FUNDAÇÃO DE ENSINO "EURÍPIDES SOARES DA ROCHA"
CENTRO UNIVERSITÁRIO EURÍPIDES DE MARÍLIA – UNIVEM
CURSO DE BACHARELADO EM SISTEMAS DE INFORMAÇÃO

DANIEL BOTTER DOS SANTOS

IMPLANTAÇÃO DE TESTE DE SOFTWARE EM EMPRESA DE PEQUENO PORTE: UM ESTUDO DE CASO

MARÍLIA
2016

DANIEL BOTTER DOS SANTOS

IMPLANTAÇÃO DE TESTE DE SOFTWARE EM EMPRESA DE PEQUENO PORTE: UM ESTUDO DE CASO

Trabalho de Curso apresentado ao Curso de Bacharelado em Sistemas de Informação da Fundação de Ensino "Eurípides Soares da Rocha", mantenedora do Centro Universitário Eurípides de Marília – UNIVEM, como requisito parcial para obtenção do grau de Bacharel em Sistemas de Informação.

Orientador
Prof. Ms. Allan Cesar Moreira de Oliveira

MARÍLIA
2016

DOS SANTOS, Daniel Botter

Implantação de teste de software em empresa de pequeno porte: um estudo de caso / Daniel Botter dos Santos; orientador: Prof. Ms. Allan Cesar Moreira de Oliveira. Marília, SP: [s.n.], 2016.

78 folhas

Monografia (Bacharelado em Sistemas de Informação): Centro Universitário Eurípides de Marília.

CENTRO UNIVERSITÁRIO EURÍPIDES DE MARÍLIA - UNIVEM
MANTIDO PELA FUNDAÇÃO DE ENSINO "EURÍPIDES SOARES DA ROCHA"

BACHARELADO EM SISTEMAS DE INFORMAÇÃO

Daniel Botter dos Santos

IMPLANTAÇÃO DE TESTE E QUALIDADE DE SOFTWARE EM EMPRESA DE PEQUENO PORTE: UM ESTUDO DE CASO.

Banca examinadora da monografia apresentada ao Curso de Bacharelado em Sistemas de Informação do UNIVEM/F.E.E.S.R., para obtenção do Título de Bacharel em Sistemas de Informação.

Nota: 8,5 (oito e meio)

Orientador:Allan Cesar Moreira de Oliveira _____

1°.Examinador:Fabio Lucio Meira _____

2°.Examinador:Leandro Yukio Mano Alves _____

Marília, 06 de dezembro de 2016.

Dedico este trabalho à todas as pessoas que, de alguma forma, contribuíram para a sua conclusão e formação acadêmica.

AGRADECIMENTOS

Agradeço primeiramente a Deus, que me deu o dom da Vida e me proporcionou a chegar neste momento, me dando a capacidade para aprender e vivenciar esta e outras experiências.

À minha família, que me ajudou e me motivou a chegar até o final deste curso, me dando todo o apoio necessário.

À minha namorada, Jhemili, que me acompanhou, ajudou e incentivou, mesmos nos momentos mais difíceis, para a conclusão deste projeto e curso.

Ao Prof. Ms. Allan Cesar Moreira de Oliveira pelo apoio e orientação que nortearam esse trabalho e impactaram significativamente nos resultados aqui mencionados.

Aos demais professores que, por meio de seus conhecimentos, contribuíram para minha formação acadêmica e me prepararam para o mercado de trabalho.

À AFL Sistemas e seus colaboradores, que me concedeu a oportunidade de trabalhar e utilizar seus recursos e infraestrutura para que este trabalho pudesse ser concluído, agregando assim algum valor a ela mesma.

Aos meus amigos, em especial o André Sabes, Carlos Hordane, Denis Barboni e Rafael Petelin, pelo apoio e companheirismo durante todo o período de curso.

À BestCode, especialmente ao Fagner Paes, que me ajudou com a formação deste trabalho, me dando apoio nos momentos que precisei e me recebendo de braços abertos em sua empresa.

Sonhos determinam o que você quer. Ação determina o que você conquista.
Aldo Novak

SUMÁRIO

INTRODUÇÃO 17

1 CONCEITOS DE TESTE DE SOFTWARE 20

1.1 Técnicas de Teste de Software 23

1.1.1 Técnica Funcional 23

1.2 Outros tipos de teste 28

1.2.1 Teste *ad-hoc* 28

1.2.2 Teste de recuperação de falhas 29

1.2.3 Teste de segurança de acesso 29

1.2.4 Teste de carga 29

1.2.5 Teste de desempenho 30

1.2.6 Teste de portabilidade 30

2 FERRAMENTAS DE TESTE DE SOFTWARE 31

2.1 Ferramentas de gestão de defeitos 31

2.1.1 Mantis 31

2.1.2 Bugzilla 34

2.2 Ferramentas de gerenciamento de teste 36

2.2.1 TestLink 36

3 APLICAÇÃO DA METODOLOGIA DE TRABALHO 38

3.1 Empresa antes 38

3.1.1 Equipe de teste 40

3.1.2 Execução do teste 40

3.1.3 Fechamento de versão 41

3.1.4 Considerações 41

3.2 Empresa depois 41

3.2.1 Equipe de teste 42

3.2.2 Fechamento de versão 42

3.2.3 Padronização do Mantis 43

3.2.4 *Dashboard* de acompanhamento do BugMantis 46

3.2.5 Padronização do Teste 52

3.2.6 Resultados 67

CONCLUSÃO 70

REFERÊNCIAS 72

APÊNDICE A – *SELECTS* UTILIZADAS NOS *DASHBOARDS* 77

LISTA DE ILUSTRAÇÕES

Figura 1 - Relacionamento entre desenvolvimento e fase de teste 21
Figura 2 - Diferença entre defeito, falha e erro 22
Figura 3 - Classes de equivalência, seus conjuntos de elementos e seu estado atual 24
Figura 4 - Notação utilizada para elaboração do Grafo Causa-Efeito 27
Figura 5 - Grafo Causa-Efeito do programa "Cadeia de Caracteres" 27
Figura 6 - Página principal do Mantis 32
Figura 7 - Campos para registro de um caso no Mantis 33
Figura 8 - Resumo dos casos do Mantis 34
Figura 9 - Página inicial do Bugzilla 35
Figura 10 - Ciclo e situações de um caso no Bugzilla 35
Figura 11 - Hierarquia de um projeto no TestLink 37
Figura 12 - BPMN do processo de fechamento de versão 43
Figura 13 - Ciclo de vida de um caso no BugMantis 45
Figura 14 - Filtro aplicado no *dashboard* 46
Figura 15 - Configuração do *dashboard* 52
Figura 16 - Projeto de teste no TestLink 53
Figura 17 - Árvore do TestLink, com Suítes e Casos de Teste 54
Figura 18 - Criando um caso de teste no TestLink 55
Figura 19 - Especificando passo de um Caso de Teste no TestLink 56
Figura 20 - Caso de Teste especificado no TestLink 57
Figura 21 - Plano de teste no TestLink 58
Figura 22 - Atribuindo um Caso de Teste ao Plano de Teste no TestLink 58
Figura 23 - Adicionando vários Casos de Teste ao Plano de Teste no TestLink 59
Figura 24 - Criação da *Baseline* no TestLink 60
Figura 25 - Atribuindo um caso de teste ao testador no TestLink 60
Figura 26 - Atribuindo o testador à vários Casos de Teste no TestLink 61
Figura 27 - Atribuindo prioridade em casos de teste no TestLink 62
Figura 28 - Casos de testes atribuídos para o usuário “daniel.santos” no TestLink 62

Figura 29 - Iniciando a execução do teste em um Caso de Teste no TestLink 63
Figura 30 - Registrando o resultado da execução do teste de um Caso de Teste no TestLink 64
Figura 31 - Resultado após salvar a execução do teste a um Caso de Teste no TestLink 64
Figura 32 - Caso de teste registrado com falha e um caso do BugMantis relacionado 65
Figura 33 - BPMN do fluxo de teste, com o TestLink, Fechamento de Versão e o Mantis 66
Figura 34 - Resultado dos testes de acordo com prioridade 67
Figura 35 - Resultado de acordo com os Suítes de Teste do primeiro nível 68
Figura 36 - Casos de Teste com o *status* e caso do BugMantis vinculado 69
Figura 37 - *Select* usada no *dashboard* com todos os casos 77
Figura 38 – *Sele ct* usada no *dashboard* com os casos de retorno 78

LISTA DE TABELAS

Tabela 1 - Classes de equivalência do programa *Identifier* 25
Tabela 2 - Casos de teste que satisfaz o critério de Análise do Valor Limite 26
Tabela 3 - Tabela de decisão do Grafo Causa-Efeito da Figura 5 28
Tabela 4 - Caso de teste do programa "Cadeia de Caracteres" 28
Tabela 5 - *Status* possíveis de um caso no Mantis 32
Tabela 6 - Casos abertos em 2015 sem resolução 39
Tabela 7 - Projetos e quantidade de casos no *dashboard* 47
Tabela 8 - Tabela de casos no *dashboard* 50
Tabela 9 - Tabela com o cliente e a quantidade de casos relacionados 50
Tabela 10 - Casos com ao menos um retorno 51

LISTA DE GRÁFICOS

Gráfico 1 - Total de chamados por tipo em 2015 39
Gráfico 2 - Casos agrupados por categoria no *dashboard* 47
Gráfico 3 - Casos agrupados por Status no *dashboard* 48
Gráfico 4 - Casos agrupados por prioridade no *dashboard* 49
Gráfico 5 - Casos agrupados por severidade no *dashboard* 49
Gráfico 6 - Resultado final depois da execução dos testes 67

LISTA DE ABREVIATURAS E SIGLAS

BPMN	Business Process Modeling Notation
CGI	Common Gateway Interface
HTML	Hypertext Markup Language (Linguagem de Marcação de Hipertexto)
HTTP	Hypertext Transfer Protocol (Protocolo de Transferência de Hipertexto)
IEEE	Institute of Electrical and Electronics Engineers (Instituto de Engenheiros Elétricos e Eletrônicos)
PHP	Hypertext Preprocessor
QA	Quality Assurance
SEFAZ	Secretaria da Fazenda
TI	Tecnologia da Informação
VV&T	Validação, Verificação e Teste
XML	Extensible Markup Language

RESUMO

A atividade de teste e qualidade de software, nos dias atuais, é imprescindível para que as empresas tenham um produto que seja competitivo no mercado. Todavia, nem todas as empresas, principalmente as micro e pequenas, conseguem praticar essa atividade e se conseguem, praticam de forma incorreta. Isso porque essas empresas não encontram tempo para se planejar e estruturar, para que haja um processo de teste adequado. Em alguns casos, empresas chegam a testar, somente quando existe a necessidade, chamado de teste *ad-hoc*. Sendo assim, o intuito deste projeto é demonstrar que uma empresa pequena é capaz de se estruturar e ter um processo de teste e qualidade de software, sendo apta a tornar seus produtos mais competitivos e tornar-se exemplo de teste e qualidade. Deste modo, é possível realizar a comparação do antes e depois desta empresa, em relação aos seus processos, na qual irá ocorrer uma padronização de testes, utilizando ferramentas *open source* para auxiliar, podendo assim visualizar o impacto que essa atividade causou, desde a sua implantação até o processo de execução dos testes, analisando os problemas encontrados, entre outros parâmetros.

Palavras-Chave: software, teste de software, qualidade de software, padronização, TestLink, ferramenta *open source*

ABSTRACT

The activity of testing and quality of software, nowadays, is essential for companies to have a product that is competitive in the market. However, not all companies, especially micro and small ones, are able to practice this activity and if they do, they practice incorrectly. This is because these companies cannot find the time to plan and structure, so there is an adequate testing process. In some cases, companies come to test, only when there is a need, called the ad-hoc test. Therefore, the purpose of this project is to demonstrate that a small company is able to structure itself and have a process of testing and quality of software, being able to make its products more competitive and become an example of test and quality. In this way, it is possible to perform a set of data before and after the company, in relation to its processes, in the qualitative evaluation of the standardization of tests, using open source tools to help, and can thus visualize the impact that this activity caused, from its Implantation until the process of execution of the tests, analysis of the problems found, among others.

Keywords: software, software testing, software quality, standardization, TestLink, open source tools

INTRODUÇÃO

O teste de software tem por objetivo "[...] analisar um programa com a intenção de descobrir erros e defeitos" (Myers). Seguindo este conceito levantado por Myers, podemos considerar que o teste de software é capaz de reduzir a probabilidade de ocorrer um defeito no produto, além de minimizar riscos para a empresa que o fornece, garantido a qualidade do mesmo.

Com investimento, o teste de software reduz em 70% o índice de retrabalho para a correção de falhas com o produto já no ambiente de produção, além de obter uma melhora no *deploy* de novas versões do produto, com uma grande probabilidade de que as falhas sejam detectadas antes da homologação do produto (SOFTEX, 2011).

Segundo a empresa de consultoria francesa "*Capgemini*", a principal tendência para o mercado de teste de software no Brasil é o investimento em qualidade, onde o Brasil superou em 2014 a marca de 33% dos orçamentos das empresas de Tecnologia de Informação (TI) para os setores de teste de software, contra um total de 26% do resto do mundo (COSTA, 2015).

Hoje, para que uma empresa possa oferecer um produto, considerando que o mesmo está apto para executar suas funções, a estratégia de negócio está no investimento em teste e qualidade de software. Desta forma a empresa garante que seus clientes (e futuros clientes) não troquem seu produto pelo do concorrente.

Para que a empresa possa oferecer um produto de qualidade para seus clientes é necessário aplicar técnicas de teste de software (tipos de teste) e usar ferramentas para auxiliar a equipe de teste e qualidade.

Desta forma, é essencial que a empresa esteja preparada para mudanças nos processos, a fim de agregar valor ao seu produto e melhorar a sua marca no mercado, buscando a competitividade e a produtividade, com qualidade em seus produtos.

Motivação e Justificativa

O teste de software é uma atividade que, embora presente nos dias de hoje, há casos em que empresas, principalmente as que estão iniciando, não possuem ou não conseguem produzir os seus softwares com a qualidade desejada pelo cliente. Muitas vezes, os programadores e os analistas não têm conhecimento adequado para lidar com o assunto,

causando transtornos para a empresa, além de gerar um alto número de chamados ao suporte técnico, visto que, há casos em que muitas falhas não são encontradas durante a execução dos testes.

A atividade de teste de software exige cuidado em relação ao planejamento e aos preparos durante a execução do teste, precisando assim de pessoas e de ferramentas com a capacidade de introduzir o teste de software de forma correta no ambiente da empresa.

Ademais, existe o problema de poucos recursos para as pequenas e médias empresas. Quando os recursos estão em falta, algumas empresas não buscam a implantação de teste e qualidade em seus processos, visto que em alguns casos, isso irá demandar recursos e assim, não conseguem aplicar os testes necessários para seus produtos.

