadorGTI	Testador	GTI	testadorgti@ucs.br	tester	pt_BR	 

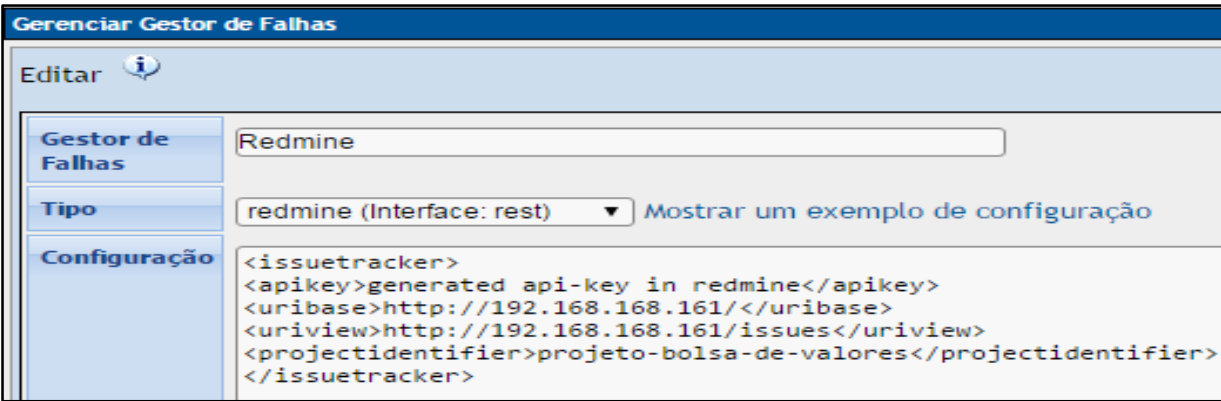
Fonte: Próprio autor

A configuração de usuários realizada demonstra a existência de diferentes permissões por usuário, atendendo assim o requisito “Permitir atribuições diferentes por usuário”.

## b) Configuração da integração, usuários e customizações

Após a configuração dos usuários, configurou-se a integração entre o Testlink e o gerenciador de não conformidades Redmine a fim de atender o requisito “Integração entre ferramentas”.

Ilustração 23: Configuração da integração entre Testlink e Redmine.

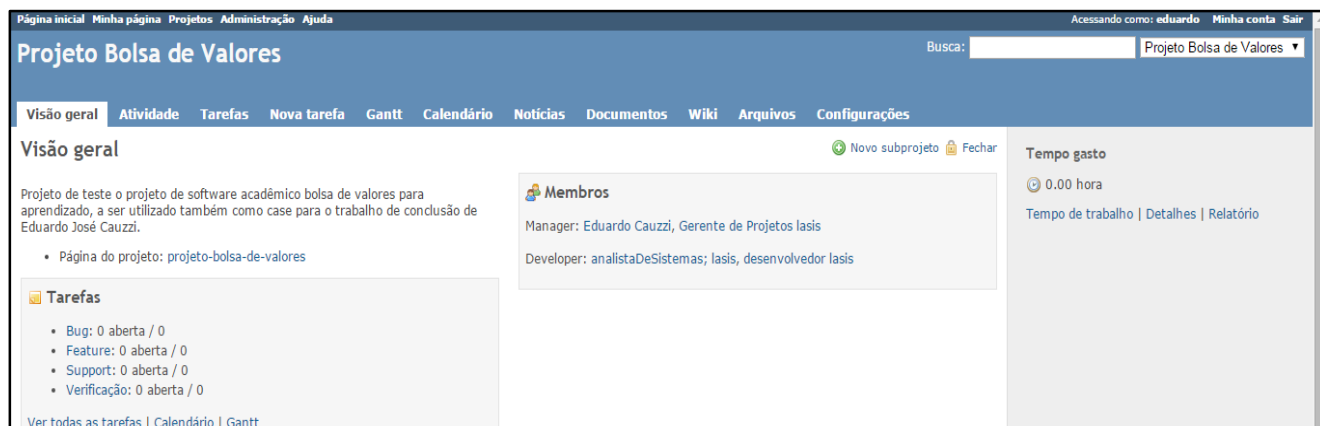


Gerenciar Gestor de Falhas	
<div> <div>Editar</div> <div> <div>Gestor de Falhas</div> <div>Tipo</div> <div>Configuração</div> </div> </div>	
	<div>Redmine</div> <div>redmine (Interface: rest) <a href="#">Mostrar um exemplo de configuração</a></div> <div> <pre> &lt;issuetracker&gt; &lt;apikey&gt;generated api-key in redmine&lt;/apikey&gt; &lt;uribase&gt;http://192.168.168.161/&lt;/uribase&gt; &lt;uriview&gt;http://192.168.168.161/issues&lt;/uriview&gt; &lt;projectidentifier&gt;projeto-bolsa-de-valores&lt;/projectidentifier&gt; &lt;/issuetracker&gt; </pre> </div>

Fonte: Próprio autor

Para que a integração entre as duas ferramentas funcionasse, as atribuições realizadas no Testlink tiveram que ser replicadas no Redmine. Além dos usuários já definidos, foi necessária a criação de um projeto que atendesse ao projeto referenciado na *tag* `<projectidentifier>` apresentado na ilustração 23. A atribuição de papéis no Redmine ocorre por projeto, resultando nas seguintes configurações.

Ilustração 24 – Criação do projeto no Redmine



Fonte: Redmine

Ilustração 25 – Usuários criados e seus papéis no projeto



Fonte: Redmine

Alguns testes nas ferramentas foram realizados antes da criação do projeto de teste para validação dos requisitos que deveriam ser atingidos. Nestes testes verificou-se a ausência do campo ator na ferramenta Testlink, utilizado pela documentação de análise de requisitos e de sistema, porém ausente no cadastro de requisitos. Em leitura da documentação oficial da ferramenta (Testlink, 2010), confirmou-se a ausência deste campo. Diante desta ausência, optou-se pela criação deste campo, através da funcionalidade “Gerenciar Campos Personalizados”. Para isto, foram identificados os possíveis valores que este campo poderia receber através da documentação da ferramenta e se configurou um campo do tipo lista com os valores identificados.



Ilustração 26 – Cadastro campo customizado para requisitos no Testlink

**Campos Personalizados** ?

**Editar - Ator**

Nome	Ator
Rótulo	Ator
Disponível para	Requisito
Tipo	list
Valores Possíveis	Coordenador Professor Outro SISACAD/Outros Sistemas Cc

**Atribuir para os Projetos de Teste**

Projeto Bolsa de Valores

Fonte: Testlink

Ao se testar os indicadores da ferramenta Testlink, constatou-se que elas eram formuladas sempre tomando como base as últimas execuções de testes e não todas as execuções de testes. Isto faria com que as métricas gerais de *bugs* por fase, categoria e estratégia não fossem possíveis da forma como haviam sido propostas. Diante disso optou-se por customizar duas páginas. A primeira foi à página referente ao relatório de “Métricas Gerais do Plano de Teste”, que passou a receber um novo relatório chamado “Erros por Palavra-chave”, a ser demonstrado no decorrer deste trabalho. A outra página customizada foi a chamada “Gráficos”. Optou-se neste caso pela criação de uma nova página, com um código semelhante e com a adição de um gráfico chamado “Erros por palavra-chave”, também a ser demonstrado no decorrer deste trabalho. Ambas as alterações foram realizadas através de código PHP nos arquivos localizados na máquina virtual da ferramenta.

Por fim, foi necessária uma classificação dos casos de teste para atender alguns dos requisitos propostos por este trabalho. A classificação dos testes envolvia tipo, estratégia, fase, entre outros, e foi realizada através de palavras-chave, funcionalidade nativa da ferramenta Testlink que permite com que algumas informações por caso de teste sejam sumariadas nos relatórios e gráficos a serem demonstrados ao longo do trabalho. Este tipo de classificação é recomendada pela própria documentação da ferramenta.

Ilustração 27 – Palavras-chave para classificação dos casos de teste

Palavra-chave	Descrição	Deletar
Abordagem bottom-up	Abordagem de teste bottom-up.	X
Abordagem top-down	Abordagem de teste top-down.	X
Estratégia caixa-branca	Estratégia de teste caixa-branca.	X
Estratégia caixa-preta	Estratégia de teste caixa-preta.	X
Teste de configuração	Categoria de teste de configuração.	X
Teste de desempenho	Categoria de teste de desempenho.	X
Teste de estresse	Categoria de teste de estresse.	X
Teste de funcionalidade	Categoria de teste de funcionalidade.	X
Teste de segurança	Categoria de teste de segurança.	X
Teste de usabilidade	Categoria de teste de usabilidade.	X
Testes de aceite	Testes de aceite	X
Testes de integração	Testes de integração	X
Testes de sistema	Testes de sistema	X
Testes de verificação	Testes de verificação	X

Fonte: Testlink

### c) Criação do projeto de teste

O próximo passo foi criar um projeto de teste no Testlink, obrigatório para o relacionamento futuro dos requisitos. Através do projeto de teste é criada a relação entre o plano de teste e as funcionalidades desejadas. Neste momento, as funcionalidades relevantes selecionadas no cadastro foram a de requisitos, priorização de teste e integração com gestor de falhas, necessárias no processo proposto do laboratório. O cadastro completo pode ser verificado na ilustração 28.

Ilustração 28 – Cadastro do projeto de teste no Testlink

**Editar Projeto Bolsa de Valores** [Visualizar o histórico do Evento]

Nome \*

Prefixo (usado para o ID do Caso de Teste) \*

Descrição do Projeto

Projeto de teste o projeto de software acadêmico bolsa de valores para aprendizado, a ser utilizado também como case para o trabalho de conclusão de Eduardo José Cauzzi.

Funcionalidades

- ☒ Habilitar a funcionalidade de Requisitos
- ☒ Habilitar as Prioridades de Teste
- ☒ Habilitar a Automação de Teste (API keys)
- ☒ Habilitar Inventário

Integração com Gestor de Falhas

- ☒ Ativo
- Gestor de Falhas

Disponibilidade

- ☒ Ativo
- ☒ Público

API Key 486eb0839ef5cbcb15f233cbcf852257ddcbabb63f2e52c8ef14ecb5b7758c29

Fonte: Próprio autor.

Os requisitos foram selecionados para que seja possível relacionar o requisito ao plano de teste, a priorização de teste foi selecionada pois faz parte dos requisitos e traz mais uma forma de gerenciamento dos testes e a integração com gestor de falhas foi selecionada para que seja possível relacionar os casos de testes com falhas de execução com a não conformidade gerada no Redmine.

#### d) Identificação das plataformas

O próximo passo foi a identificação das plataformas existentes neste projeto. As plataformas são ambientes em que a ferramenta deve funcionar ou métodos de testes a serem utilizados. A ferramenta Testlink sumariza os testes vinculados a plataformas em gráficos e relatórios específicos na página de relatórios, a ser demonstrada ainda no decorrer deste trabalho, portanto é importante a definição de plataformas para um melhor controle da execução do projeto.

Analisando o contexto, necessitou-se de uma plataforma padrão para os testes de verificação, que foi chamada de reunião de verificação, e os ambientes mencionados como obrigatórios para o funcionamento do software em desenvolvimento.

Para os testes de validação, a documentação da ferramenta apontava o funcionamento em computadores e em *smartphones*, o primeiro para a primeira entrega e o segundo para a segunda entrega. Diante disso procurou-se criar duas plataformas para cada um dos dispositivos citados, resultando na estrutura demonstrada na ilustração 29.

Ilustração 29 – Plataformas cadastradas no Testlink

Plataforma	Descrição
Reunião de verificação	Alocação de recursos diferentes do elaborador do artefato para uma reunião para...
SmartphoneAndroid	A plataforma de testes deve ser um smartphone com sistema operacional Android.
SmartphoneiOS	A plataforma de testes deve ser um smartphone com sistema operacional iOS.
TesteAutomatizado	Teste automatizado através de ferramenta técnica utilizada pelo desenvolvimento.
Win7Chrome	A plataforma de testes deve ser um computador com Windows 7 e navegador Mozilla Firefox versão 40 ou posterior.
Win7Firefox	A plataforma de testes deve ser um computador com Windows 7 e navegador Mozilla Firefox versão 40 ou posterior.

Fonte: Próprio autor.

### e) Criação do plano de teste e sua estrutura

Finalizada a criação das plataformas, foi necessário criar o plano de teste do projeto para elencar as tarefas de verificação. O plano de teste na ferramenta Testlink serve para agrupar todas as execuções de testes a serem realizadas. O cadastro do plano de teste foi realizado conforme a ilustração 30. Atingiu-se com este cadastro e com os demais a serem realizados na sequência o requisito “Permitir a criação de plano de testes”.

Se comparado com o processo proposto, a criação do plano de teste não deveria estar sendo realizada neste momento do processo. Porém, como apenas informações de cabeçalho serão informadas neste momento para que as tarefas de verificação possam ser executadas, decidiu-se por já criar o plano de teste sem ser alterado o processo proposto no capítulo 4. Isto pois a maior parte das informações necessárias ao plano de teste só serão colocadas na ferramenta após a verificação dos requisitos e análise, na tarefa de definição do processo proposto.

Ilustração 30 – Cadastro do plano de teste no software Testlink

Fonte: Testlink

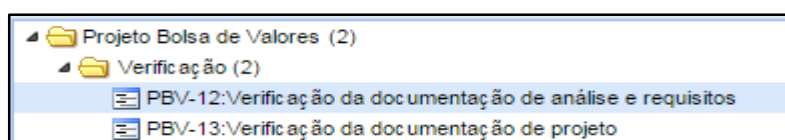
O plano de teste cadastrado e evidenciado na ilustração 30 contempla os atributos de cabeçalho propostos para ele no capítulo 4. As informações detalhadas são contempladas pelas suítes e casos de teste, a serem descritas no decorrer do trabalho.

O plano de teste, em sua estrutura de teste propriamente dita, teve a sua primeira suíte de teste criada. A suíte foi denominada “Verificação” e recebeu já as duas tarefas de teste de verificação da primeira fase do projeto, verificação da documentação de análise e requisitos e verificação da documentação de projeto. As duas tarefas de teste foram elencadas tomando como base o processo de verificação formal proposto no capítulo 4.

A ferramenta Testlink nomeia as tarefas de teste como “casos de teste”, tanto para verificação, quanto para validação. De qualquer forma, ambos os tipos de processos são perfeitamente contemplados independentemente da nomenclatura.

No decorrer dos casos de teste, podem ser colocados os passos do teste. Cada um dos pontos a serem verificados nos documentos foram colocados como passos do teste, contemplando assim os checklists propostos no capítulo 4. Além disso, a ferramenta Testlink nativamente permite a inclusão de anexos nos testes, o que permite que cada tipo de verificação receba os seus respectivos artefatos por anexo. Nas duas próximas ilustrações serão demonstrados a estrutura e os testes propostos, com os passos descritos.

Ilustração 31 – Plano de teste com suíte de verificação no Testlink



Fonte: Testlink

Ilustração 32 – Caso de teste para verificação de requisitos e análise e seus passos

**Caso de Teste**

PBV-12:Verificação da documentação de análise e requisitos

Versão 1

**Objetivo do Teste:**  
Verificação da documentação de análise e requisitos

**Pré-condições**  
Arquivos entregues pelo analista de sistemas.

#	Ações do Passo	Resultados Esperados:	Execução	
1	Verificar se os requisitos estão completos.	Requisitos completos.	Manual	
2	Verificar se os requisitos estão claros.	Requisitos claros.	Manual	
3	Verificar se os requisitos estão simples.	Requisitos simples.	Manual	
4	Verificar se os requisitos estão consistentes.	Requisitos consistentes.	Manual	
5	Verificar se os requisitos estão relevantes.	Requisitos relevantes.	Manual	
6	Verificar se os requisitos estão testáveis.	Requisitos testáveis.	Manual	
7	Verificar se os requisitos estão factíveis.	Requisitos factíveis.	Manual	
8	Verificar se os diagramas UML abrangem os requisitos propostos.	Diagramas UML conforme os requisitos.	Manual	
9	Verificar se os diagramas UML estão no padrão do LASIS.	Diagramas UML no padrão do LASIS.	Manual	
10	Verificar se a arquitetura contempla a aplicação.	Arquitetura de acordo com a aplicação.	Manual	
11	Verificar se os nomes e descrições estão claros.	Nomes e descrições estão claros.	Manual	

Status:  Prioridade:

Tipo de Execução:  Estimated exec. (min):

Palavras-chave: Nenhum

Requisitos : Nenhum

**Relações**

Nova relação: This test case

**Utilização do Plano de Teste**

Versão	Plano de Teste	Plataforma
1	Plano de teste Bolsa de valores	Reunião de verificação

**Arquivos anexados:**

Análise de requisitos - explicação-bolsa.docx (28069 bytes, application/vnd.openxmlformats-officedocument.wordprocessingml.document) 06/11/2015

Análise de sistemas - leito-requisitos.asta (69504 bytes, application/octet-stream) 06/11/2015

Arquivo:  Nenhum arquivo selecionado Title/name:

Fonte: Testlink

Para o teste de verificação do projeto, não ilustrado na ilustração 31, se procurou seguir o mesmo padrão. Os passos de verificação criados foram os seguintes:

- Verificar se os recursos foram estipulados de forma correta;
- Verificar se os riscos existentes foram documentados e antecipados;
- Verificar se as tarefas estão corretamente descritas;

- Verificar se a comunicação entre as partes está prevista;
- Verificar se os custos do projeto estão corretamente descritos e estão coesos com o que se espera praticar;
- Verificar se os marcos do projeto foram estipulados e estão corretos;
- Verificar se o prazo estabelecido pelo cliente será atendido.

Através destas definições, alcançaram-se os requisitos “Permitir classificação do caso de teste quanto à verificação ou validação” e “Permitir arquivos em anexo”. O cadastro dos casos de testes de verificação corroborou com o atingimento dos atributos definidos no plano de teste e provou satisfazer o checklist proposto, ambos definidos no capítulo 4, da seguinte forma:

- Título
  - Contemplado pelo campo “Título do caso de teste”;
- Nome do projeto
  - Contemplado pelo campo “Projeto de teste”;
- Fase do projeto
  - Contemplado por palavras chave e pela suíte de verificação;
- Data da avaliação
  - A ser contemplada pela data de execução do caso de teste;
- Avaliadores
  - A ser contemplado quando os responsáveis forem atribuídos;
- Cada uma das características de qualidade esperadas para o projeto
  - Contempladas pelos passos de teste;
- Verificação do cumprimento
  - Contemplado pelo campo Result, na execução dos testes.

Após definidas as tarefas de verificação, as mesmas foram atribuídas ao responsável por conduzir a verificação dos artefatos, que neste caso foi o usuário analistaDeSistemas, cumprindo assim o requisito “Permitir a atribuição de responsáveis”. Obviamente que, conforme mencionado anteriormente, o usuário analistaDeSistemas no Testlink não é o executor pela análise de requisitos ou pela documentação de projetos e sim um profissional par responsável por conduzir o

processo de verificação. Na ilustração 33 é mostrada a tela onde é realizada a atribuição dos testes de verificação.

Ilustração 33 – Atribuição dos testes de verificação ao responsável

The screenshot shows the 'Atribuir Casos de Teste' (Assign Test Cases) screen in Testlink. At the top, there's a header bar with the text 'Atribuir Casos de Teste para a execução no Build 1a Entrega Bolsa de valores do Plano de Teste Plano de teste Bolsa de valores'. Below this, there's a dropdown menu for 'Todas plataformas' and a button 'Marcar/Desmarcar todos'. A section titled 'Atribuição de usuários em lote:' contains a text input field, 'OK', 'Salvar', and a checkbox. Below that is a checkbox 'Do Bulk user remove'. The main area is titled 'Verificação' and contains a table with the following data:

<input checked="" type="checkbox"/>	Caso de teste [ Versão]	Plataforma	Prioridade	Atribuído a	Atribuir para
<input checked="" type="checkbox"/>	PBV-12 : Verificação da documentação de análise e requisitos [ 1 ]	Reunião de verificação	Médio		analistaDeSistemas x
<input checked="" type="checkbox"/>	PBV-13 : Verificação da documentação de projeto [ 1 ]	Reunião de verificação	Médio		analistaDeSistemas x

Fonte: Testlink

#### f) Simulação de testes de verificação de análise e requisitos

Seguindo o processo, iniciou-se a simulação de um teste de verificação, quando o usuário analistaDeSistemas realizou os passos contidos no teste “Verificação da documentação de análise e requisitos”.