O teste de software é primordial para o ciclo de vida de um software e necessita de pessoas capacitadas para tal função, ferramentas e processos internos adequados. Nem sempre as empresas testam seus softwares adequadamente ou simplesmente não realizam os testes em seus produtos, ocasionando uma queda no desempenho dos sistemas, além de afetar diretamente a velocidade do desenvolvimento e consequentemente a qualidade do produto.

A ideia deste projeto é realizar uma implantação de teste de software em uma empresa de pequeno porte, onde se destaca o estudo de caso, levando em consideração toda sua infraestrutura e recursos em geral, com o foco em técnicas de teste funcional e ferramentas para seu suporte. Com isso, busca-se realizar apresentação de resultados antes e depois da aplicação do processo sistematizado de teste, fazendo o comparativo entre essas condições. Do mesmo modo há a possibilidade de conseguir a eliminação/diminuição de retrabalhos, o que auxilia a redução de custos, bem como obter a satisfação dos clientes, por meio de seu *feedback*.

Objetivos Gerais

Como dito anteriormente, existe a tendência de crescimento de teste e qualidade de software para as empresas de TI no Brasil. Sendo assim, para que o processo de teste funcione, é necessário à sua implantação por meio de técnicas e ferramentas, para que seja possível oferecer um produto de qualidade aos clientes e conquistar reconhecimento.

A principal característica deste trabalho é a descrição de um estudo de caso de implantação de teste e qualidade de software em uma empresa de pequeno porte, composta de cinco a 20 funcionários. A implantação será executada realizando a padronização dos processos de testes, utilizando a ferramenta de gerenciamento de testes *open source* “TestLink”. Também serão apresentados os processos utilizados, com a capacidade de

demonstrar os seus resultados obtidos, tanto antes como depois da implantação, evidenciando tanto o lado positivo quanto o negativo, e desta forma, propiciar outras empresas do mesmo porte a ter um processo de teste e qualidade de software, sem demandar recursos, para que sejam capazes de também oferecer um produto de qualidade e crescerem no mercado.

Objetivos específicos

Os objetivos específicos são:

- Criação de um estudo de caso que demonstre o processo para que micro e pequenas empresas sejam capazes realizarem testes em seus softwares de forma correta.
- Implantação de um processo de teste e qualidade em uma empresa de TI.
- Padronização dos testes, utilizando a ferramenta TestLink.
- Expor os resultados obtidos durante a implantação do teste e qualidade de software (resultados antes e depois).

1 CONCEITOS DE TESTE DE SOFTWARE

Inthurn (2001) definiu que teste é uma das áreas da Engenharia de Software que tem por objetivo aprimorar a produtividade e fornecer evidências sobre a confiabilidade e a qualidade do software em complemento a outras atividades de garantia de qualidade, ao longo do processo de desenvolvimento (INTHURN, 2001).

Com a evolução dos testes nos dias atuais, surge a necessidade de técnicas, métodos e ferramentas para a produção de softwares. Isso se ocorre, pois, a utilização de sistemas de informação tem sido necessária em quase todas as atividades em que envolve o ser humano e/ou sua interação, ou seja, uma forte produtividade e qualidade de sistemas é necessária para que os produtos gerados possam atender o mercado de forma eficaz (DELAMARO *et. al.*, 2004).

Contudo, a construção de software não é uma tarefa fácil, tendo em vista que alguns softwares que são oferecidos no mercado não são testados adequadamente antes de sua entrega ao cliente final, e até mesmo os que são, de alguma maneira, seja ela na forma padronizada ou não, ainda é possível que erros sejam encontrados.

A atividade de Garantia de Qualidade de Software, como por exemplo a atividade de Validação, Verificação e Teste (VV&T), que tem a finalidade de garantir que o produto seja realizado dentro das especificações definidas e com qualidade, ou seja, minimizar e/ou eliminar os erros e riscos que são encontrados, além da entrega correta da construção do produto (MALDONADO *et. al.*, 2007).

A atividade de teste é a atividade que fará a identificação dos erros e, posteriormente, sua eliminação. Essa atividade possui quatro etapas, sendo elas o planejamento de testes, o projeto de casos de teste, execução e avaliação dos resultados dos testes (DELAMARO *et. al.*, 2004).

Segundo MALDONADO *et. al.* (2007), as atividades devem ser executadas ao longo do processo de desenvolvimento do software e se concretizam em quatro fases de teste: de unidade, de integração, de sistema e de aceitação, cada uma delas com objetivos diferentes.

O teste de unidade trabalha com partes menores do sistema, ou seja, é executada em funções e procedimentos mais específicos do software e procura identificar erros relacionados a lógica de implementação, estrutura de dados e simples erros em sua programação.

O teste de integração é realizado após o teste de unidade e é a atividade aplicada na construção do software, ou seja, as funções e procedimentos são posicionados para serem realizados juntos, visando a descoberta de erros entre eles.

O teste de sistema é realizado após a integração do sistema e visa a identificação de erros das funções, verificando se as mesmas estão de acordo com as especificações dos requisitos, levantados antes do início do desenvolvimento.

Por fim, o teste de aceitação está relacionado a um software e pode ser executado pelo usuário final ou cliente, em que durante os testes é possível verificar os requisitos e avaliar se os mesmos foram atendidos de forma correta.

A Figura 1 (CRAIG, 2002), expõe o relacionamento entre as etapas do desenvolvimento e as fases descritas anteriormente.

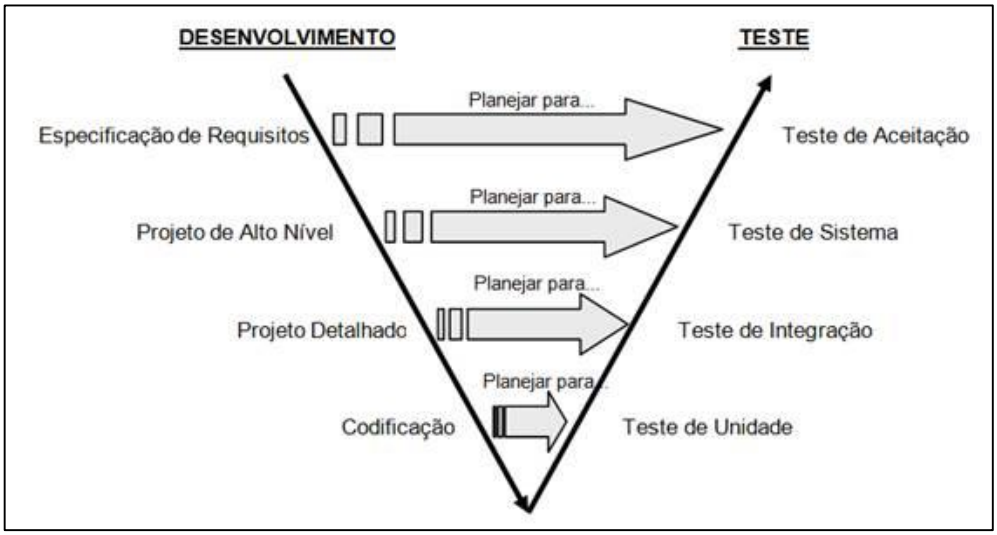


Figura 1 - Relacionamento entre desenvolvimento e fase de teste (CRAIG, 2002)

O *Institute of Electrical and Electronics Engineers* (IEEE), instituto que visa criar padronizações e inovações, cita no padrão de número 610.12-1990 de 1990, a diferença entre os termos: defeito (*fault*), engano (*mistake*), erro (*error*) e falha (*failure*). O defeito está relacionado a algo que está feito/implementado de forma incorreta, sendo que já persiste no software/produto. Engano está relacionado na ação humana que produz um resultado incorreto. Erro é a diferença entre o valor obtido e o valor esperado, ou seja, qualquer estado intermediário incorreto ou resultado inesperado na execução do programa constitui um erro. Falha é o resultado errado, que teve a origem em um defeito. Esses termos acima citados referem-se a um comportamento incorreto de um programa.

A Figura 2 (NETO, 2015) retrata a diferença entre esses conceitos. Defeitos fazem parte do universo físico (a aplicação propriamente dita) e são causados por pessoas, por

exemplo, através do mal-uso de uma tecnologia. Defeitos podem ocasionar a manifestação de erros em um produto, ou seja, a construção de um software de forma diferente ao que foi especificado (universo de informação). Por fim, os erros geram falhas, que são comportamentos inesperados em um software que afetam diretamente o usuário final da aplicação (universo do usuário) e pode inviabilizar a utilização de um software.

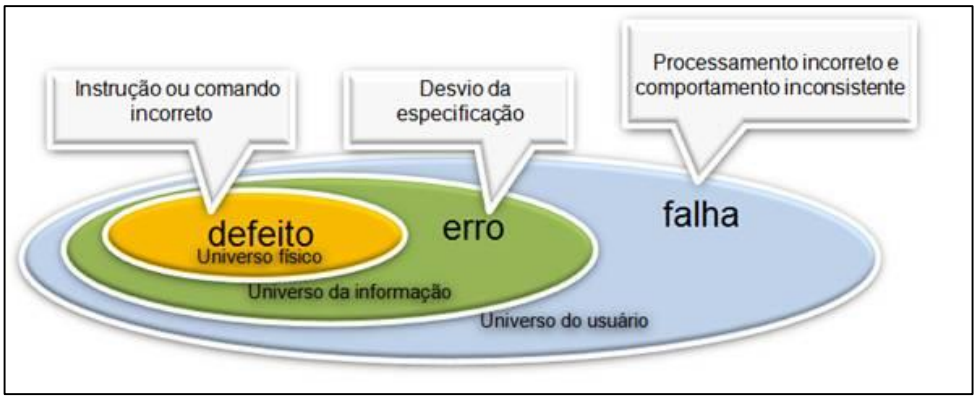


Figura 2 – Diferença entre defeito, falha e erro (NETO, 2015)

Sendo assim, os critérios de teste de software são estabelecidos, a partir de três técnicas, sendo elas a funcional, a estrutural e a baseada em erros. Na técnica funcional, as definições do teste são baseadas no que foi definido no escopo do software, na técnica estrutural, são as características específicas que são levadas em consideração. Já na técnica baseada em erros, são baseados nos erros que são encontrados ao longo do tempo de desenvolvimento do software.

Esses critérios são aplicados nas duas maneiras citados por Howden (1987): teste baseado em especificação (*specification-based testing*) e teste baseado em programa (*program-based testing*). O teste baseado em especificação, ou teste caixa-preta, está relacionado a técnica funcional, em que o foco é testar o software de maneira a verificar as funções do sistema, analisando se atende aos requisitos do escopo. Já o teste baseado em programa, está relacionado a técnica estrutural, baseia-se no conhecimento da estrutura interna da implementação, onde os chamados “caminhos lógicos” do software são testados, que fornecem casos de teste, que são capazes de verificar se existe algum erro com a implementação realizada.

Ademais, destaca-se o uso de casos de testes para a execução das técnicas. Os casos de teste são utilizados durante todo o processo de teste e é por meio deles que os analistas de teste definem a entrada, para que seja possível analisar a saída ou o resultado, para a validação do teste.

Desta forma, as técnicas e critérios de teste de software são importantes para a execução do teste de forma correta e objetiva, com o auxílio dos casos de teste. Na subseção 1.1, será abordado com mais abrangência este assunto.

1.1 Técnicas de Teste de Software

Conforme citado anteriormente, existem as técnicas de teste de software funcional e estrutural. Este trabalho está relacionado com a técnica de teste funcional; a seguir o mesmo será apresentado com mais detalhes, descrevendo suas características.

1.1.1 Técnica Funcional

A técnica funcional é utilizada para a realização de testes em nível de usuário, ou seja, são feitas as entradas dos dados no software e são analisadas as saídas, observando se estão corretas, de acordo com o que foi estipulado pelo escopo do projeto/desenvolvimento. Essa técnica é conhecida como caixa-preta por isso, em que seu conteúdo é desconhecido e só é possível visualizar sua parte externa (PRESSMAN, 2005).

Nesta técnica é possível descobrir os erros de um software, porém, exigirá inúmeras entradas de dados para a realização deste feito. É uma pratica chamada de teste exaustivo, que é uma atividade não recomendável, considerando o volume de dados de entrada e de tempo que demanda para sua execução (MALDONADO *et. al.*, 2007).

> [...] Teste Exaustivo correspondem a exercitar o programa com todos os valores possíveis do domínio de entrada. (FELIZARDO, 2014).

Com essa limitação da atividade, fez-se necessário criar regras ou critérios para que se possa praticar essa técnica. Algumas delas são: Particionamento de Equivalência; Análise do Valor Limite; e Grafo de Causa-Efeito.

Além dos critérios, o teste funcional tem dois passos principais. O primeiro está em identificar as funções que o software deve realizar. Essas funções são identificadas com a utilização do escopo que foi realizado anteriormente. O segundo está em criar os casos de testes, para que possam verificar se as funções estão sendo realizadas corretamente. Vale lembrar que especificações ausentes ou incompletas, podem resultar em casos de testes falhos e tornarão difícil a realização do teste (DELAMARO *et. al.*, 2004).

Em relação aos critérios desta técnica, elas serão descritas a seguir.

1.1.1.1 Particionamento de Equivalência

O particionamento de equivalência faz com que todos os possíveis dados de entrada sejam separados em classes de equivalência. Essas classes de equivalência podem ter um ou *n* dados de entrada, que variam de acordo com a especificação do software (DELAMARO, 2004).

Após isso, pode-se dizer que qualquer elemento de uma classe de equivalência pode ser considerado um representante dela, pois, estão juntos por existir algo similar entre eles. Desta forma, se um elemento encontrar algum defeito, qualquer outro também irá detectar e o contrário também, se um deles não detectar, os demais também não irão detectar. Em alguns casos, quando elementos de uma mesma classe não se portam da mesma maneira, há a necessidade de uma redução naquela classe de equivalência. Essa redução é o ato de tornar os dados da classe de equivalência distintos, separando os que se comportam de maneira igual, criando outras classes de equivalência (MALDONADO *et. al.*, 2007).

Isso é o que este critério faz, tenta reduzir ao máximo o número de possíveis entradas para algo que seja viável, sendo que, poderá ser tratado durante toda a atividade do teste. Além disso, cada classe de equivalência representa um conjunto de estados válidos ou inválidos para as entradas, conforme a Figura 3 (RIBEIRO, 2011).

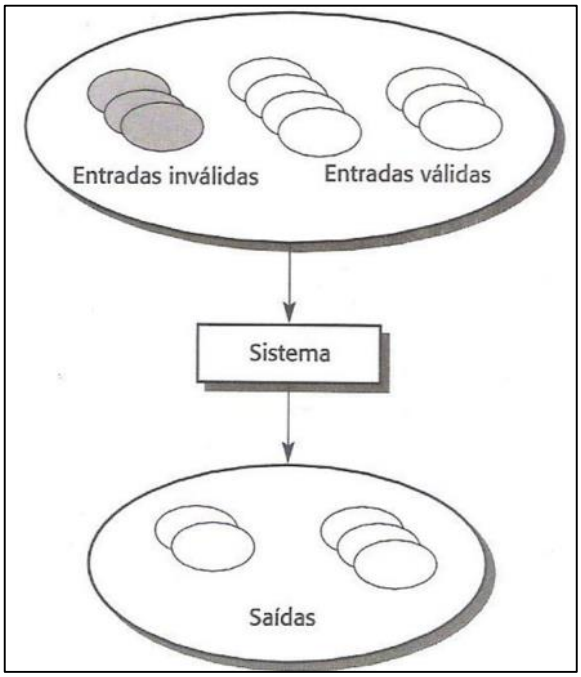


Figura 3 - Classes de equivalência, seus conjuntos de elementos e seu estado atual (RIBEIRO, 2011)

De acordo com os dados apresentados anteriormente sobre o critério e sobre a

técnica, segue um exemplo com o cenário do programa *Identifier*, de DELAMARO *et. al.* (2004).

> O programa deve determinar se um identificador é ou não valido em '*Silly Pascal*' (uma estranha variante do Pascal). Um identificador valido deve começar com uma letra e conter apenas letras ou dígitos. Além disso, deve ter no mínimo um caractere e no máximo seis caracteres de comprimento. (DELAMARO *et al.*, 2004).

Seguindo este cenário, tem-se os seguintes exemplos de entrada:

- wxy123 (válido);
- ghj*34 (inválido);
- 1qwert (inválido);
- a1b2c3d4 (inválido);

Com isso, é possível que seja definido as classes de equivalência. A Tabela 1 (RAMOS, 2009) expõe as classes de equivalência identificadas.

Tabela 1 - Classes de equivalência do programa *Identifier* (RAMOS, 2009)

Condições de entrada	Classes válidas	Classes inválidas
Tamanho da cadeia de caracteres	$1 \leq t \leq 6$ (1)	$t > 6$ (2)
Primeiro caractere *c* é uma letra	Sim (3)	Não (4)
Só contém caracteres válidos	Sim (5)	Não (6)

1.1.1.2 Análise do Valor Limite

O critério de análise do valor limite é muito similar ao critério de particionamento de classes, porém, a escolha de qualquer elemento de uma classe não é praticada (DELAMARO *et. al.*, 2004).

Segundo Bartié (2002), a análise de valor limite tem duas diferenças de acordo com o particionamento de classes.

O primeiro está relacionado com a seleção do elemento de uma classe de equivalência, como dito anteriormente, pois devem ser selecionados os casos de teste de cada classe, sendo eles os valores-limite. Isso porque, segundo o autor, os erros do software estão mais relacionados com os dados de entrada. O segundo está em considerar, além dos dados de entrada, o conjunto de saídas que são apresentados pelos casos de testes.

Para efeito de exemplo, a Tabela 2, com base nas classes de equivalência do critério Particionamento de Classes (Tabela 1), destaca como deve ser os casos de testes que correspondem a este critério.

Tabela 2 - Casos de teste que satisfaz o critério de Análise do Valor Limite (Fonte própria)

Entradas		Saídas
T	CC	
7	a1b2c3d4	Inválido
6	1qwert	Inválido
6	ghj*34	Inválido
6	wxy123	Válido

Considere **T** como o tamanho do comprimento dos caracteres e **CC** como os dados de entrada.

1.1.1.3 Grafo de Causa-Efeito

O critério Grafo de Causa-Efeito está voltado em explorar as combinações dos dados de entrada, ou seja, todos as situações de um software podem ser testadas com a utilização deste critério (DELAMARO *et. al.,* 2004).

O grafo de causa-efeito é dividido nas ações de extração das causas e efeitos, sendo que as causas estão relacionadas com o domínio de entrada e os efeitos, com as saídas esperadas, podendo enumerar as causas e os efeitos de um a *n*. Logo após, o grafo é desenvolvido, com a ligação das causas com os efeitos, utilizando a notação do critério da Figura 4 (DELAMARO *et. al.,* 2004). Por fim, realiza-se a conversão do grafo em uma tabela de decisão que, posteriormente, é construído os casos de testes.

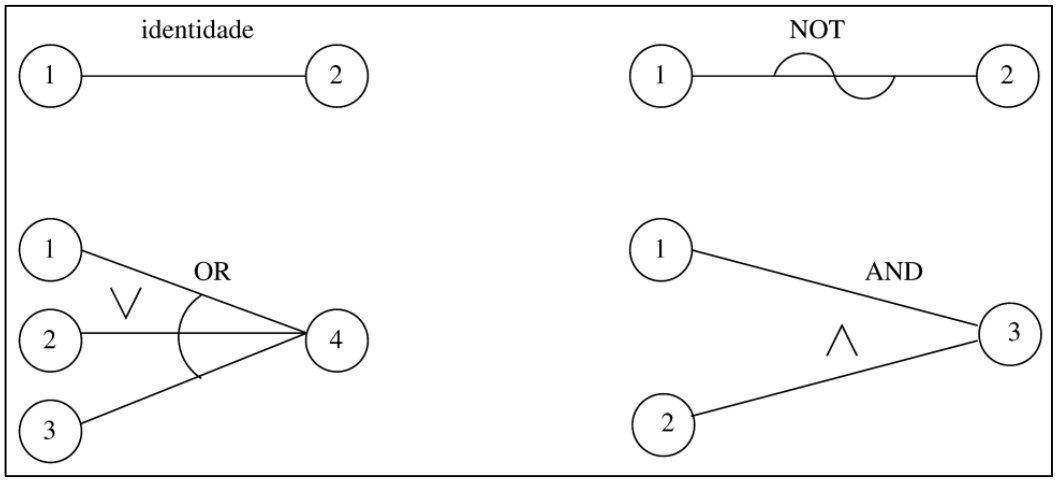


Figura 4 - Notação utilizada para elaboração do Grafo Causa-Efeito (DELAMARO *et al.*, 2004)

Além da notação nominal, existe também a notação de restrição do grafo de causa-efeito.

Para a elaboração da tabela de decisão, deve ser destacado que cada nó do grafo pode assumir o valor de 0 ou 1, que representa a existência ou inexistência de um vínculo entre a causa e o efeito (MALDONADO *et. al.,* 2007). Há quem utilize o termo booleano (verdadeiro [V] e falso [F]) para a representação.

De acordo com o programa "Cadeia de Caracteres" de MALDONADO *et al.* (2007), a Figura 5 exprime o Grafo Causa-Efeito e a Tabela 3 expõe a tabela de decisão deste programa exemplo.

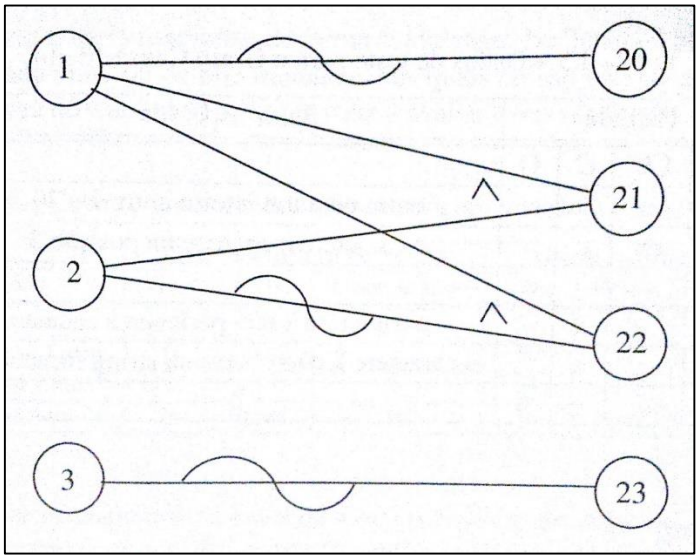


Figura 5 - Grafo Causa-Efeito do programa "Cadeia de Caracteres" (MALDONADO, 2007)

Tabela 3 - Tabela de decisão do Grafo Causa-Efeito da Figura 5 (MALDONADO, 2007)

1	**F**	**V**	**V**	**-**
2	**-**	**V**	**F**	**-**
3	**-**	**V**	**V**	**F**
20	**X**	**-**	**-**	**-**
21	**-**	**X**	**-**	**-**
22	**-**	**-**	**X**	**-**
23	**-**	**-**	**-**	**X**

Contudo, segue o caso de teste para este cenário na Tabela 4.

Tabela 4 - Caso de teste do programa "Cadeia de Caracteres" (Fonte própria)

Entrada	**Válido**	**Inválido**
Intervalo (I)	**(C1) I ∈ {1-20}**	**(C2) I ∉ {1-20}**
Char (C)	**(C3) C ∈ {a-Z}**	**(C4) C ∉ {a-Z}**
Opção (O)	**(C5) O ∈ {0-1}**	**(C6) O ∉ {0-1}**
#	**Classes**	**Entrada**
Válida	**C1**	**6**
Válida	**C3**	**D**
Válida	**C5**	**0**
Inválidas	**C2**	**25**
Inválidas	**C4**	**$**
Inválidas	**C6**	**2**

1.2 Outros tipos de teste

A seguir serão descritos outros tipos de testes de software, que são relevantes para esta monografia.

1.2.1 Teste *ad-hoc*

O teste *ad-hoc* são testes realizados de modo informal, na qual não há nenhuma preparação de teste, não é utilizada nenhuma técnica de projeto de teste reconhecida, sendo possível mencionar a exatidão dos resultados, bem como os testes são executados de forma aleatória, até que se encontre erros (COMITÉ FRANÇAIS DES TESTS LOGICIELS, 2007).

Este tipo de teste é considerado informal. Sendo assim, o teste *ad-hoc* é criticado por não possuir uma estrutura a ser seguida, na qual o teste é realizado com improvisação, sendo que o testador procura encontrar os erros de qualquer maneira que lhe pareça apropriado, dependendo de si mesmo para que o teste obtenha êxito.

1.2.2 Teste de recuperação de falhas

Teste de recuperação de falhas é um teste utilizado para verificar a força e também a capacidade de um determinado software para retornar a um estado operacional após estar em um estado de falha, ou seja, testar quão bem o software se recupera de falhas e de outros problemas similares.

O teste em pauta força falhas no software de diversas maneiras para verificar se a recuperação é adequada. Esta recuperação é a capacidade de reiniciar operações após a perda da integridade de uma aplicação. Isto garante que o sistema possa se recuperar de uma situação de falha de *hardware*, software, mal funcionamento da rede, entre outros.

O teste tem por objetivo verificar se o processo de recuperação (manual ou automático) recupera os dados corretamente e se o sistema se comporta de maneira correta.

1.2.3 Teste de segurança de acesso

O Teste de Segurança de acesso é utilizado para testar sistemas que gerenciam informações importantes ou causam ações que podem prejudicar/beneficiar indivíduos. Os testes tentam verificar se os mecanismos de proteção irão, de fato, proteger de acessos indevidos.

Deste modo, o testador simula o papel do indivíduo que deseja acessar o sistema, devendo tentar adquirir senhas por meios diversos e pode também sobrecarregar o sistema, danificando o serviço de outros usuários.

Objetivo do teste é garantir que o sistema se comporte adequadamente mediante tentativas ilegais de acesso, fazendo com que somente pessoas autorizadas tenham acesso às determinadas informações.

1.2.4 Teste de carga

O teste de carga tem o objetivo de avaliar o software, identificando se o mesmo suporta altas capacidades de transações de dados. Este teste é realizado de acordo com situações reais de uso, visando buscar um equilíbrio, verificando até que ponto o software pode atingir, sem deixar de funcionar.

Além disso, erros como de infraestrutura e de desenvolvimento inadequado poderão ser encontrados, isso porque o seu foco é em todo o ambiente de produção, visando

atingir o seu objetivo máximo.

1.2.5 Teste de desempenho

O teste de desempenho, como o nome já diz, é focado para testar o desempenho do software durante sua execução. Seu objetivo é identificar as situações em que o software possa vir a falhar, verificando a sua confiabilidade, escalabilidade e estabilidade.

Para que o teste seja efetivo, o ambiente de teste deve-se parecer muito com o ambiente de produção, onde todos os demais processos estarão integrados.

1.2.6 Teste de portabilidade

O teste de portabilidade tem como principal objetivo de verificar se o software tem a capacidade de trabalhar em diversos ambientes e situações. Em outras palavras, este teste verifica se o software é capaz de trabalhar com diversos dispositivos de *hardware* e com outros softwares em um único ambiente.

Este tipo de teste consiste em avaliar algumas das seguintes característica de uma aplicação: Adaptabilidade, analisando se o software é adaptável a outras plataformas; Coexistência, avaliando se o software consegue compartilhar seus recursos com softwares independentes; e Capacidade de substituição, avaliando se o software se capaz de se substituir por outro.

2 FERRAMENTAS DE TESTE DE SOFTWARE

Com o desenvolvimento do teste de software e de suas técnicas e atividades, existe a necessidade de ferramentas para acompanhar este avanço. As ferramentas de teste são muito importantes para se trabalhar com as técnicas, a fim de uma melhor avaliação dos testes. O seu principal objetivo é simplificar as atividades envolvidas visando um melhor resultado.

Sendo assim, neste capítulo, serão abordadas algumas ferramentas existentes no mercado, que sendo fundamentais para a elaboração deste projeto. As ferramentas que serão mencionadas são *Open Source*, e descreverão algumas de suas funcionalidades e características.

Serão apresentadas ferramentas de gestão de defeitos (ou *Bugtrackers*) e ferramentas de gerenciamento de testes.

2.1 Ferramentas de gestão de defeitos

As ferramentas de gestão de defeitos têm o fundamental objetivo de registrar os *bugs* que são encontrados durante toda vida do software, ou seja, quando houver manutenção, deve-se haver os registros. Além disso, essas ferramentas conseguem permitir a rastreabilidade das mudanças que aconteceram ao longo do projeto.

A seguir serão descritas as ferramentas *Open Source*, Mantis e Bugzilla.

2.1.1 Mantis

A ferramenta de gestão de defeitos Mantis é um sistema *Web*, desenvolvido na linguagem *Hypertext Preprocessor* (PHP), utilizando banco de dados MySQL, SQL Server ou PostegreSQL.

Uma das principais características do Mantis é a sua capacidade de acompanhar e gerar estatísticas das mudanças. Além disso, quando houver a modificação de um caso, isso é registrado e enviado para a equipe, em que o gerente pode acompanhar a execução, sendo notificado por meio e-mail.

A Figura 6 estampa a página inicial do Mantis em um ambiente real.

Figura 6 – Página principal do Mantis (AFL SISTEMAS, 2016)

Nesta área principal da ferramenta, é possível filtrar os casos por projeto. Além disso, o Mantis destaca em cores cada status de um caso.