Para simulação do comportamento do sistema, estimou-se que dos onze passos necessários de validação, dois falhassem. Os passos “Verificar se os requisitos estão completos” e “Verificar se os diagramas UML estão no padrão do laboratório” foram os escolhidos para receber o status “Com Falha”. Ao final da execução do teste, informa-se o tempo utilizado para esta execução. A ilustração 34 demonstra como são informados os resultados e o status do teste de cada passo do teste.



Ilustração 34 – Simulação de execução dos testes de verificação

Caso de Teste PBV-12 :: Versão: 1 :: Verificação da documentação de análise e requisitos  
Atribuído à : analistaDeSistemas

**Sumário**  
Verificação da documentação de análise e requisitos

**Pré-condições**  
Arquivos entregues pelo analista de sistemas.

#	Ações do Passo	Resultados Esperados	Execução	Execution notes	Result
1	Verificar se os requisitos estão completos.	Requisitos completos.	Manual	Os requisitos não encontram-se claros.	Com Falha
Arquivo: Escolher arquivos Nenhum arquivo selecionado					
2	Verificar se os requisitos estão claros.	Requisitos claros.	Manual	Os requisitos encontram-se simples.	Passou
Arquivo: Escolher arquivos Nenhum arquivo selecionado					
3	Verificar se os requisitos estão simples.	Requisitos simples.	Manual	Os requisitos estão consistentes.	Passou
Arquivo: Escolher arquivos Nenhum arquivo selecionado					
4	Verificar se os requisitos estão consistentes.	Requisitos consistentes.	Manual	Os requisitos são relevantes.	Passou

Fonte: Testlink

Diante de um caso de falha, optou-se por criar uma *bug* no Redmine, referenciando o teste de verificação que originou a falha, conforme previsto no item 4.1.3. Nativamente, o Testlink possui opção para criar o *bug* (nomenclatura usada por ambas as aplicações para as não conformidades) e tarefas automaticamente no Redmine, porém a funcionalidade está apresentando erros no momento da criação. Diante disso foi criado manualmente *bug* no Redmine e vinculado manualmente este *bug* no teste que falhou, o que funciona corretamente. A operação realizada será demonstrada nas ilustrações a seguir.

Ilustração 35 – Criação do bug no Redmine

Página inicial Minha página Projetos Ajuda

Projeto Bolsa de Valores

Busca:

Visão geral Atividade **Tarefas** Nova tarefa Gantt Calendário Notícias Documentos Wiki Arquivos Configurações

✓ Tarefa #4 criada.

**Bug #4** [Editar](#) [Tempo de trabalho](#) [Observar](#)

**Verificar falhas nos documentos de análise**

Adicionado por analistaDeSistemas; lasis menos de um minuto atrás.

<b>Situação:</b>	New	<b>Início:</b>	27/10/2015
<b>Prioridade:</b>	Normal	<b>Data prevista:</b>	28/10/2015
<b>Atribuído para:</b>	analistaDeSistemasExecutor Lasis	<b>% Terminado:</b>	0%
<b>Categoria:</b>	-	<b>Tempo estimado:</b>	0.30 h
<b>Versão:</b>	-		

**Descrição** [Responder](#)

Ajustar as falhas nos itens "Verificar se os requisitos estão completos" e "Verificar se os os diagramas UML estão no padrão do LASIS", do plano de teste, conforme reunião de verificação

explicação-bolsa.docx - Requisitos (27,4 KB) analistaDeSistemas; lasis, 16/11/2015 23:13 h

levto-requisitos.asta - Análise de sistema (67,9 KB) analistaDeSistemas; lasis, 16/11/2015 23:13 h

**Subtarefas** [Adicionar](#)

**Tarefas relacionadas** [Adicionar](#)

[Editar](#) [Tempo de trabalho](#) [Observar](#)

Exportar para [Atom](#) | [PDF](#)

Fonte: Redmine

Ilustração 36 – Relação entre teste de verificação no Testlink e bug no Redmine

Última execução (baseline qualquer)

Data : 06/11/2015 17:38:35 - Testador : analistaDeSistemas - Baseline : 1a Entrega Bolsa de valores - Status : Com Falha

Última execução (baseline atual) - Baseline : 1a Entrega Bolsa de valores

Data	Baseline	Plataforma	Testador	Status	Exec (min)	Versão	Gerenciamento de Bugs	Exec.
06/11/2015 17:38:35	1a Entrega Bolsa de valores	Reunião de verificação	analistaDeSistemas	Com Falha	100.00	1	<a href="#">Verificar falhas nos documentos de análise</a>	<a href="#">Exec.</a>

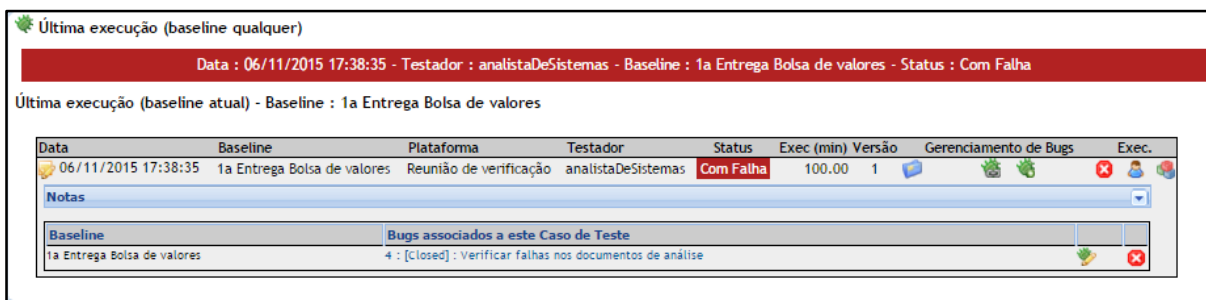
**Notas**

**Bugs associados a este Caso de Teste**

1a Entrega Bolsa de valores 4 : [New] : Verificar falhas nos documentos de análise

Fonte: Testlink

Segundo a simulação da utilização da ferramenta, o usuário analistaDeSistemasExecutor realizou a correção nos artefatos conforme solicitado via *bug*, fechando-o no Redmine. Se verificado no Testlink, o status do *bug* é automaticamente atualizado para *Closed*, representando o status atual do *bug* no Redmine e confirmando que a integração está funcionando corretamente.

Ilustração 37 – Status atualizado do *bug* no Testlink

Fonte: Próprio autor.

Através destes processos, foi possível demonstrar que as ferramentas atenderam aos requisitos “Controle de não conformidades”, “Possuir campo para descrição da não conformidade” e “Possuir controle de estado atual da não conformidade”.

Diante disso, estipulou-se uma nova verificação. Desta vez o checklist ficou com todos os pontos atendidos, completando com sucesso esta etapa. O histórico desta execução está na ilustração 38.

Ilustração 38 – Histórico de verificações do artefato

Data	Testador	Status	Exec (min)	Versão	Gerenciamento de Bugs	Exec.
15/11/2015 11:33:03	analistaDeSistemas	Passou	140.00	1		
15/11/2015 09:42:34	analistaDeSistemas	Com Falha	100.00	1		

Notas

Baseline

1a Entrega Bolsa de valores

Bugs associados a este Caso de Teste

4 : [Closed] : Verificar falhas nos documentos de análise

Diante da aprovação da análise, o processo segue para a documentação do plano de teste, a ser realizado no próximo item. Foi possível também verificar que ambas as ferramentas atingiram o processo proposto de verificação da documentação de análise e requisitos.

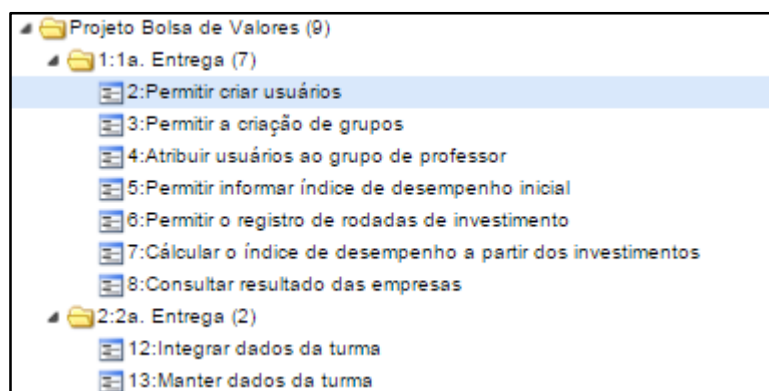
### g) Documentação do plano de teste

Como parte do processo de documentação do plano de teste, a ferramenta sugere que sejam cadastrados antes os requisitos a serem atendidos por tarefas de teste. Vale ressaltar que esta tarefa trata-se de uma replicação do que foi já documentado em uma tarefa anterior. Decidiu-se documentar os requisitos no Testlink somente neste momento e não na tarefa de documentação de análise visto que este cadastro serve para o gerenciamento dos testes na ferramenta de gerenciamento de testes e optou-se por utilizar a última versão verificada do artefato UML para esta replicação.

Portanto, para a documentação dos requisitos, foi decidido por especificar uma parte dos requisitos da primeira entrega e mais dois requisitos da segunda entrega. Não foram especificados todos os requisitos existentes nas documentações de análise justamente por se tratar de um projeto protótipo. Ao todo 9 requisitos foram selecionados.

Como faziam parte de duas entregas, os requisitos especificados foram colocados em duas “especificações de requisitos”. A especificação de requisitos é uma funcionalidade de agrupamento de requisitos da ferramenta, permitindo uma melhor visualização. A organização dos requisitos na ferramenta ficou conforme a ilustração 39.

Ilustração 39 – Organização dos requisitos no software Testlink



Fonte: Próprio autor.

Os requisitos foram criados com as informações de título, descrição/escopo, status, tipo, número de casos de teste necessários, utilizado posteriormente para

cálculo de cobertura, e ator, conforme ilustração 40. Através deste cadastro, o requisito “Permitir a criação requisitos” foi atingido.

Ilustração 40 – Criação dos requisitos no software Testlink

The screenshot shows the 'Permitir criar usuários' requirement form in Testlink. The form includes the following fields and options:

- Salvar / Cancelar** buttons at the top.
- ID do Documento**: A text field containing the value '2'.
- Título**: A text field containing the value 'Permitir criar usuários'.
- Escopo**: A rich text editor area containing the text:
 

Permitir criar usuários do sistema.