A Tabela 5 expõe os possíveis status de um caso no Mantis.

Tabela 5 – *Status* possíveis de um caso no Mantis (MANTIS BT TEAM)

Status	Situação
Novo	**Quando o caso está em aberto.**
Atribuído	**Quando o caso está atribuído a alguém da equipe.**
Retorno	**Quando o caso foi retornado para o relator ou alguém da equipe.**
Reconhecido	**Quando o caso será realmente desenvolvido.**
Confirmado	**Quando o caso está confirmado e inicia-se o desenvolvimento.**
Resolvido	**Quando o caso é totalmente desenvolvido.**
Fechado	**Quando o caso está testado e encontra-se homologado.**

Para cada caso, existem as informações que detalham a mudança/desenvolvimento. O interessante do Mantis é que o gerente da ferramenta de gestão de defeitos pode manipular os campos/opções que detalham o caso, ou seja, o Mantis não se limita somente no que ele oferece, mas sim na capacidade que ele tem de tornar mais dinâmico o seu uso. Além disso, esses novos campos/opções podem ser utilizados para a extração de relatórios mais específicos, de acordo com a necessidade de quem à utiliza.

A Figura 7 destaca os campos necessários para relatar um caso no Mantis.

Figura 7 - Campos para registro de um caso no Mantis (AFL SISTEMAS, 2016)

Como destacado anteriormente sobre as características do Mantis, é possível retirar relatórios e estatísticas. O Mantis permite a análise das mudanças que foram acontecendo para os casos, além de ser possível filtrar por categoria, prioridade, status, entre outros. Além disso, é possível filtrar casos de acordo com os campos adicionados a mais pelo gerente da ferramenta.

A opção “Resumo” oferece este recurso. A Figura 8 retrata brevemente este recurso.

Resumo

Por Projeto	aberto	resolvido	fechado	total
AFL - Corporativo (Light)	123	54	1713	1890
» AFL Light - Monitor Web	0	0	4	4

Por Status	aberto	resolvido	fechado	total
novo	1	-	-	1
retorno	7	-	-	7
admitido	52	-	-	52
confirmado	3	-	-	3
atribuído	60	-	-	60
resolvido	-	54	-	54
fechado	-	-	1717	1717

Mais Ativos	Mudanças
0005554 - CADASTRO DE PRODUTOS - DESCRIÇÃO COMPLEMENTAR DO PRODUTO	38
0004356 - [PROPOSTA] DESENVOLVER NOVO RELATÓRIO DE PRODUÇÃO	37
0004605 - SGFA0201-Pedido venda /SGFA0301-Emissão NFs (permitir % desconto acima com senha)	35
0005592 - SPED0201/SPED0203 - GERAÇÃO DE ARQUIVO PADRÃO SPED	31
0004831 - PROCESSO DE TERCEIROS - Sincronizar processos com AFLNFeClient	29
0005855 - Desenvolvimento sistema central para feira e um programa para criar pedidos no coletor de dados	27
0004147 - SGFA0700-Vendas balcão-Consignação (gravando registro de entrada de estoque em empresa errada)	27
0005619 - [PROPOSTA] - Implementação módulo de estoque - controle de estoque por projeto / depósito	26
0003371 - PROCESSO DE EMISSÃO DE NFS DE SERVIÇOS - PREFEITURA DE LENÇÓIS PAULISTA	26
0004279 - ORDEM SERVICO - Ajustes ORSE0031-Árvore Produto / ORSE0032-Ordem Serviço-Projeto	25

Figura 8 – Resumo dos casos do Mantis (AFL SISTEMAS, 2016)

2.1.2 Bugzilla

A ferramenta Bugzilla, assim como o Mantis, é um sistema Web e tem suporte aos bancos de dados MySQL, PostegreSQL e Oracle, lembrando um pouco o concorrente. Porém, sua linguagem de programação é em PHP, utilizando a tecnologia de *scripts* Perl, em modo *Common Gateway Interface* (CGI).

As principais características do Bugzilla são: Sistema de busca rápida, parecido com o Google; Relatórios e gráficos; Detecção de casos duplicados; Notificações por e-mail; Anexos privados e comentários (anotações); Priorização dos casos; Distribuir/Repartir casos entre a equipe (desenvolvimento).

A Figura 9 expõe a página inicial do Bugzilla.

Figura 9 – Página inicial do Bugzilla (SOFTPEDIA)

Assim como no BugMantis, cada caso do Bugzilla consiste de um *status*.

A Figura 10 expõe quais são as possíveis situações de um caso no Bugzilla, envolvido em um fluxo, na qual caracteriza como o seu ciclo de vida.

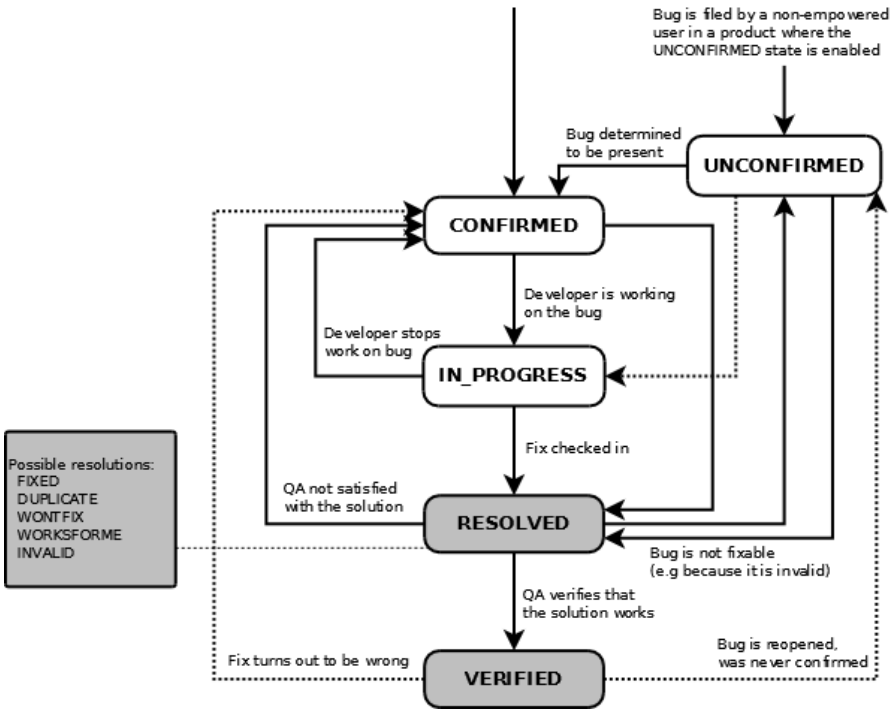


Figura 10 – Ciclo e situações de um caso no Bugzilla (BUGZILLA, 2015)

2.2 Ferramentas de gerenciamento de teste

As ferramentas de gerenciamento de teste auxiliam na gestão dos mesmos e estão presentes em todas as etapas do ciclo de vida do teste. Essas ferramentas ajudam em alguns pontos como na priorização dos testes, criação de cronogramas das atividades, criação de registros dos resultados, possibilidade de rastreamento do progresso e no gerenciamento dos incidentes dos testes.

Além disso, as ferramentas de gerenciamento de testes foram criadas para substituir o que, em alguns casos, é feito de forma manual, em papéis.

Essas ferramentas, utilizadas de forma correta, permitem uma tomada de decisão mais adequada, de acordo com os interesses do gestor.

A seguir será descrito sobre a ferramenta TestLink.

2.2.1 TestLink

O TestLink é uma ferramenta *Open Source*, desenvolvido em PHP, com suporte aos bancos de dados MySQL e PostgreSQL. Com esta ferramenta é possível a geração de registro e organização dos requisitos do projeto de testes, sendo possível garantir e verificar a sua rastreabilidade entre os casos de teste cadastrados e os requisitos a serem verificados.

Nesta ferramenta, é possível cadastrar diversos projetos de teste. Cada projeto tem as suas funcionalidades. Uma delas é a criação de planos de testes. Os planos de testes possuem os casos de testes, e estes possuem os passos para sua execução.

A Figura 11 exprime a hierarquia de um projeto no TestLink.

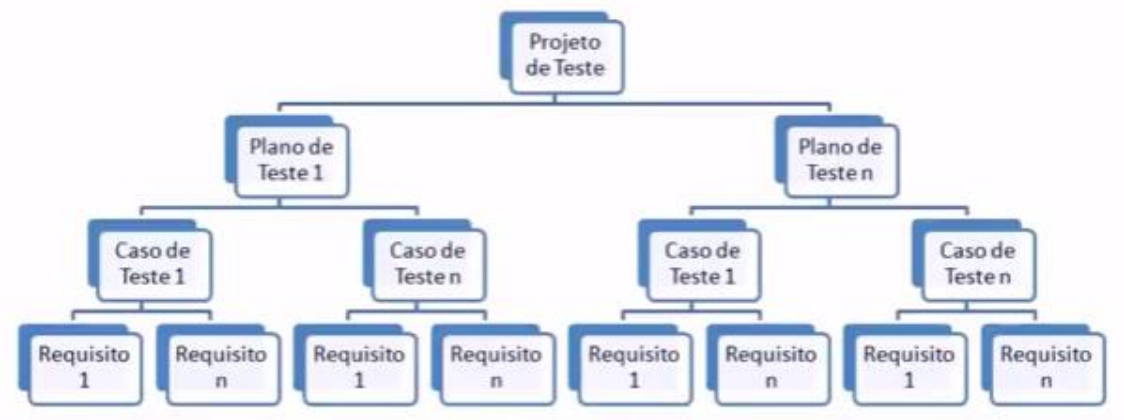


Figura 11 - Hierarquia de um projeto no TestLink (DE BITTENCOURT, 2015)

O uso correto da ferramenta facilita uma maior qualidade no processo de teste. Além disso, esta ferramenta, pode ser usada com ferramentas de gestão de erros (defeitos), como o Mantis, por exemplo, que oferece uma integração entre elas.

3 APLICAÇÃO DA METODOLOGIA DE TRABALHO

No decorrer deste capítulo, será descrito a empresa onde o estudo de caso se passou, demonstrando seu estado antes da aplicação da metodologia de teste. Também será descrito como e o que foi realizado nesta empresa para que houvesse um procedimento de teste padronizado.

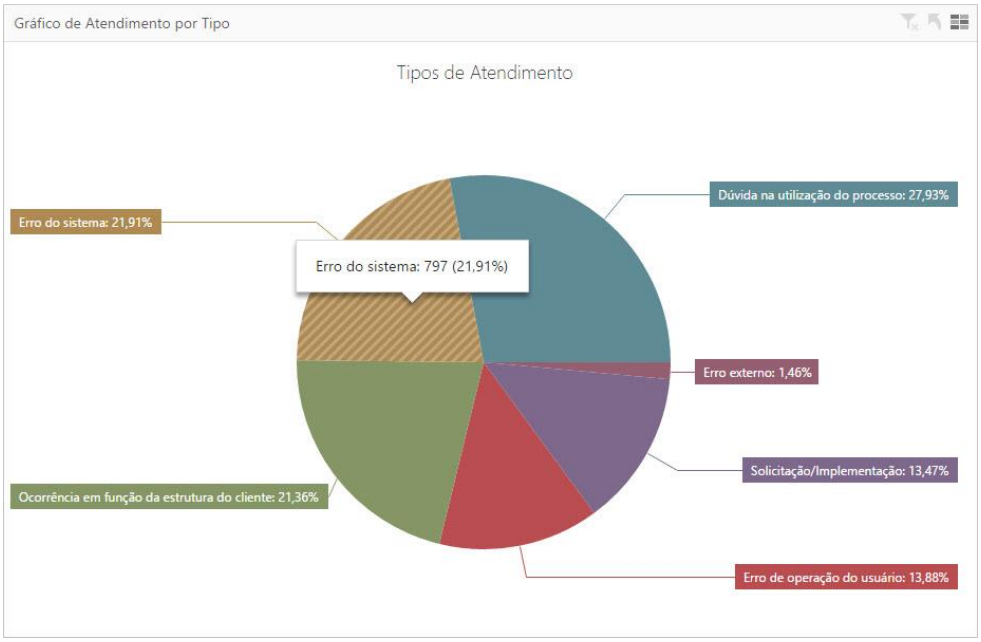
3.1 Empresa antes

A empresa onde se passou o estudo possui problemas com o produto que oferece. Estes problemas estão relacionados com os erros que este produto traz e isso gera muitos chamados ao suporte técnico e mão de obra de desenvolvimento para correção dos mesmos. Em muitos casos, os programadores precisam fazer hora extra para conseguir corrigir o problema e entregar a correção o quanto antes.

Erros de software geram muito custo para a empresa, tanto para o suporte, demandando muita gente para a função, quanto para a programação, devido ao alto custo com hora extra.

O Gráfico 1 expõe que em 2015 foram 797 chamados com situação de erro de sistema. Além disso existem outros tipos de atendimentos que foram registrados. Em relação a dúvidas na utilização do processo aplica-se quando o usuário está com dúvida quanto a realizar alguma operação no sistema. Erros externos estão relacionados com a infraestrutura de terceiros, como por exemplo a Secretaria da Fazenda (SEFAZ), quando a mesma está indisponível. Erros de operação do usuário são quando o cliente aciona o suporte técnico relatando um erro, onde o erro na verdade foi cometido pelo mesmo. Solicitação/Implementações está ligado diretamente com as solicitações de desenvolvimento que o cliente realiza e por fim, ocorrência em função da estrutura do cliente está relacionado quando algum problema ocorre na infraestrutura do cliente e o mesmo solicita o suporte para resolução. Se aplica quando há problemas com impressoras, redes, entre outros.

Gráfico 1 - Total de chamados por tipo em 2015 (Fonte própria)



A partir dos atendimentos e das situações de erro, são gerados casos para correção. Desses casos, ainda é possível visualizar casos abertos para correção.

A Tabela 6 exprime o que foi dito, na qual podemos considerar Código como o número do caso do BugMantis, Descrição como o título do caso, Status seria a situação atual do caso e o Tempo seria o período desde a sua abertura em dias. Neste caso, o tempo destacado de vermelho significa que o caso está a mais de 10 dias aberto, ou seja, não houve fechamento/resolução em menos de 10 dias.

Tabela 6 - Casos abertos em 2015 sem resolução (Fonte própria)

Descrição dos casos			
Código	Descrição	Status	Tempo
4.550	CAMPO DE DESCONTO NA IMPRESSÃO DE PEDIDO INIBINDO OS VALORES	Atribuído	661
4.616	SPED0201-Geração dados EFD (NF emitida de entrada gerada c/código erado)	Atribuído	646
4.860	SGFA0117-Controle de numeração da nota; ajustar layout 60 e 64 – IMEP/DAP	Atribuído	526
4.927	Melhoria e correções nos processos de terceiros	Atribuído	493
4.973	Impressão do código do cliente na DANFE	Atribuído	464
4.977	SPED0204-Manutenção valores aproximados dos produtos; ajustes na importação	Atribuído	460
5.337	Verificação ao habilitar uso do código do fabricante	Atribuído	323

A seguir serão descritos como eram realizados os procedimentos de teste e de

fechamento de versão na empresa na qual foi baseado o estudo de caso.