O usuário deve possuir o *username* e uma senha.
- Status**: A dropdown menu set to 'Finalizado'.
- Tipo**: A dropdown menu set to 'Função do Sistema'.
- Número de Casos de Teste necessários**: A text field containing the value '1'.
- Ator**: A dropdown menu with the following options:
  - Coordenador
  - Professor
  - Outro
  - SISACAD/Outros Sistemas
  - Coordenador núcleo empreendedorismo

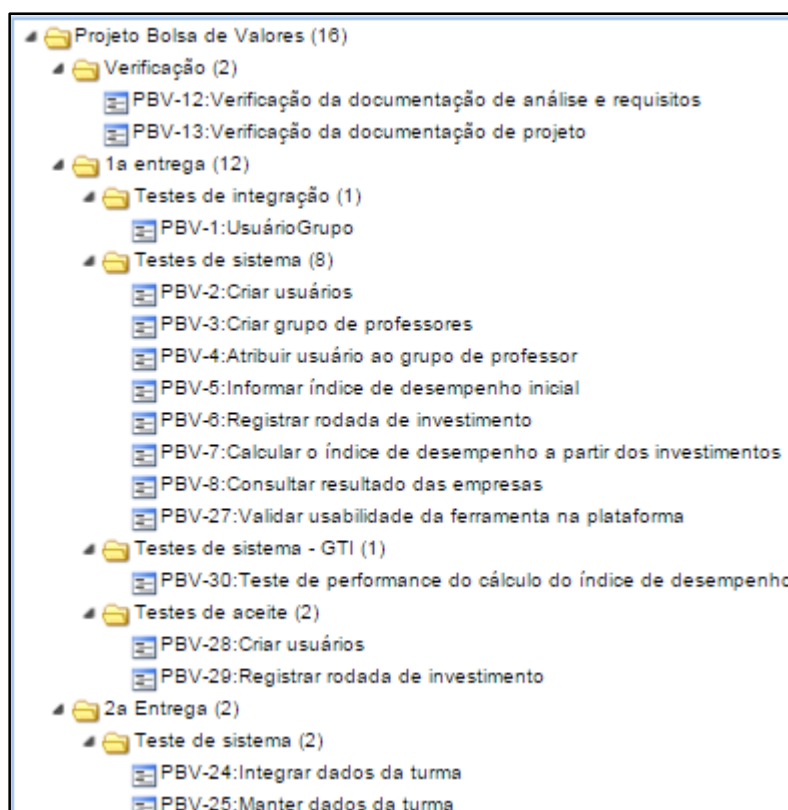
Fonte: Testlink

## h) Criação dos casos de teste

Seguindo a simulação realizou-se a criação dos casos de teste de validação, completando assim o plano de teste já cadastrado para os testes de verificação com o que havia sido documentado como necessário na tarefa “definição do plano de testes”, no capítulo 4.

Cada teste de validação foi colocado em uma suíte que identificava a fase de teste em que se estará trabalhando. Esta suíte, por sua vez, foi colocada outra suíte de nível superior que identifica a entrega a que ela se refere. A ilustração 41 demonstra a estrutura do plano de teste criada. As suítes são os agrupadores, com símbolo de cor amarela à esquerda. Os casos de teste são os itens com símbolo de cor azul, que aparecem hierarquicamente abaixo dos itens em amarelo.

Ilustração 41 – Estrutura do plano de testes



Fonte: Elaborado pelo autor.

Cada caso de teste foi documentado seguindo o padrão ilustrado na ilustração 42. A estrutura utilizada foi a seguinte:

- Objetivo do teste;
- Passos do teste, que orientam o testador exatamente como o teste deve ser realizado, com a ordem das ações;
- Arelados aos casos de teste foram colocados:
  - Parâmetros de entrada;
  - Resultado esperado.

Diante da estrutura definida, foram atendidos os requisitos “Permitir informar parâmetros de entrada” e “Permitir informar o comportamento esperado”. Apenas o responsável pelo teste ainda não foi contemplado neste momento da simulação, porém será demonstrado na sequência como é feita esta atribuição.

Além das atribuições citadas, relacionou-se o caso de teste ao requisito, a fim de se verificar a cobertura dos requisitos pelos casos de teste e atender o requisito

“Permitir relacionamento entre caso de testes e requisito”. Considerando as características citadas neste item e considerando as características citadas na elaboração dos casos de teste de validação, os requisitos das ferramentas “Permitir a criação de casos de testes”, “Permitir categorização de testes”, “Permitir informar a estratégia de teste” e “Permitir classificação por abordagem do teste” foram atingidos.

Ilustração 42 – Estrutura do caso de teste no Testlink

**Caso de Teste**

**PBV-2: Criar usuários**

Versão 1

**Objetivo do Teste:**  
Validar se a criação de usuários está ocorrendo de forma correta.

**Pré-condições**

#	Ações do Passo	Resultados Esperados:	Execução
1	Informar o nome do usuário desejado. Ex: eduardo.cauzzi		Manual <span>✖</span> <span>✔</span>
4	Informar a senha inicial. Ex: 12qwaszx		Manual <span>✖</span> <span>✔</span>
5	Salvar.	Criação do usuário realizada com sucesso.	Manual <span>✖</span> <span>✔</span>
6	Tentar realizar login com o usuário criado.	Login realizado com sucesso.	Manual <span>✖</span> <span>✔</span>

[Criar um passo](#) [Resequenciar Passos](#)

Status: Final Prioridade: Médio

Tipo de Execução: Manual Estimated exec. (min):  [Salvar](#)

**Palavras-chave:** Estratégia caixa-branca ✖  
Teste de funcionalidade ✖  
Testes de sistema ✖

**Requisitos:** [1a. Entrega] 2 : Permitir criar usuários

Fonte: Próprio autor.

Para definir a ordem das fases dos testes de validação foi analisada a complexidade e a criticidade do projeto. Aliado a isto e levando em consideração que se trata de um projeto protótipo, decidiu-se colocar em prática as 4 fases de validação propostas no capítulo 4, sendo elas:

- Testes de integração;
- Testes de sistema;
- Testes de sistema – GTI;
- Testes de aceite.

Para simplificar a demonstração, estipulou-se apenas um teste de integração, um teste de sistema – GTI e dois testes de aceite.

Relacionaram-se cada um dos casos de teste com as plataformas citadas anteriormente. Assim será possível acompanhar o andamento dos testes por plataformas e demonstrar ao testador em qual ambiente está sendo proposto o teste. O exemplo do resultado de atribuição pode ser verificado na ilustração 43.

Ilustração 43 – Atribuição dos casos de teste às plataformas

Utilização do Plano de Teste		
Versão	Plano de Teste	Plataforma
1	Plano de teste Bolsa de valores	Win7Firefox
1	Plano de teste Bolsa de valores	Win7Chrome

Fonte: Próprio autor.

Por fim, atribuem-se os casos de testes definidos anteriormente aos seus responsáveis pela execução, completando os requisitos propostos para esta parte da ferramenta no capítulo 4. A atribuição pode ser acompanhada através da ilustração 44.

Ilustração 44 – Atribuição dos casos de teste aos seus executores

Atribuir Casos de Teste para a execução no Build 2a Entrega Bolsa de valores do Plano de Teste Plano de teste Bolsa de valores

Todas plataformas

Marcar/Desmarcar todos

Atribuição de usuários em lote:

OK

Salvar

Do Bulk user remove

Testes de sistema

<input checked="" type="checkbox"/>	Caso de teste [ Versão]	Plataforma	Prioridade	Atribuído à	Atribuir para
<input type="checkbox"/>	<div><div></div><div></div><div></div></div> PBV-2 : Criar usuários [ 1 ]	Win7Firefox	Médio	<div><div></div> testador</div>	
<input type="checkbox"/>	<div><div></div><div></div><div></div></div> PBV-2 : Criar usuários [ 1 ]	Win7Chrome	Médio	<div><div></div> testador</div>	
<input type="checkbox"/>	<div><div></div><div></div><div></div></div> PBV-3 : Criar grupo de professores [ 1 ]	Win7Firefox	Baixo	<div><div></div> testador</div>	
<input type="checkbox"/>	<div><div></div><div></div><div></div></div> PBV-3 : Criar grupo de professores [ 1 ]	Win7Chrome	Baixo	<div><div></div> testador</div>	
<input type="checkbox"/>	<div><div></div><div></div><div></div></div> PBV-4 : Atribuir usuário ao grupo de professor [ 1 ]	Win7Firefox	Médio	<div><div></div> testador</div>	
<input type="checkbox"/>	<div><div></div><div></div><div></div></div> PBV-4 : Atribuir usuário ao grupo de professor [ 1 ]	Win7Chrome	Médio	<div><div></div> testador</div>	
<input type="checkbox"/>	<div><div></div><div></div><div></div></div> PBV-5 : Informar índice de desempenho inicial [ 1 ]	Win7Firefox	Médio	<div><div></div> testador</div>	
<input type="checkbox"/>	<div><div></div><div></div><div></div></div> PBV-5 : Informar índice de desempenho inicial [ 1 ]	Win7Chrome	Médio	<div><div></div> testador</div>	
<input type="checkbox"/>	<div><div></div><div></div><div></div></div> PBV-6 : Registrar rodada de investimento [ 1 ]	Win7Firefox	Alto	<div><div></div> testador</div>	
<input type="checkbox"/>	<div><div></div><div></div><div></div></div> PBV-6 : Registrar rodada de investimento [ 1 ]	Win7Chrome	Alto	<div><div></div> testador</div>	
<input type="checkbox"/>	<div><div></div><div></div><div></div></div> PBV-7 : Calcular o índice de desempenho a partir dos investimentos [ 1 ]	Win7Firefox	Alto	<div><div></div> testador</div>	



Fonte: Próprio autor.

## i) Verificação da documentação do projeto

Após as tarefas de definição do projeto, simulou-se também a verificação da documentação do projeto. A execução desta tarefa na ferramenta seguiu exatamente o mesmo padrão da verificação de análise e requisitos.

Nesta execução, foi simulada uma situação em que todos os checklists foram considerados terminados com sucesso, com o status “Passou”, padrão da ferramenta. Abaixo segue imagem do relatório de passos de teste com a execução completa, emitido da própria página de casos de teste. Através desta funcionalidade é possível confirmar o atendimento do requisito “Permitir informar e consultar o resultado do teste”.

Ilustração 45 – Histórico de passos da verificação da documentação do projeto

Caso de Teste PBV-13: Verificação da documentação de projeto [Versão: 1]				
Autor:		gerenteDeProjetos - 05/11/2015 22:45:20		
Objetivo do Teste:				
Verificação da documentação de projeto.				
Pré-condições:				
Verificação da documentação de projetos.				
#	Ações do Passo:	Resultados Esperados:	Execution notes:	Execution Status:
1	Verificar se os recursos foram estipulados de forma correta	Recursos estipulados de forma correta.	Os recursos foram estipulados de forma correta.	Passou
2	Verificar se os riscos existentes foram documentados e antecipados.	Riscos documentados e antecipados.	Os riscos foram documentados e antecipados.	Passou
3	Verificar se as tarefas estão corretamente descritas.	Tarefas descritas corretamente.	As tarefas foram descritas corretamente.	Passou
4	Verificar se a comunicação entre as partes está prevista.	Comunicação prevista.	A comunicação entre as partes foi prevista.	Passou
5	Verificar se os custos do projeto estão corretamente descritos e estão coesos com o que se espera praticar.	Custos corretamente estipulados.	Os custos foram estipulados corretamente.	Passou
6	Verificar se os marcos do projeto foram estipulados e estão corretos.	Marcos do projeto estipulados.	Os marcos do projeto foram estipulados.	Passou
8	Verificar se o prazo estabelecido pelo cliente será atendido.	Prazo atendido.	O prazo foi atendido.	Passou
Tipo de Execução:		Manual		
Estimated exec. duration (min):		30.00		
Prioridade:		Médio		
Requisitos		Nenhum		
Palavras-chave:		Testes de verificação		
Execution Details				
Testador		analistaDeSistemas		
Execution Result:		Passou		
Execution Mode:		Manual		
Execution duration (min):		30.00		

Fonte: Próprio autor.

As próximas tarefas do processo de criação e planejamento do projeto não envolviam tarefas de gestão da qualidade de software, terminando assim o processo de criação e planejamento do projeto proposto. As tarefas de desenvolvimento não necessitavam de formalização de execução nas ferramentas de gerenciamento da qualidade, fazendo com que a próxima atividade de validação gerenciada pela ferramenta fosse o teste de integração.

## j) Execução dos testes de integração

O teste de integração definido no plano de teste foi executado pelo analistaDeSistemas, responsável pelo teste conforme documentação do capítulo 4 através de ferramentas automatizadas. Nesta simulação, todos os passos foram realizados com sucesso, fazendo com que o processo de testes de integração fosse encerrado. Pela ilustração 46, além destas informações, pode-se confirmar que a abordagem bottom-up e a estratégia caixa branca foram sugeridas pelo responsável pelo cadastro deste caso de teste, através de palavras-chave.

Ilustração 46 – Execução do caso de teste de integração

Caso de Teste PBV-1: UsuárioGrupo [Versão: 1]				
Autor:		analistaDeSistemas - 10/11/2015 15:46:08		
Objetivo do Teste::				
A atribuição a ser realizada entre os componentes de usuários e grupo de professores está funcionando?				
Pré-condições:				
Usuários já cadastrados;				
Grupos de professores já cadastrados.				
#	Ações do Passo:	Resultados Esperados::	Execution notes:	Execution Status:
1	Simular inclusão de dados de usuários uma lista de usuários selecionados. Ex: Aluno A Aluno B Aluno C			Passou
2	Simular atribuição do professor ao grupo de usuários selecionados. Ex: Professor Ciclano.	Atribuição realizada com sucesso.		Passou
3	Salvar as atribuições no banco de dados.	Atribuições salvas com sucesso.	Informações salvas.	Passou
Tipo de Execução:		Automatizado		
Estimated exec. duration (min):		10.00		
Prioridade:		Médio		
Requisitos		Nenhum		
Palavras-chave:		Testes de integração Abordagem bottom-up Estratégia caixa-branca		
Execution Details				
Testador		analistaDeSistemas		
Execution Result:		Passou		
Execution Mode:		Automatizado		
Execution duration (min):		9.00		
Comentários		Teste de integração realizado com sucesso.		

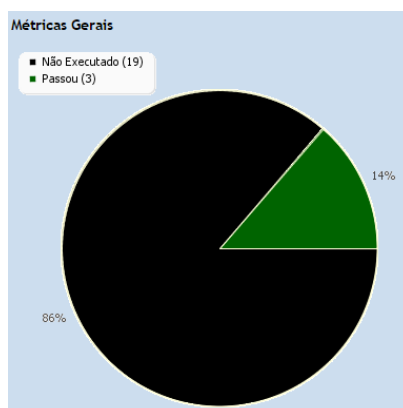
Fonte: Próprio autor.

## k) Acompanhamento do andamento dos testes

Após a demonstração dos testes de integração foi necessário verificar o andamento do projeto através de gráficos. Conforme os gráficos a seguir, pode-se entender que três dos vinte e dois casos de testes foram executados com sucesso. Quando se verifica o gráfico por testes de verificação, pode-se notar que todos os testes de verificação foram executados com sucesso, assim como os testes na

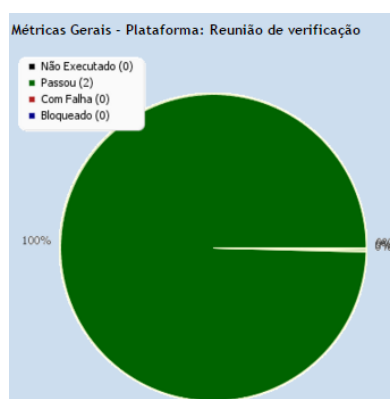
plataforma de teste automatizado. A seguir, as ilustrações 47, 48 e 49 demonstram as conclusões citadas neste item.

Ilustração 47 – Métricas gerais de casos de teste



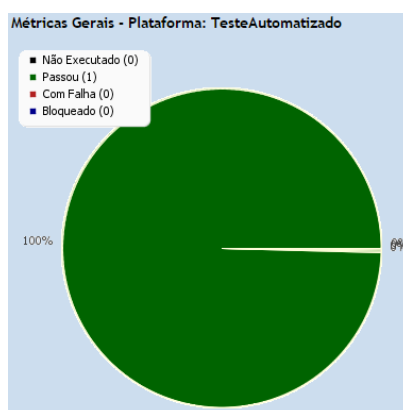
Fonte: Próprio autor.

Ilustração 48 – Métricas gerais de testes de verificação



Fonte: Próprio autor.

Ilustração 49 – Métrica por plataforma: Teste automatizado



Fonte: Próprio autor.

Vale salientar que a ferramenta possui outros gráficos, a serem ilustrados ao longo do texto.

### **I) Execução dos testes de sistema**

Os testes de sistema foram atribuídos ao usuário testador, que simula neste caso um usuário responsável pelos testes na equipe de testes. Cada um dos sete casos de teste de sistema estipulados foram atribuídos, na primeira entrega, a duas plataformas: Win7Chrome e Win7Firefox. Portanto, cada execução de teste deve ser realizada duas vezes, uma para cada plataforma.

Na simulação, dos 14 testes a serem, 2 falharam, sendo eles:

- Criar usuário
  - Caixa-preta;
  - Teste funcional;
  - Plataforma Win7Firefox;
- Registrar rodada de investimento
  - Caixa-preta;
  - Teste funcional;
  - Plataforma Win7Firefox.

Na ilustração 50 é demonstrado o log de execução de teste para o caso de teste “Criar usuários”. O mesmo padrão de execução foi utilizado para a falha no teste “Registrar rodada de investimento”.

Ilustração 50 – Histórico de execução do caso de teste criar usuários

Caso de Teste PBV-2: Criar usuários [Versão: 1]				
Autor:		analistaDeSistemas - 03/11/2015 13:44:22		
Objetivo do Teste::				
Validar se a criação de usuários está ocorrendo de forma correta.				
#	Ações do Passo:	Resultados Esperados::	Execution notes:	Execution Status:
1	Informar o nome do usuário desejado. Ex: eduardo.cauzzi		Usuário informado com sucesso.	Passou
4	Informar a senha inicial. Ex: 12qwaszx		Senha informada com sucesso.	Passou
5	Salvar.	Criação do usuário realizada com sucesso.	Ao clicar no botão salvar, a página perde conexão e cai.	Com Falha
6	Tentar realizar login com o usuário criado.	Login realizado com sucesso.	Não foi possível realizar o login pois o usuário não foi criado ao utilizar o Firefox.	Com Falha
Tipo de Execução:		Manual		
Estimated exec. duration (min):				
Prioridade:		Médio		
Requisitos		2: Permitir criar usuários		
Palavras-chave:		Testes de sistema Teste de funcionalidade Estratégia caixa-preta		
Execution Details				
Testador		testador		
Execution Result:		Com Falha		
Execution Mode:		Manual		
Execution duration (min):		5.00		
Comentários		Teste com falhas ao utilizar o Firefox.		
Bugs reportados		5 : [New] : Criação de usuários falha ao utilizar o firefox		

Fonte: Próprio autor.

Os demais casos de teste foram executados com sucesso. Ambas as falhas foram registradas nos *bugs* 5 e 6 no Redmine. As falhas, registradas pelo testador, seguiram o padrão apresentado na figura 51. Através do padrão apresentado para as não conformidades no Redmine e a vinculação da não conformidade com o caso de teste foi possível atender todos os atributos descritos no capítulo 4, sendo eles:

- Identificação própria
  - Atendido através do campo título no Redmine;
- Identificação do projeto a que se refere
  - Atendido através do projeto em que o *bug* foi criado no Redmine;
- Fase do teste em que foi identificada a inconsistência
  - Atendido através da relação entre o *bug* do Redmine e caso de teste do Testlink;
- Categoria em que foi identificada a inconsistência
  - Atendido através da relação entre o *bug* do Redmine e caso de teste do Testlink;
- Descrição do problema
  - Atendido através do campo descrição no Redmine;
- Data da identificação

- Atendido através do campo início no Redmine e do histórico de execução do teste no Testlink;
- Prioridade para resolução
  - Atendido através do campo Prioridade no Redmine;
- Responsável pela identificação do problema
  - Atendido através do campo testador no Testlink;
- Identificação de referência à ferramenta de controle da resolução
  - Atendido através da relação entre o *bug* do Redmine e caso de teste do Testlink;
- *Status* atual da resolução da situação
  - Atendido através do campo situação no Redmine.
- Responsável pela resolução da não conformidade
  - Atendido pelo campo “atribuído para” no Redmine;
- Possibilidade de priorização para resolução
  - Atendido pelo campo prioridade no Redmine;

Ilustração 51 – Bug criado no Redmine para o erro ao criar o usuário

**Projeto Bolsa de Valores**

Visão geral Atividade Tarefas **Nova tarefa** Gantt Calendário Notícias Documentos Wiki Arquivos

**Nova tarefa**

Tipo \* Bug

Título \* Criação de usuários falha ao utilizar o firefox

Descrição

Boa tarde.