3.1.1 Equipe de teste

Nesta empresa não existe uma equipe alocada exclusivamente para a realização dos testes. A equipe de teste atualmente consiste nos analistas responsáveis por realizar o teste em sua solicitação de correção ou implementação, sendo estes analistas também responsáveis por suporte ao cliente.

3.1.2 Execução do teste

Após o desenvolvimento, o programador libera em um repositório a compilação necessária, junto com *scripts* de banco de dados, para a realização do teste. Sendo assim, na execução do teste o analista responsável pela abertura da solicitação faz a atualização de *scripts* em um banco de dados copiado, idêntico ao que é utilizado em produção, e atualiza os executáveis necessários para a realização dos testes.

Além disso atualmente os testes ocorrem de acordo com o erro ou desenvolvimento relatado, sem nenhum procedimento para seguir, ou seja, são testes *ad-hoc*.

Sendo assim, para os casos de erros, o analista faz a simulação do processo que ocasionou o erro. Se o problema não acontecer, é simulado outros processos que cheguem ao mesmo resultado final, ou seja, testa-se os demais programas que fazem o mesmo processo envolvido na correção.

Esse processo de teste não é acompanhado por nenhum *checklist* ou algo parecido. O analista irá simplesmente fazer os testes nos processos de acordo com o conhecimento que o mesmo possui.

Quando se trata de um erro que envolve processos importantes e/ou complexos, como emissão de nota fiscal eletrônica, por exemplo, outros analistas são envolvidos para que haja mais de uma “metodologia” (ou conhecimento) de teste a ser aplicada. Sendo assim, são efetuados os testes no processo relatado e também testes nos demais processos ou programas envolvidos. Existem processos específicos para clientes que nem todos os analistas tem conhecimento, e dessa forma, torna-se necessário o envolvimento dos demais.

Para os casos de desenvolvimento, dependendo do módulo ou processo, é realizado uma troca de e-mails ou reuniões para definição e melhor planejamento para abertura do caso de desenvolvimento. Isso para que os demais analistas possam compartilhar

conhecimento ou sugestões afim de evitar desenvolvimentos específicos ou desnecessários.

Quando acontece o retorno da programação sobre a implementação, o analista responsável realiza a atualização em seu ambiente de testes, *scripts* e executáveis envolvidos no processo e posteriormente realiza novamente os testes.

3.1.3 Fechamento de versão

Já em relação ao processo de liberação de versão ou fechamento de versão, não existe um padrão ou até mesmo processo nesta empresa. Todas as solicitações de implementação que são homologados, após serem testados, vão para um repositório onde encontram-se todos os executáveis e *scripts* de banco de dados consideráveis estáveis para uma possível atualização no cliente.

3.1.4 Considerações

Vale a pena ressaltar alguns pontos mediante ao processo antigo.

- Não há um roteiro com os programas e processos necessários para teste.
- Não há uma ferramenta que possa ser usada para que qualquer analista consiga realizar os testes e validações nos módulos com correções ou desenvolvimentos.
- Há uma dependência e vínculo entre o caso de desenvolvimento e o analista, bem como uma responsabilidade após a liberação para homologação.
- Não há um resumo analítico ou sintético das correções ou desenvolvimentos feitos na versão.
- Não há um controle de fechamento de versão e dos desenvolvimentos/correções efetuados. Dessa forma faz-se necessário que o analista de suporte realize atualizações parciais na base de dados do cliente, liberando executáveis aleatórios com base na compilação.

3.2 Empresa depois

Para que seja possível uma melhora na empresa em pauta, em relação ao seu produto e de uma redução de custo, a seguir será descrito o que foi realizado na mesma, para

que houvesse uma padronização de teste e de fechamento de versão.

3.2.1 Equipe de teste

No tocante a equipe de teste, conforme mencionado no item 3.1.1, a equipe de analistas de suporte era responsável por suporte e testes. Deste modo, foi criado um setor dentro da empresa afim de que os testes fiquem centralizados para uma equipe capacitada, com o intuito de realizar uma execução correta dos testes.

Esta equipe é composta por três pessoas, que foram treinadas para que o teste seja executado de forma correta, com o uso da ferramenta TestLink e com o auxílio do Mantis.

Esta equipe realizará testes por solicitações efetuadas, que já foram implementadas e também realizarão testes em cada fechamento de versão. Em todos os casos, seja por solicitação ou no fechamento da versão, será utilizado o Mantis, para o registro das mudanças e o TestLink para o gerenciamento do teste.

3.2.2 Fechamento de versão

Em relação ao fechamento de versão, conforme relatado no item 3.1.3, o mesmo também não existe; somente há um repositório que concentra todos os executáveis consideráveis como estáveis. Sendo assim, em comum acordo, foi estabelecido que haverá todos os meses ao menos um fechamento de versão. Este fechamento de versão será composto por todas as solicitações registradas no Mantis, que foram desenvolvidas e testadas.

Em relação ao fluxo, todos os pacotes que foram implementados são reunidos e posteriormente o programador faz a compilação de todos os módulos do produto, gerando assim os executáveis. Um pacote consiste em um arquivo que contém o código fonte da alteração e, se houve, os *scripts* de banco de dados.

Após esta compilação, o programador libera em um repositório todos esses executáveis e *scripts* para que a equipe realize os testes. Em caso de erro ou defeito encontrado, o responsável pelo teste irá abrir um caso no Mantis para que haja a correção do problema.

A Figura 12 destaca o *Business Process Modeling Notation* (BPMN) que descreve o processo de fechamento de versão.

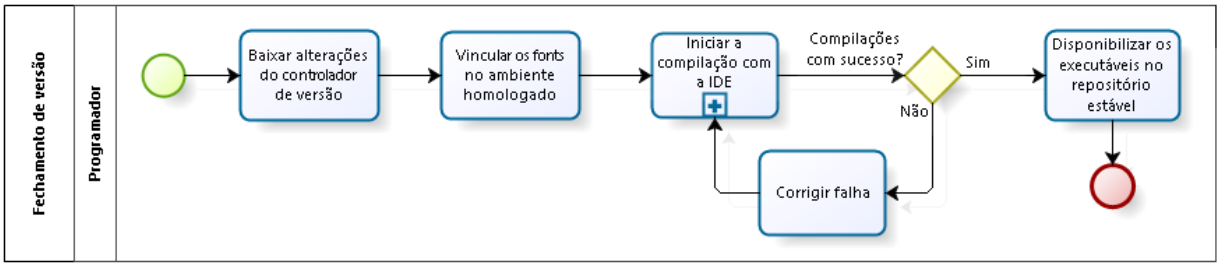


Figura 12 – BPMN do processo de fechamento de versão (Fonte própria)

3.2.3 Padronização do Mantis

Em relação ao Mantis, para que haja um processo adequado para o seu uso, foi criado um fluxo por meio do BPMN. O fluxo consiste desde a abertura até o fechamento de um caso.

Um caso pode ter o seu status como novo, atribuído, admitido, confirmado, retorno e fechado, conforme Tabela 5, que destaca o status e sua descrição.

A Figura 13 retrata o BPMN que descreve o fluxo de um caso Mantis, na qual observa-se que o fluxo se inicia e se encerra no analista, na qual o mesmo relata o caso, anexando os arquivos e documentos da análise e fecha-o quando testado e homologado. Depois de aberto o caso, o analista é responsável por atribuir o caso para o gerente de desenvolvimento, que posteriormente fará o escalonamento, admitindo o caso para o programador.

O programador recebe o caso e irá realizar o desenvolvimento de acordo com o seu cronograma. Após a confirmação do desenvolvimento, o programador faz a liberação da alteração e realiza alteração do status do caso no BugMantis e manda um e-mail para o analista.

O analista recebe o caso, onde realiza os testes de acordo com a padronização adotada. Caso haja erros encontrados, o caso é retornado ao programador, mudando o status no BugMantis e enviando o e-mail. Em seguida, o programador realiza as alterações necessárias e reenvia o caso para o analista, fazendo o mesmo processo anterior.

Quando o caso está aprovado, ou seja, quando os testes estão de acordo com o resultado esperado, o analista altera o status do caso para “resolvido” e realiza o envio do e-mail para o programador e demais envolvidos.

Nesta situação de homologação, o programador faz a compilação necessária e libera no repositório de executáveis homologados e confirma o caso para o analista realizar os testes no ambiente homologado.

Depois do caso homologado, o analista realiza os testes de acordo com a padronização adotada, conforme mencionado anteriormente. Caso haja algum problema com a versão homologada, é retornado ao programador, mudando o status do caso novamente para retorno, onde o programador fará as devidas correções e confirma novamente o caso para o analista.

Em caso de aprovação no ambiente homologado, ou seja, caso testado nos dois ambientes (*Quality Assurance* (QA) e homologado), o analista fecha o caso, concluindo-se assim o seu ciclo.

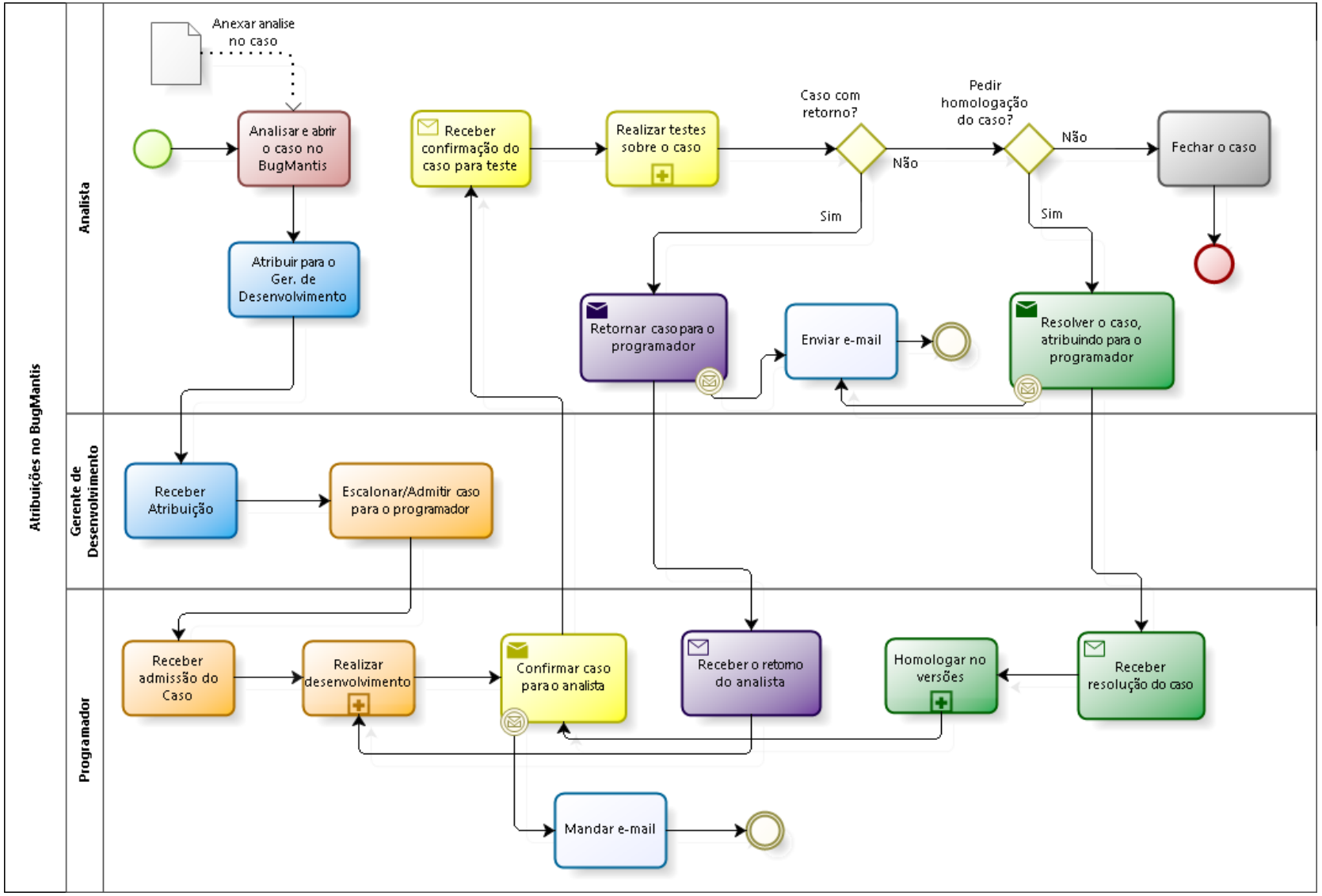


Figura 13 - Ciclo de vida de um caso no BugMantis (Fonte própria)

Essa padronização do fluxo/ciclo de vida auxilia no acompanhamento de um caso, sendo possível saber quantos retornos houve e o tempo desde sua atribuição até seu fechamento (período do ciclo de vida), deste modo, é possível ter uma gerência correta dos casos com um fluxo correto, como será destacado no item 3.2.4.

3.2.4 *Dashboard* de acompanhamento do BugMantis

Para complementar este projeto, foi criado um *dashboard* que demonstra algumas informações relevantes do BugMantis, sendo que o mesmo até apresenta alguns relatórios, porém, tudo de forma separada e estática, diferente do *dashboard* criado.

Criado utilizando o recurso "*Dashboard Design*" do *framework* da empresa Developer Express Inc., este *dashboard* apresenta informações como período, projetos cadastrados no BugMantis, relatório em gráfico com relação a Categoria, Status, Prioridade, Severidade e tabelas que demonstram os casos do Mantis e os clientes relacionados com cada caso.

Conforme mencionado, os relatórios do BugMantis são estáticos. Com o *dashboard* filtros são mais fáceis de serem aplicados. Em relação ao período dos casos, a Figura 14 estampa o *dashboard* filtrando somente no período de 2015.

Figura 14 – Filtro aplicado no *dashboard* (Fonte própria)

Com o filtro aplicado, todo o *dashboard* em geral é atualizado, ou seja, as informações das outras grades são atualizadas com as informações do período informado. Isso não se aplica somente para a grade do período. Outras grades, quando são filtradas, também atualizam todo o *dashboard*.

A Tabela 7 expõe o resultado do projeto após a aplicação do filtro no período. Tabela que contém a descrição do projeto e total de casos de acordo os filtros aplicados. Também é possível observar que existem 178 casos registrados no BugMantis para o projeto "AFL – Corporativo (LIGHT)".