Ao utilizar o navegador firefox, o sistema perde conexão com o banco de dados ao submeter a criação do usuário.

Att.  
Testador

Situação \* New

Prioridade \* Normal

Atribuído para desenvolvedor lasis

Início 2015-11-03

Data prevista 2015-11-04

Tempo estimado 1 Horas

% Terminado 0 %

Arquivos

Erro.png

PrintScreen do erro

Escolher arquivos Nenhum arquivo selecionado (Tamanho máximo: 5 MB)

Criar Criar e continuar Pré-visualizar

Fonte: Próprio autor.

Para verificação do andamento do projeto de testes foram executados novos relatórios e gráficos, sendo eles:

- Na ilustração 52 são demonstrados todos os casos de teste que falharam e que possuem não conformidades (*bugs*) associadas;
- Na ilustração 53 são demonstrados os resultados parciais por suíte de teste;
  - Com este indicador é possível acompanhar o andamento dos testes por entrega.
- Na ilustração 54 são demonstrados os resultados parciais por plataforma;
- Na ilustração 55 foram trazidos os resultados parciais por palavra-chave;
  - Com este indicador é possível acompanhar o andamento dos testes por fase, categoria, estratégia e abordagem.
- Na ilustração 56 o gráfico parcial por palavra-chave
  - Com este gráfico é possível acompanhar o andamento dos testes por fase, categoria, estratégia e abordagem.
- Na ilustração 57, o número total de não conformidades por palavra-chave.
  - Com este indicador é possível acompanhar o número total de não conformidades por fase, categoria, estratégia e abordagem.

Ilustração 52 – Bugs por Caso de Teste

Bugs por Caso de Teste

Projeto de Teste : Projeto Bolsa de Valores

Plano de Teste : Plano de teste Bolsa de valores

Aberto	Resolvido	Total	Casos de Teste com Bugs
2	0	2	2

Expandir/Minimizar Grupos

Mostrar todas as Colunas

Resetar para o Estado Padrão

Refresh

Resetar Filtros

Multiclassificação

Caso de Teste

Bugs do Caso de Teste

Suíte de Teste: Testes de sistema (2 Items)

PBV-2: Criar usuários

5 : [New] : Criação de usuários falha ao utilizar o firefox

PBV-6: Registrar rodada de investimento

6 : [New] : Campo bloqueado no registro da rodada de investimento

Fonte: Próprio autor.

Ilustração 53 – Resultados parciais por suíte de teste

Resultado pela Suite de Teste de Nivel Top										
Suíte de Teste	Total	Não Executado	[%]	Passou	[%]	Com Falha	[%]	Bloqueado	[%]	[%] Completado
2a Entrega	2	2	100.0	0	0.0	0	0.0	0	0.0	0
1a entrega	18	3	16.7	13	72.2	2	11.1	0	0.0	83.3
Verificação	2	0	0.0	2	100.0	0	0.0	0	0.0	100.0

Fonte: Próprio autor.

Ilustração 54 – Resultados parciais por plataforma

Resultados por Plataforma										
Plataforma	Total	Não Executado	[%]	Passou	[%]	Com Falha	[%]	Bloqueado	[%]	[%] Completado
Reunião de verificação	2	0	0.0	2	100.0	0	0.0	0	0.0	100.0
TesteAutomatizado	1	0	0.0	1	100.0	0	0.0	0	0.0	100.0
Win7Chrome	10	3	30.0	7	70.0	0	0.0	0	0.0	70.0
Win7Firefox	9	2	22.2	5	55.6	2	22.2	0	0.0	77.8

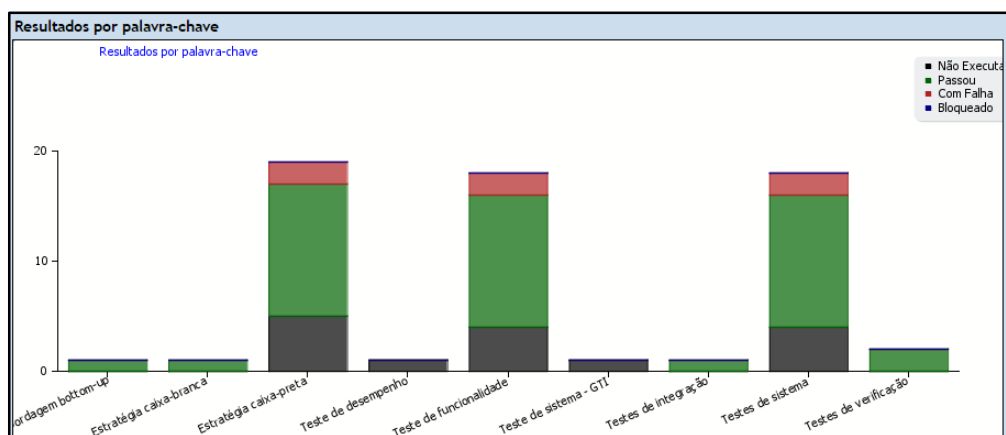
Fonte: Próprio autor.

Ilustração 55 – Resultados parciais por palavra-chave

Resultados por Palavra-chave										
Palavra-chave	Total	Não Executado	[%]	Passou	[%]	Com Falha	[%]	Bloqueado	[%]	[%] Completado
Abordagem bottom-up	1	0	0.0	1	100.0	0	0.0	0	0.0	100.0
Estratégia caixa-branca	1	0	0.0	1	100.0	0	0.0	0	0.0	100.0
Estratégia caixa-preta	19	5	26.3	12	63.2	2	10.5	0	0.0	73.7
Teste de desempenho	1	1	100.0	0	0.0	0	0.0	0	0.0	0.0
Teste de funcionalidade	18	4	22.2	12	66.7	2	11.1	0	0.0	77.8
Teste de sistema - GTI	1	1	100.0	0	0.0	0	0.0	0	0.0	0.0
Testes de integração	1	0	0.0	1	100.0	0	0.0	0	0.0	100.0
Testes de sistema	18	4	22.2	12	66.7	2	11.1	0	0.0	77.8
Testes de verificação	2	0	0.0	2	100.0	0	0.0	0	0.0	100.0

Fonte: Próprio autor.

Ilustração 56 – Gráfico de resultados parciais por palavra-chave



Fonte: Próprio autor.



Ilustração 57 – Bugs totais por palavra-chave

Bugs por Palavra-chave		
Palavra-chave	Com falha	
Estratégia caixa-preta	2	
Teste de funcionalidade	2	
Testes de sistema	2	
Testes de verificação	1	

Fonte: Próprio autor.

Através da análise dos indicadores listados nas últimas ilustrações, o gerente de projetos solicitou ao desenvolvedor que corrigisse as duas não conformidades encontradas nos testes de sistema. Depois de corrigidas, o gerente de projetos solicitou ao testador que executasse novamente os testes, que foram concluídos com êxito. Finalizando assim os testes de sistema.

#### m) Execução dos testes de sistema - GTI

O teste de sistema do GTI foi atribuído ao usuário testadorGTI. Este teste, particularmente, foi classificado como teste de desempenho. Para ser aprovado, o tempo de execução do teste deveria ficar igual ou abaixo do valor informado pelo analista de sistemas na definição do caso de teste no campo “Estimated exec. duration (min)”, destinado a isto pelo Testlink.

Conforme pode ser visto na ilustração 58, o tempo do teste ficou abaixo do tempo máximo estipulado teste de performance com sucesso, considera-se encerrada a fase de teste de sistema – GTI.

### Ilustração 58 – Histórico de execução do teste de performance

Caso de Teste PBV-30: Teste de performance do cálculo do índice de desempenho [Versão: 1]				
Autor:		analistaDeSistemas - 03/11/2015 13:44:22		
Objetivo do Teste::				
Testar a performance do cálculo do índice de desempenho, verificando se atende os pré-requisitos. O tempo de execução deve ser o estimado através do campo (Estimated exec. duration (min)).				
Pré-condições:				
Grupos criados;				
Professores atribuídos;				
Índice de desempenho inicial informado;				
Rodadas de investimento cadastradas.				
#:	Ações do Passo:	Resultados Esperados::	Execution notes:	Execution Status:
1	Executar o cálculo do índice de desempenho.	Tempo igual ou menor ao informado no campo Estimated exec. duration (min).	Execução realizada em tempo satisfatório.	Passou
Tipo de Execução:		Manual		
Estimated exec. duration (min):		10.00		
Prioridade:		Baixo		
Requisitos		Nenhum		
Palavras-chave:		Teste de desempenho Estratégia caixa-preta Teste de sistema - GTI		
Execution Details				
Testador		testadorGTI		
Execution Result:		Passou		
Execution Mode:		Manual		
Execution duration (min):		8.00		

Fonte: Próprio autor.

#### n) Execução dos testes de aceite

Ao final do processo de validação, realizaram-se os testes de aceite. Existiam dois testes de aceite estipulados, ambos de funcionalidade e caixa-preta, sendo eles “Criar usuários” e “Registrar rodada de investimento”. Como o teste de aceite é realizado pelo cliente para verificar o atendimento dos requisitos propostos com o resultado da aplicação, os casos de teste foram atribuídos ao usuário usuarioFinal. Ambas as execuções de teste foram realizadas com sucesso, ou seja, sem nenhuma não conformidade encontrada, encerrando assim os testes de aceite e, por consequência, o projeto de teste da primeira entrega.

#### o) Execução dos testes de sistema – 2ª entrega

Conforme mencionado anteriormente, como este projeto possuía mais de uma entrega prevista aproveitou-se para estipular dois testes de sistema na segunda entrega para demonstrar como o sistema se comporta quando existem mais de uma entrega prevista, para diferentes plataformas, em um mesmo projeto. Os testes escolhidos foram: “Integrar dados da turma” e “Manter dados da turma”. Os casos de testes eram ambos de funcionalidade e caixa-preta. Os dois casos foram atribuídos

às plataformas SmartphoneAndroid e SmartphoneiOS demonstrando que deveriam funcionar em dois sistemas operacionais diferentes para telefones celulares.

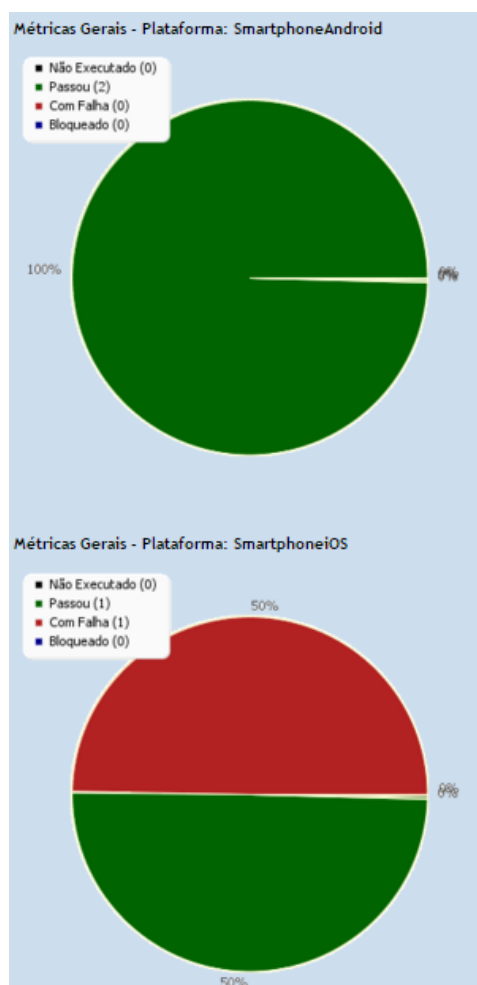
Na simulação, o teste de manter dados da turma falhou para a plataforma SmartphoneiOS, enquanto os demais ocorreram com sucesso. Diante da falha, criou-se a não conformidade 7. Destacam-se os seguintes indicadores parciais entre os demais já demonstrados.

Ilustração 59 – Não conformidades por caso de teste e por status

Bugs per Test Case (All Executions)			
Projeto de Teste: <u>Projeto Bolsa de Valores</u>			
Plano de Teste : <u>Plano de teste Bolsa de valores</u>			
Bugs per Test Case (All Executions)			
Aberto	Resolvido	Total	Casos de Teste com Bugs
1	3	4	4
Expandir/Minimizar Grupos    Mostrar todas as Colunas    Resetar para o Estado Padrão    Refresh    Resetar Filtros    Multiclassificação ↓			
Caso de Teste ▾		Bugs do Caso de Teste	
Suite de Teste: Teste de sistema (1 Item)			
PBV-25: Manter dados da turma		7 : [New] : Manutenção de dados da turma não está funcionando	
Suite de Teste: Testes de sistema (2 Items)			
PBV-2: Criar usuários		5 : [Closed] : Criação de usuários falha ao utilizar o firefox	
PBV-6: Registrar rodada de investimento		6 : [Closed] : Campo bloqueado no registro da rodada de investimento	
Suite de Teste: Verificação (1 Item)			
PBV-12: Verificação da documentação de análise e requisitos		4 : [Closed] : Verificar falhas nos documentos de análise	

Fonte: Próprio autor.

Ilustração 60 – Não conformidades por plataforma



Fonte: Próprio autor.

Após a análise destas métricas, utilizou-se o método já mencionado para resolução do problema e validação da correção. Como o método já foi amplamente apresentado, seguem as ações realizadas de forma sintética:

- Correção da não conformidade 7, pelo desenvolvedor.
- Fechamento do *bug* por parte do desenvolvedor no Redmine.
- Realizado novo teste pelo usuário testador, concluindo que o problema estava corrigido.

Diante do término desta etapa, verifica-se que o plano de teste proposto foi concluído pela equipe com sucesso. Terminado o plano de testes, entende-se que o processo de homologação foi encerrado, bastando o cliente formalize o aceite

através de um documento. Estando formalizado este aceite, o projeto deve ser conduzido aos processos de implantação e encerramento, onde não há intervenção do plano de garantia da qualidade de software proposto.

#### p) Análise das métricas

As métricas geradas pelas ferramentas escolhidas podem ser separadas em duas categorias: métricas de parciais e métricas totais. O que é classificado aqui como métricas parciais são todas as métricas, seja por gráficos ou relatórios, que reproduzem o cenário atual e mais recente do projeto de teste. Já as métricas totais, são as métricas que sumarizam todas as execuções realizadas no plano de testes. Destacam-se nas métricas gerais, as métricas de bugs, a serem exploradas ainda neste item.

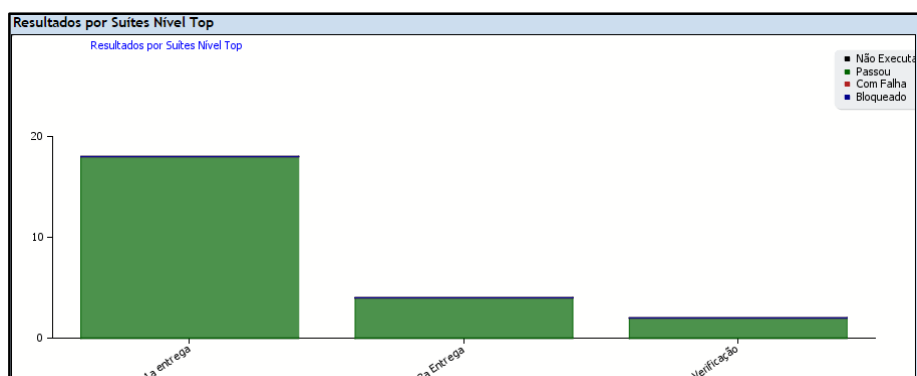
Analisando primeiramente o andamento do projeto, verifica-se através das seguintes métricas, que todos os tipos, abordagens, estratégias e fases de testes acabaram sendo executados com sucesso, conforme ilustração 61, bem como todas as entregas foram realizadas, conforme ilustração 62.

Ilustração 61 – Métrica parcial por palavra-chave



Fonte: Próprio autor.

Ilustração 62 – Métrica parcial por entrega



Fonte: Próprio autor.

Ao se verificar as métricas totais, percebe-se que foram identificadas ao longo da execução do plano de testes, quatro não conformidades, demonstradas na ilustração 63.

Ilustração 63 – Relatório de bugs por caso de teste

Bugs per Test Case (All Executions)

Projeto de Teste : Projeto Bolsa de Valores

Plano de Teste : Plano de teste Bolsa de valores

Bugs per Test Case (All Executions)

Aberto	Resolvido	Total	Casos de Teste com Bugs
0	4	4	4

Expandir/Minimizar Grupos

Mostrar todas as Colunas

Resetar para o Estado Padrão

Refresh

Resetar Filtros

Multiclassificação

Caso de Teste

Bugs do Caso de Teste

Suite de Teste: Teste de sistema (1 Item)

PBV-25:Manter dados da turma

7 : [Closed] : Manutenção de dados da turma não está funcionando

Suite de Teste: Testes de sistema (2 Items)

PBV-2:Criar usuários

5 : [Closed] : Criação de usuários falha ao utilizar o firefox

PBV-6:Registrar rodada de investimento

6 : [Closed] : Campo bloqueado no registro da rodada de investimento

Suite de Teste: Verificação (1 Item)

PBV-12:Verificação da documentação de análise e requisitos

4 : [Closed] : Verificar falhas nos documentos de análise

Fonte: Próprio autor.

Diante da sabida quantidade de não conformidades, desejava-se poder rastrear pelos seguintes critérios:

- O número de não conformidades por fase de teste;
- O número de não conformidades por estratégia de testes;
- O número de não conformidades por categoria de testes;

- O número de não conformidades quando comparadas entre as diferentes fases de teste. Ex: Teste de sistema vs. Teste de integração ou Teste de aceite vs. Demais fases de teste.

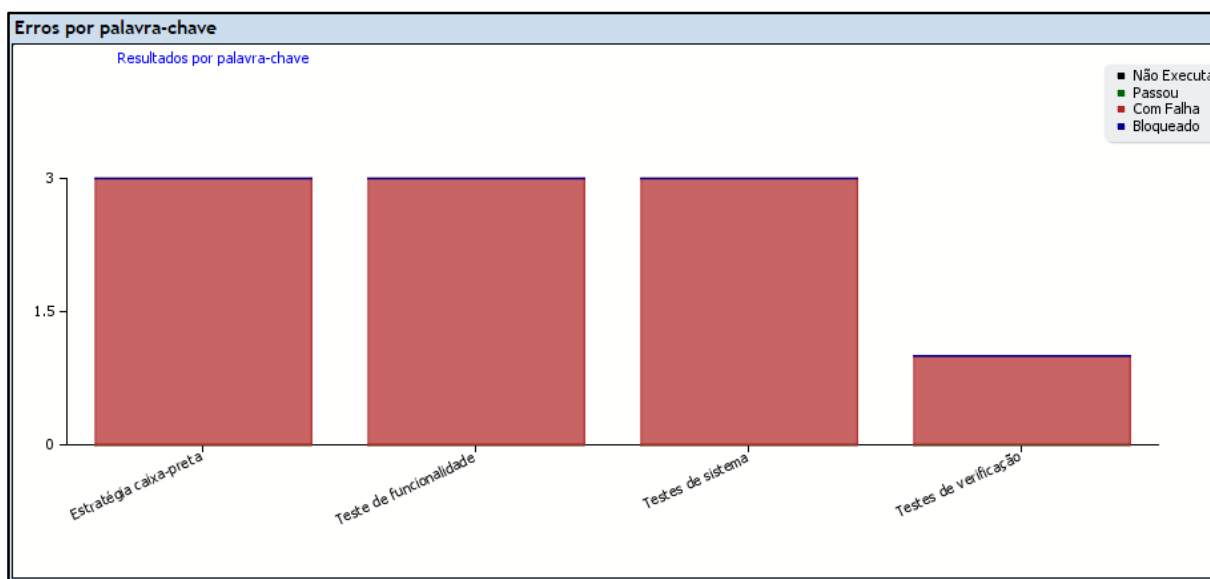
Todas estas métricas são possíveis através do relatório e gráfico apresentados respectivamente nas ilustrações 64 e 65.