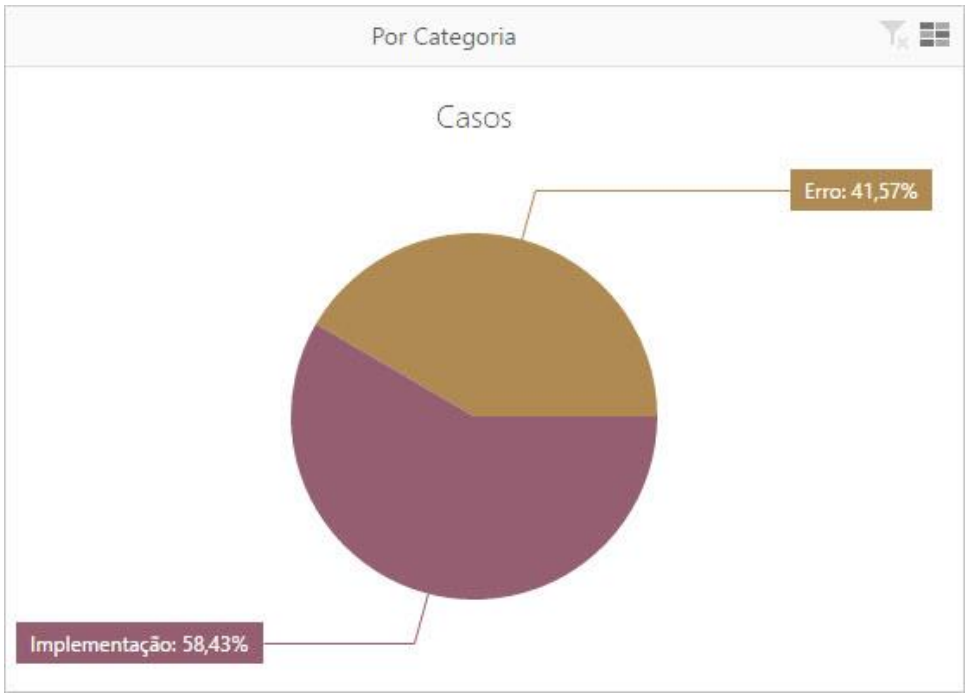
Tabela 7 - Projetos e quantidade de casos no *dashboard* (Fonte própria)

Projetos	
Projeto	Qtde casos
AFL – Corporativo (Light)	178
Brudden	66
AFL Implantações	50
AFL – Empresarial	39
AFL – Corporativo Plus	39
Fundação	21
Shalom – Corporativo	18
MMI Brasil	14
GEGÊ	9
Amendupã	1

O Gráfico 2 mostra, de acordo com o período de 2015 e o projeto mencionado anteriormente, os casos de acordo com sua categoria.

Observa-se no Gráfico 2 as categorias Erro e Implementação, porém, podem aparecer diversas categorias, como por exemplo Sugestão e Dúvidas. A categoria Erro está relacionado com casos de erros do projeto. A categoria Implementação agrupa os casos de implementação, que vieram a partir de solicitações ou de necessidades. A categoria Sugestão agrupam os casos que se caracterizam como sugestão de desenvolvimento e Dúvida agrupa os casos onde existem dúvidas, que ainda serão tratadas e filtradas.

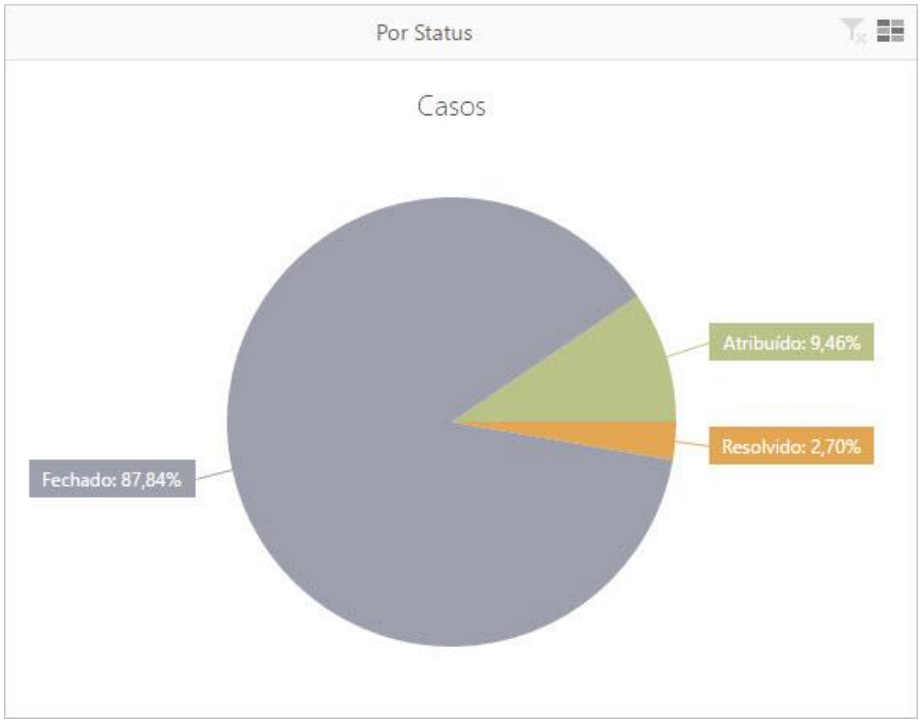
Gráfico 2 - Casos agrupados por categoria no *dashboard* (Fonte própria)



Ao filtrar pela categoria “Erro”, o que corresponde a 74 casos, o Gráfico 3 expõe os casos agrupados por status, sendo filtrando também de acordo com o período e projeto, como mencionado anteriormente.

Observa-se no Gráfico 3 os status Fechado, Atribuído e Resolvido. Porém, diversos status podem aparecer, assim como mencionado na Tabela 5 deste documento. Além disso é possível analisar que existe um percentual de casos ainda com situação de “Atribuído”. Isso significa, analisando o fluxo destacado na Figura 13, que este gráfico mostra que existem casos abertos, sem correção ou resolução.

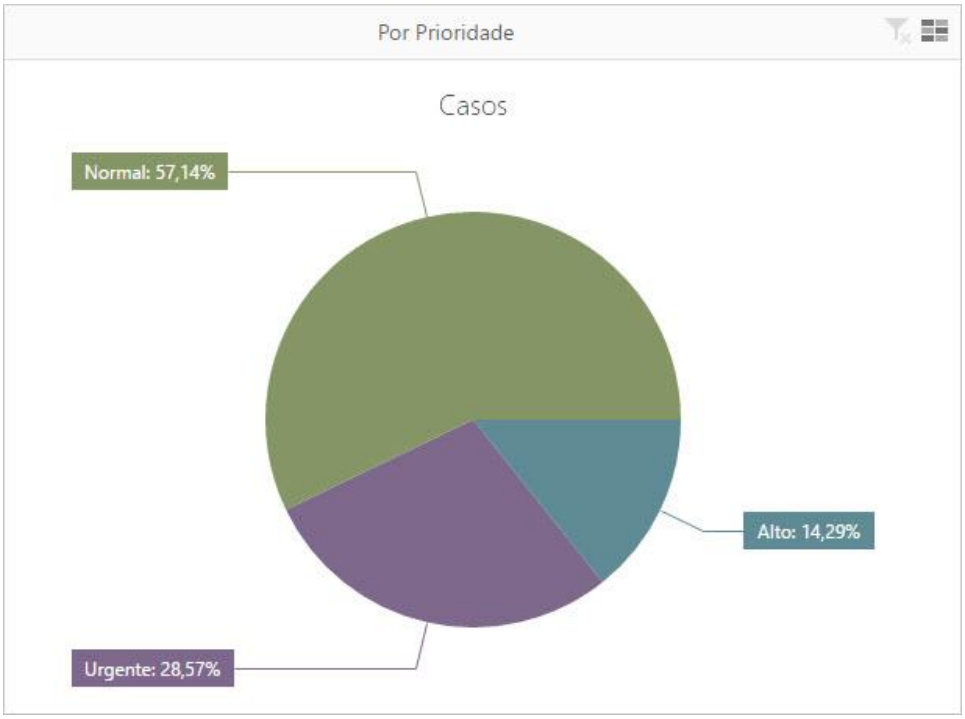
Gráfico 3 - Casos agrupados por Status no *dashboard* (Fonte própria)



O Gráfico 4 mostra os casos agrupados por prioridade. Neste caso, o filtro aplicado no *dashboard* está com o Status “Atribuído”, Categoria “Erro”, Projeto “AFL – Corporativo (*LIGHT*)”, no período de 2015, correspondendo a um total de sete casos.

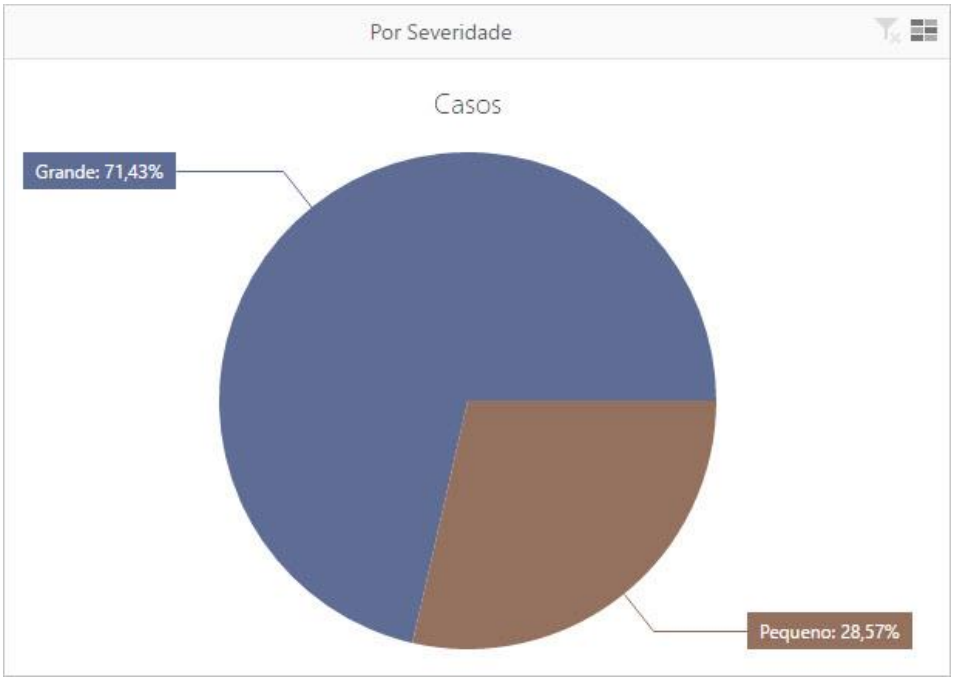
Observa-se no Gráfico 4 que dos casos ainda com situação de “Atribuído”, 14,29%, que corresponde à um caso, tem prioridade alta e 28,57% (dois) dos casos estão com prioridade “Urgente”. Os outros 57,14% (quatro) estão relacionados com casos neutros, de prioridade normal.

Gráfico 4 - Casos agrupados por prioridade no *dashboard* (Fonte própria)



O Gráfico 5 exprime a severidade destes sete casos, sendo que 28,57% (dois) dos casos com prioridade alta ou urgente estão com severidade pequena e o restante, correspondendo a cinco casos estão com severidade grande.

Gráfico 5 - Casos agrupados por severidade no *dashboard* (Fonte própria)



Observa-se na Tabela 8 os sete casos de acordo com o filtro aplicado no *dashboard*, com o seu código, descrição, situação, que são os atribuídos e o seu período, em dias, em aberto.

Tabela 8 - Tabela de casos no *dashboard* (Fonte própria)

Descrição dos casos			
Código	Descrição	Status	Tempo
4.550	CAMPO DE DESCONTO NA IMPRESSÃO DE PEDIDO INIBINDO OS VALORES	Atribuído	661
4.616	SPED0201-Geração dados EFD (NF emitida de entrada gerada c/código erado)	Atribuído	646
4.860	SGFA0117-Controle de numeração da nota; ajustar layout 60 e 64 – IMEP/DAP	Atribuído	526
4.927	Melhoria e correções nos processos de terceiros	Atribuído	493
4.973	Impressão do código do cliente na DANFE	Atribuído	464
4.977	SPED0204-Manutenção valores aproximados dos produtos; ajustes na importação	Atribuído	460
5.337	Verificação ao habilitar uso do código do fabricante	Atribuído	323

A Tabela 9 estampa a quantidade de casos de acordo com o cliente que estão filtrados no *dashboard*. Uma atenção especial ao cliente "AFL", na qual os casos com este cliente estão relacionados somente a erro e afetam todos os clientes ou ocorreram em mais de um cliente, antes de ser registrado no BugMantis.

Tabela 9 - Tabela com o cliente e a quantidade de casos relacionados (Fonte própria)

Clientes com mais casos	
Cliente	Qtde de casos
AFL	2
RM Placas	1
PALUFER	1
IMEP	1
CABOFLEX	1
AFL-LIGHT	1

Além disso, há um outro *dashboard* que apresenta os casos em que houveram ao menos um retorno. A sua estrutura é a mesma do *dashboard* relatado anteriormente, porém, a diferença está na tabela de casos, na qual é acrescentada a coluna "Retorno", que destaca a quantidade de retornos que foram registradas no caso.

A Tabela 10 expõe os casos que houveram ao menos um retorno, no período de 2015, do projeto "AFL – Corporativo (*LIGHT*)", relacionados na categoria "Erro". Além disso, é possível observar que alguns desses casos tiveram um tempo alto até o seu

fechamento. Neste caso, sempre que há retorno, significa que o caso já passou pelo processo de desenvolvimento e está em fase de teste, onde o responsável pelo teste faz o retorno, registrando no BugMantis.

Tabela 10 - Casos com ao menos um retorno (Fonte própria)

Descrição dos casos				
Código	Descrição	Status	Tempo	Retorno(s)
5.066	SGFA0301-Digitação de nota fiscal; erro emissão de NF de entrada	Fechado	117	5
4.888	Problema ao cancelar NF de pedido – Picking	Fechado	70	4
4.891	Erros no programa SGVX – XML de entrada	Fechado	70	3
5.114	Processo de validação automática – SGVX	Fechado	399	3
5.229	AFLCFe – Cupom fiscal eletrônico não gravando CPF digitado no fechamento	Fechado	27	3
5.320	SPRVPARA-Parâmetro do pré-venda ajustar impessão personalizada	Fechado	302	3
4.568	ANTECIPAÇÃO RECEBIMENTO SEM VINCULO NO PEDIDO DE VENDA	Fechado	56	2
4.722	Correções de faturamento para o AFL Picking	Fechado	595	2
4.786	Erro ao faturar pedido com CFOP não parametrizada no SGFAPARA	Fechado	559	2
5.261	Etiqueta de produo com código de barras	Fechado	16	2
4.732	SGFI0367/SGFI0356-Duplicata: ajustar regra que executada para impressão	Fechado	68	1
4.916	SGFA0201 – Pedidos – Problema com pedido com bx parcial e desconto	Fechado	32	1

Como a ideia deste projeto é ajudar a comunidade, demonstrando tudo o que foi realizado, foi criado uma página na *Web* com o *dashboard* pronto, na qual o usuário irá colocar as informações do banco de dados do BugMantis e o *dashboard* irá mostrar as informações. O endereço da página é: http://web.aflsistemas.com.br/dashboard-mantis/.