Ilustração 64 – Relatório de bugs por palavra-chave

Bugs por Palavra-chave		
Palavra-chave	Com falha	
Estratégia caixa-preta	3	
Teste de funcionalidade	3	
Testes de sistema	3	
Testes de verificação	1	

Fonte: Próprio autor.

Ilustração 65 – Gráfico de bugs por palavra-chave



Fonte: Próprio autor.

Isto é possível visto que quando os casos de teste foram cadastrados, todos eles receberam as classificações quanto à fase, tipo, estratégia e categoria conforme suas características.

Com esta relação realizada e a relação entre não conformidades e casos de teste também, é possível chegar às métricas totais desejadas, podendo concluir que no projeto protótipo:

Não foram identificados não conformidades nos testes de integração enquanto que foram identificados três não conformidades nos testes de sistema. Em um caso de um projeto não protótipo, este ponto é passível de análise para identificar se os testes de integração foram insuficientes em sua quantidade ou em sua qualidade. Analisando este projeto protótipo, fica claro que o problema foi o número de testes de integração estipulados, que causou um maior número de não conformidades na fase seguinte.

Outro ponto de atenção é que todos os testes que retornaram erro, foram de funcionalidade. Nenhum problema de performance ou usabilidade, que estavam dentre os testes estipulados, foi encontrado. Diante disso, o responsável pode verificar se a situação está sendo causada pela falta de um melhor detalhamento dos artefatos de análise ou pela falta de efetividade dos testes unitários, por exemplo.

Os indicadores por estratégia auxiliam a entender qual estratégia de testes está sendo mais efetiva para encontrar não conformidades, o que para o caso do projeto da bolsa de valores foi a estratégia de caixa-preta, com 3.

Os indicadores de não conformidades na fase de verificação, como ocorreu na simulação, podem demonstrar que os analistas necessitam evoluir em alguma área de conhecimento, seja no detalhamento técnico dos diagramas UMLs, seja na entrevista e conversa com o cliente para entender suas reais necessidades, por exemplo.

O indicador de não conformidades nos testes de aceite comparados ao restante das fases de validação é importante para saber quantos erros estão sendo antecipados. No caso deste projeto protótipo nenhuma não conformidade foi detectada pelo cliente, enquanto no processo de validação foram encontradas 3. Isto significa que o processo de homologação foi suficientemente dimensionados e capaz de antecipar os problemas.



## 7 Conclusão

Em um cotidiano em que a vida das pessoas e o funcionamento das organizações é cada vez mais dependente de sistemas informatizados, os softwares possuem uma importância enorme. Um software escrito de maneira correta e que cumpre fielmente os seus objetivos traz benefícios ao provedor e ao consumidor. Ao fornecedor para que procure trabalhar na melhoria contínua de seus produtos e serviços com o menor custo de retrabalho possível. Ao consumidor para que esteja recebendo o que contratou e consiga usufruir e desfrutar das vantagens que os sistemas de informação podem trazer em sua plenitude.

Vários fatores podem interferir na qualidade de um software em construção. Mesmo que o objetivo de um analista, desenvolvedor ou gerente de projetos seja sempre realizar o seu trabalho da melhor forma possível, todos estão passíveis de erros. Não há como evitar que as pessoas cometam equívocos. Eventualmente estes erros podem causar danos consideráveis nas organizações e nas pessoas. No entanto, deve-se garantir que estas falhas não comprometam o resultado final do projeto. Para que exista esta garantia é necessário que o processo de desenvolvimento de software esteja voltado para qualidade. Um processo de software confiável é aquele que ao longo de todo o seu decorrer verifica e valida se as decisões estão sendo tomadas de acordo com o objetivo comum e se as ações que estão sendo realizadas vão de acordo com aquilo que foi acordado. E para se alcançar este processo confiável é necessário um plano de garantia da qualidade de software.

O laboratório de criação e aplicação de software da Universidade de Caxias do Sul, mesmo diante de sua recente criação, está preocupado com a qualidade de seus produtos e serviços para que consiga se consolidar e alcançar os objetivos que almeja. O presente trabalho procurou analisar o processo atual para identificar quais são as características principais no fluxo de trabalho. Diante do entendimento da realidade da organização, procurou-se propor tarefas de verificação e validação para que os artefatos produzidos e os softwares liberados tenham a qualidade que o laboratório de criação e aplicação de software necessita e procura.

Qualquer processo é melhor gerido se possuir ferramentas para isto que lhe tragam benefícios. Um plano de garantia da qualidade, onde novas tarefas são propostas e que estas possuem características diferentes de tarefas de desenvolvimento, necessita de ferramentas adequadas ao processo para uma correta gestão e controle. O presente trabalho, tendo em vista o processo proposto, procurou buscar ferramentas que se encaixam nos padrões e necessidades do laboratório. Procurou-se assim, facilitar a distribuição das operações de qualidade e a gestão da qualidade dos projetos. Procurou-se sugerir, portanto, ferramentas que possuam as informações necessárias para um entendimento do que impacta na análise e desenvolvimento e que permitam a tomada de decisão para mudanças visando a melhoria contínua.

Para que a organização tivesse a certeza de que o processo e as ferramentas propostas estavam de acordo com a sua realidade, procurou-se demonstrar através de um protótipo a aplicação de ambos. Esta demonstração tornou clara como a integração entre o processo e ferramentas pode trazer benefícios. As tarefas estando atendidas pelas ferramentas e estas interagindo entre si trazem agilidade e confiabilidade nas informações.

O presente trabalho contribui para demonstrar quantas variáveis podem estar contidas em um plano plano da garantia da qualidade de software. Como este plano pode ser adequado e implantado mesmo em organizações que não possuem muito tempo de existência. Foi demonstrado que não se necessita que todas as tarefas de um processo sejam verificadas ou validadas para se obter qualidade e que isto vai de encontro com o que cada organização necessita e deseja. Ratificou-se que existem ferramentas de software livre capazes de gerir um plano como este em todas as tarefas e gerar informações e métricas importantes. Demonstrou-se que não conformidades são encontradas ao longo de todo o processo de software e não apenas na codificação, e que quanto antes estas não conformidades são encontradas, menor o custo para resolução. Contribuiu ainda para mostrar que um processo qualificado depende de toda a equipe envolvida.

Para trabalhos futuros, fica a necessidade de ser implantado um processo de auditoria, para garantir que o plano de garantia está sendo cumprido conforme o processo definido. A implantação de uma ferramenta para automação de testes unitários e de integração. Além disso, um maior número de indicadores totais no

Testlink é desejável para uma possibilidade maior de análise retroativa dos projetos. Combinado a isto, indicadores de projeto no Redmine trariam as informações de alocação de recursos importantes para os gestores do laboratório de criação e aplicação de software.

## 8 REFERÊNCIAS

- BARTIÉ, Alexandre. Garantia da qualidade de software. Elsevier, 2002.
- BLANCO, Mariana. Documentação de teste baseado na Norma IEEE 829, 2012.
- de Projeto, P. R. G., de Configuração, J. M. G., de Qualidade, M. S. E., & de Negócios, P. A. C. A. Plano de Testes., 2013
- CAROCA, Caio. UM PROCESSO DE VERIFICAÇÃO E VALIDAÇÃO PARA O MIDDLEWARE GINGA, 2010.
- CROSBY, Philip B. Quality is free: The art of making quality certain. Signet, 1980.
- FLEISCHER, Deborah et al. Evaluation of Open Source Tools for Test Management and Test Automation. Seminararbeit DHBW Stuttgart, 2012.
- GARVIN, David A. What does product quality really mean. Sloan management review, v. 26, n. 1, 1984.
- HÜBNER, Marcos L. F.; BAPTISTA, Michele M.; BERTÉLI, Michele O. Guia para elaboração de trabalhos acadêmicos. Caxias do Sul: UCS, 2012. 83 p. : il. ; 30 cm.
- JUNIOR, Walteno; PRADELA, Izaura; OLIVEIRA, Lucineida. O USO DA NORMA 14598 NA AVALIAÇÃO DE SOFTWARE. 2009.
- KOSCIANSKI, André; SOARES, Michel dos Santos. Qualidade de software. 2006.
- TESTLINK (Org.). Testlink User Manual. 19. ed. San Franscisco: Gnu, 2010. 52 p.
- MAFRA, S. N., & Travassos, G. H., Técnicas de leitura de software: Uma revisão sistemática. XIX Simpósio Brasileiro de Engenharia de Software, SBES, 5. 2005.
- MOLINARI, Leonardo. Testes de software: produzindo sistemas melhores e mais confiáveis: qualidade de software: soluções, técnicas e métodos. Érica, 2008.
- MYERS, Glenford J.; SANDLER, Corey; BADGETT, Tom. The art of software testing. John Wiley & Sons, 2011.
- NITA, E. F., Melhoria da Qualidade de Produto e de Processo de Software a partir da Análise de Indicadores de Teste. Simpósio Brasileiro de Engenharia de Software. Gramado, RS. 2002.

PRESSMAN, Roger S. Engenharia de software. McGraw Hill Brasil, 2011.

PROJECT MANAGEMENT INSTITUTE, Um Guia do conhecimento em gerenciamento de projetos – Quarta edição. 2008.

REDMINE. User Guide. Disponível em: <  
[http://www.redmine.org/projects/redmine/wiki/User\\_Guide](http://www.redmine.org/projects/redmine/wiki/User_Guide)> Acessado em outubro de 2015.

REZENDE, Denis Alcides, Engenharia de Software e Sistemas de Informação. 2005.

RIOS, Emerson; MOREIRA FILHO, Teste de Software. 2013.

SOMMERVILLE, Ian. Engenharia de software. São Paulo: Addison Wesley, 2011.

SILVA, Monique, MORENO Autran. Automação Em Testes Ágeis. 2011.

VASCONCELOS, Alexandre, Plano de Teste - Um Mapa Essencial para Teste de Software. 2012.

WAZLAWICK, Raul, Análise e Design Orientados a Objetos para Sistema de Informação, 3ª Edição. 2010.