A Figura 15 transparece como o usuário pode preencher as informações do banco de dados.

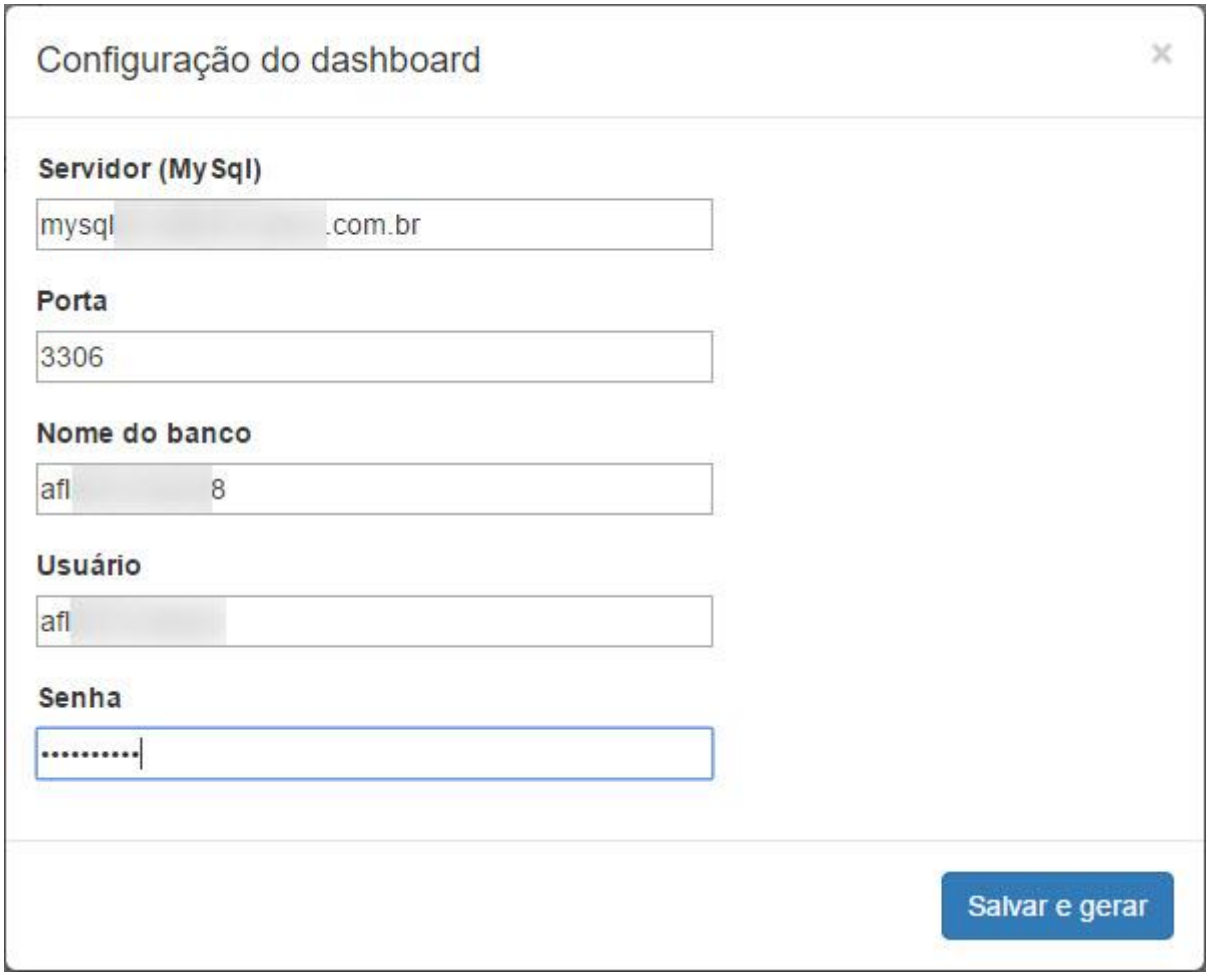


Figura 15 - Configuração do *dashboard* (Fonte própria)

Além disso, está disponibilizado no GitHub o código fonte utilizado, assim como o arquivo *Extensible Markup Language* (XML) dos *dashboard*, para que a comunidade possa melhorá-los. O link do GitHub é: https://github.com/dbsbotter/Dashboard-Mantis.

O “APÊNDICE A – SELECTS UTILIZADAS NOS DASHBOARDS” destaca as *selects* que são utilizadas nos *dashboards*, na qual é possível analisar com mais detalhes.

3.2.5 Padronização do Teste

Como dito no item 3.1.2, o procedimento de teste é realizado de acordo com o conhecimento do analista e isso pode acarretar em um teste inapropriado ou incompleto.

Para que isso não ocorra, o TestLink será utilizado para os registros dos casos de testes e para a gerência dos testes. Cada caso de teste consiste em um processo do sistema, como por exemplo, uma emissão de Nota Fiscal Eletrônica. Neste caso de teste pode haver *n* passos para serem seguidos, ou seja, são os passos que serão executados durante o processo de

teste. Além disso, a gerência de teste servirá para cooperar com as atribuições de casos de testes para os analistas, além de auxiliar no processo de execução do teste e com a exibição de relatórios analíticos.

Sendo assim, no TestLink, foi necessário criar um Projeto de Teste, cujo o nome se dá pelo nome do produto. A Figura 16 estampa o projeto de teste no TestLink.

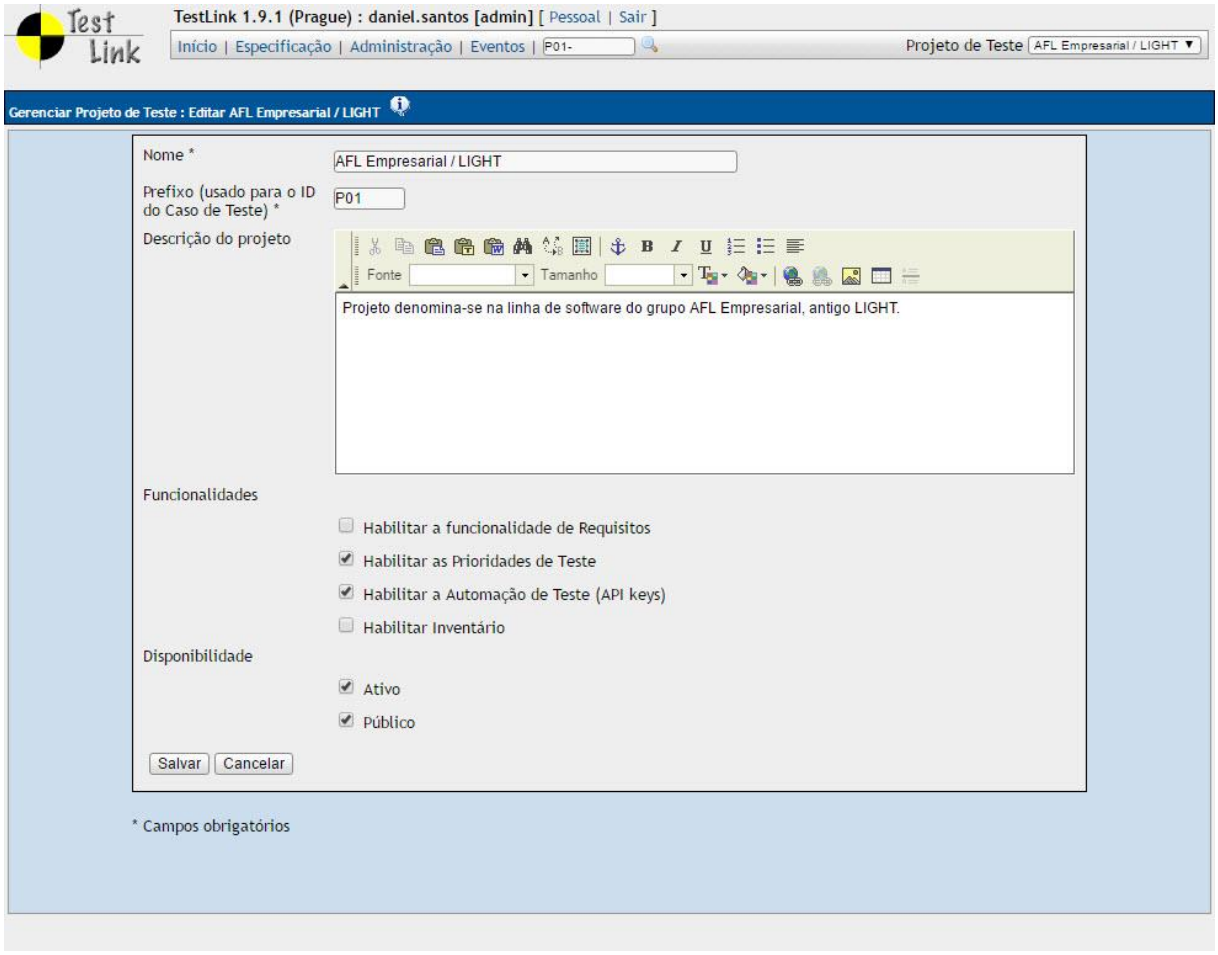


Figura 16 - Projeto de teste no TestLink (Fonte própria)

Com o projeto de teste criado, suítes de testes e casos de testes podem ser registrados. Os suítes de teste são pastas escalonáveis, que formam uma estrutura para separação dos casos de testes. Neste caso, os suítes de testes serviram para separar os casos de testes de acordo com os módulos do produto. Dentro de cada suíte de cada módulo do sistema, foram criadas outras suítes para a separação de processos que envolvem mais de um caso de teste.

A Figura 17 refere-se a árvore que foi formada com os suítes de testes e os casos de testes de cada módulo, sendo exibido como exemplo, o módulo/suíte denominado “Comercial”, que centraliza processos como pedidos de venda, faturamento de notas fiscais,

entre outros.

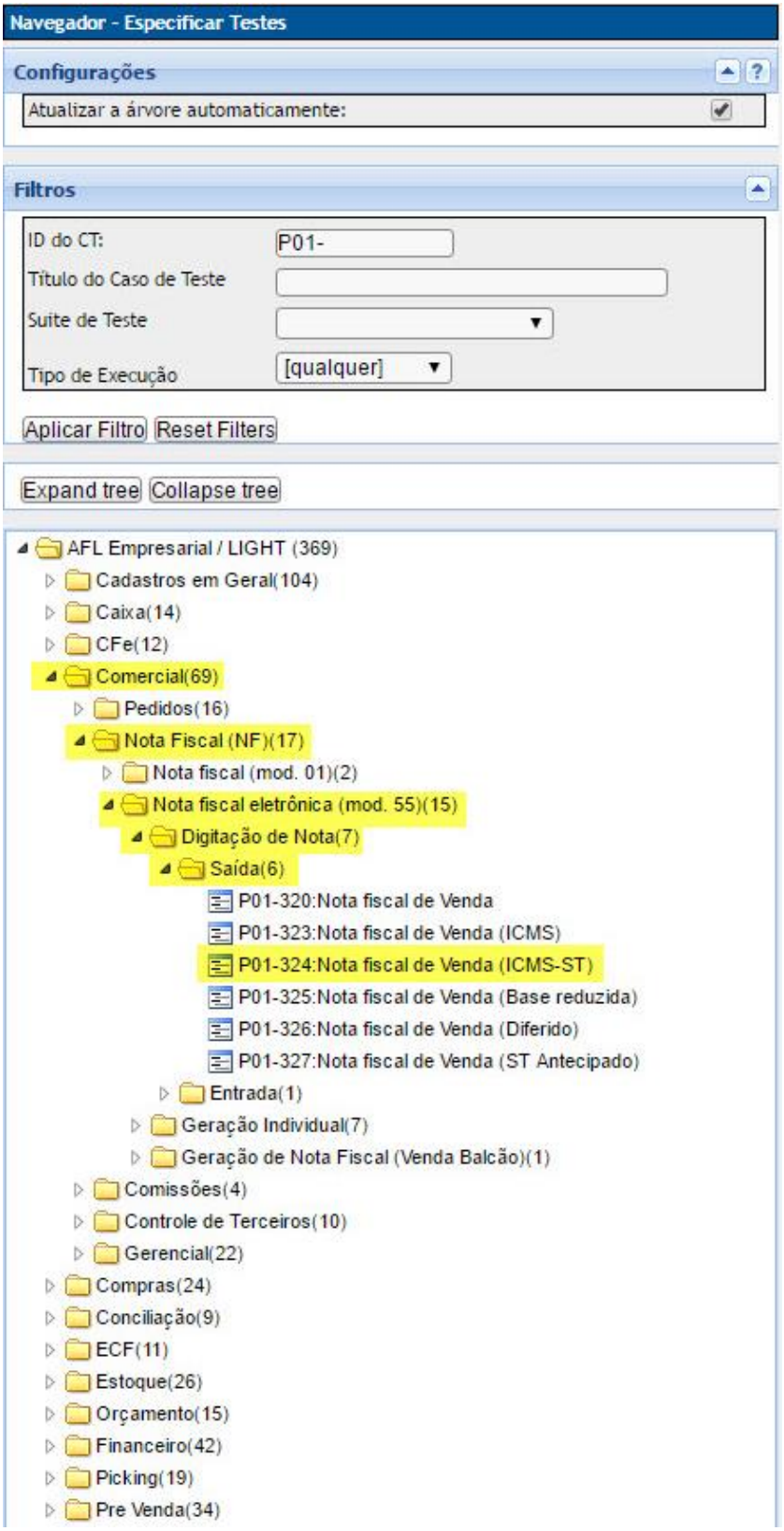


Figura 17 - Árvore do TestLink, com Suítes e Casos de Teste (Fonte própria)

Um Caso de Teste especifica um cenário de testes particular (um processo do sistema), que é descrita por um título, sumário, pré-condições, prioridade e tipo de execução. Além disso, um caso de teste é definido com as ações de cada passo e o seu resultado esperado. A Figura 18 e Figura 19 retratam como são especificados um caso de teste.

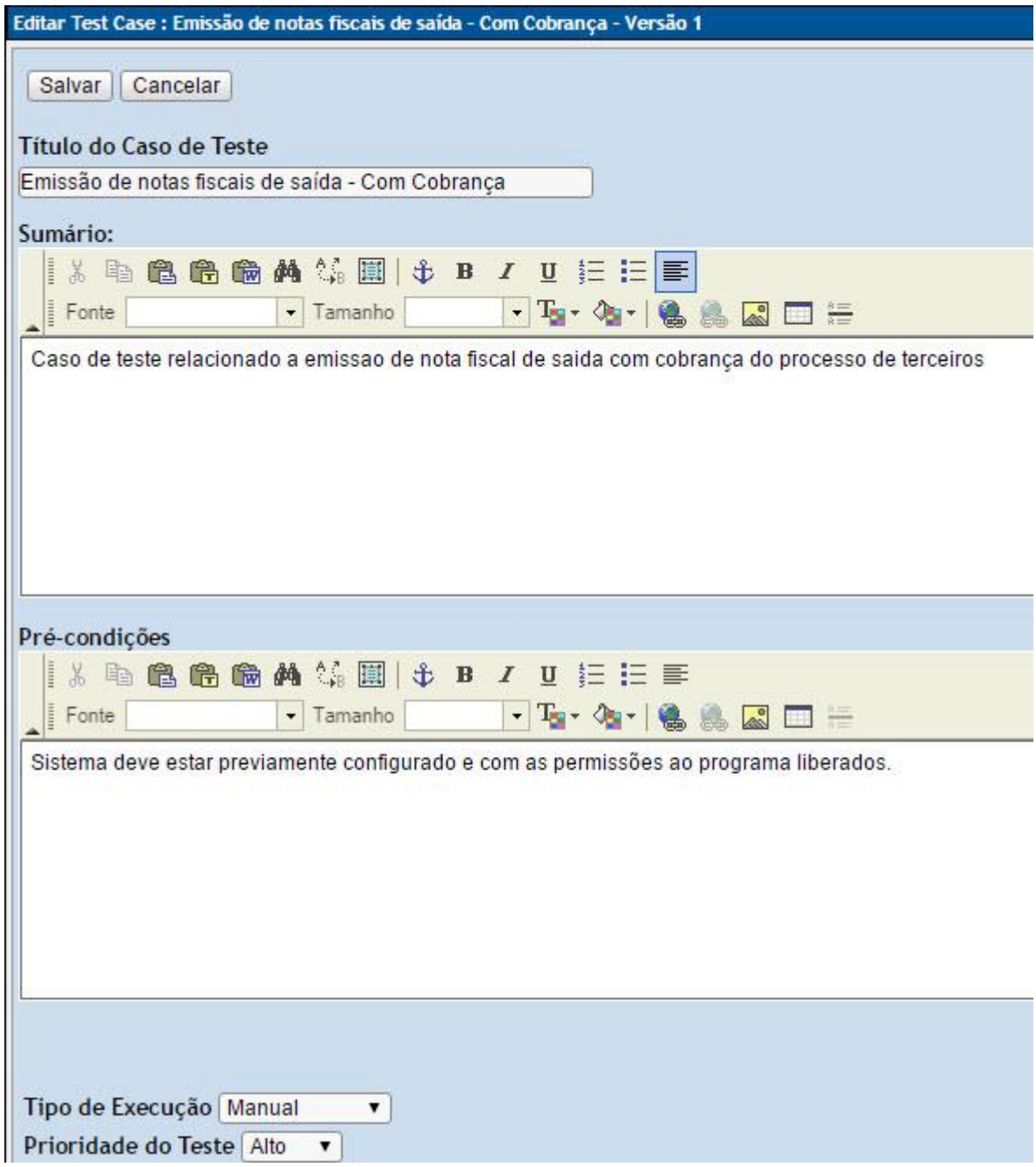


Figura 18 - Criando um caso de teste no TestLink (Fonte própria)

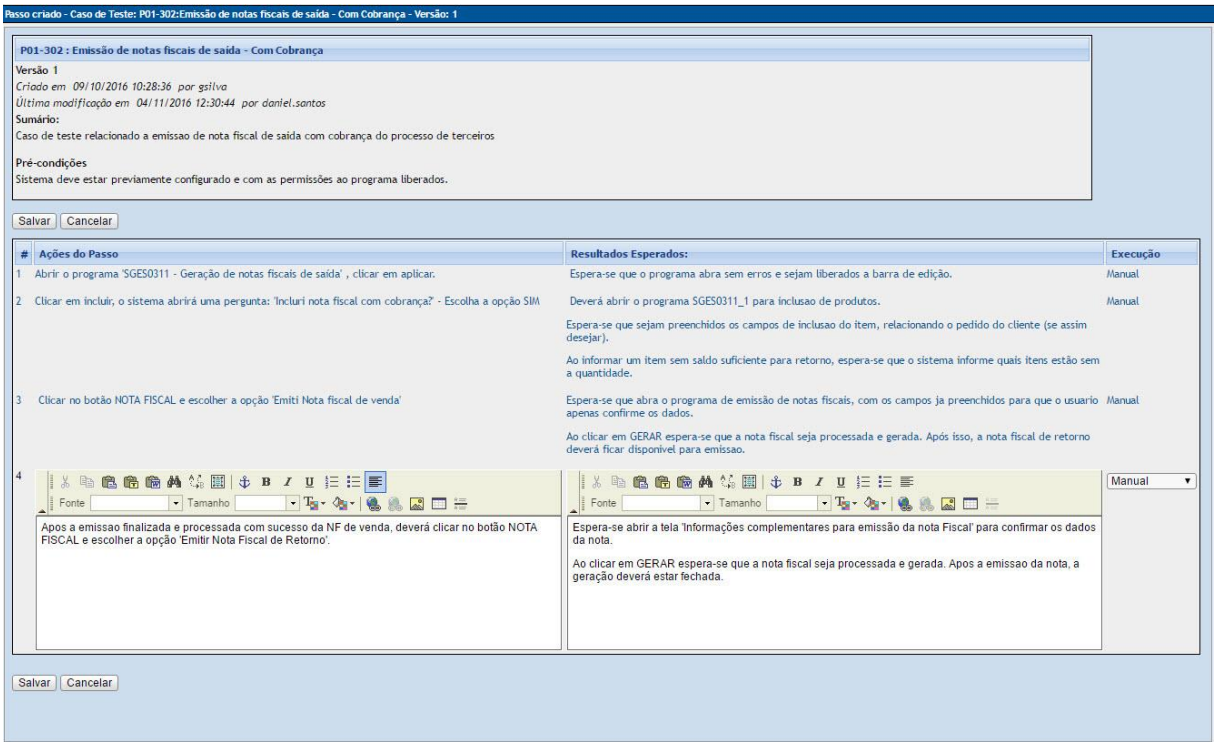


Figura 19 - Especificando passo de um Caso de Teste no TestLink (Fonte própria)

Depois de ter especificado os passos, o resultado final de um caso de teste é similar ao da Figura 20.

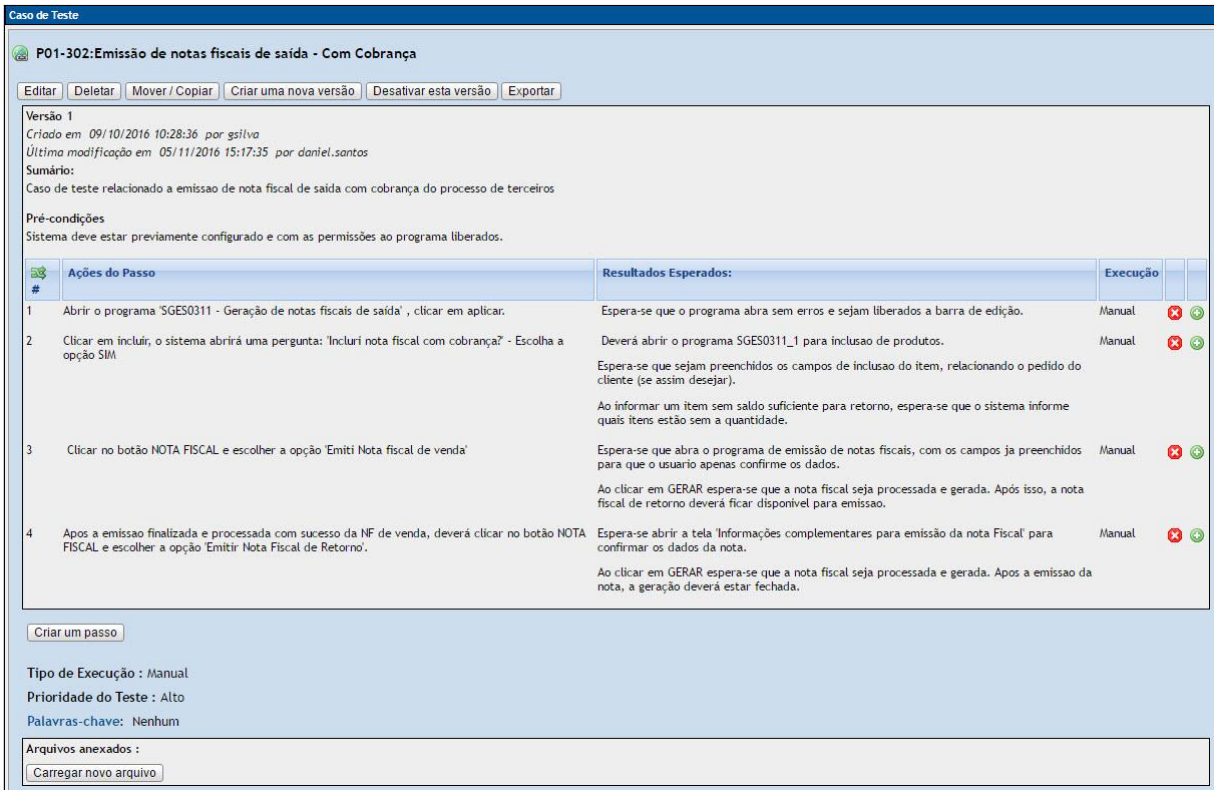


Figura 20 - Caso de Teste especificado no TestLink (Fonte própria)

Com os casos de testes especificados, o próximo passo é a criação do plano de teste. O Plano de Teste é definido por um título e descrição. Cada Plano de Teste deve conter as atividades de teste para o fechamento de uma nova versão do produto ou também conter as atividades de um básico e simples caso, quando desenvolvido e enviado para testes.

A Figura 21 expõe como é definido um Plano de Teste.

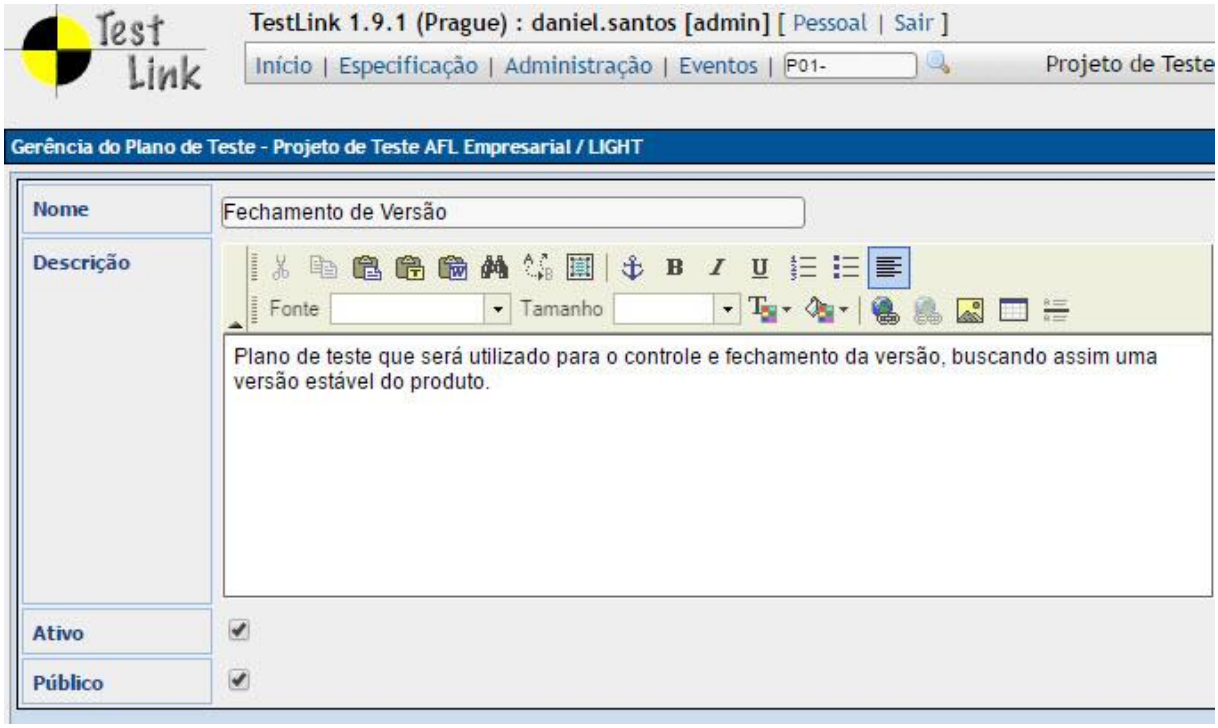


Figura 21 - Plano de teste no TestLink (Fonte própria)

Depois de definido um Plano de Teste, os Casos de Testes devem ser adicionados a ele, para que possam ser testados pelos analistas.

A Figura 22 retrata o Caso de Teste “Emissão de notas fiscais de saída – Com Cobrança”, do Suíte de Teste “Processo de Terceiros”, sendo adicionado ao plano de teste “Fechamento de Versão”.

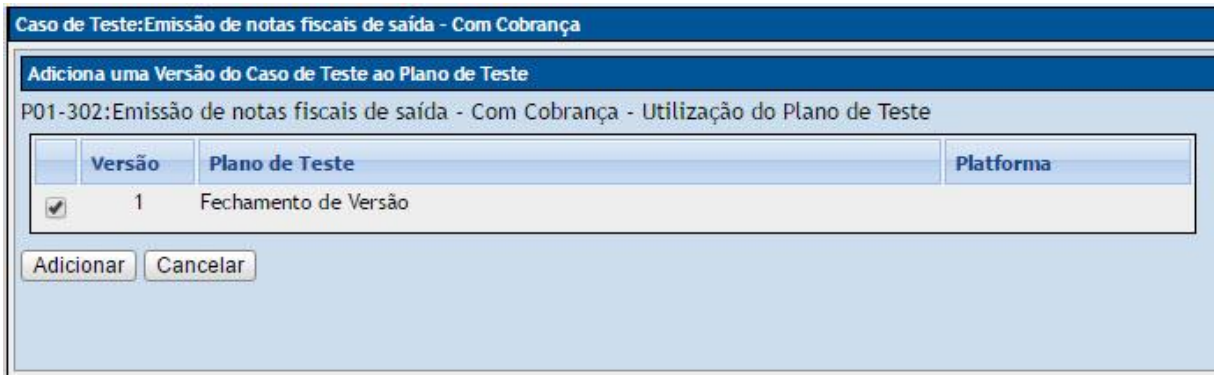


Figura 22 - Atribuindo um Caso de Teste ao Plano de Teste no TestLink (Fonte própria)

Existe também outra forma de adicionar os Casos de Teste ao Plano de Teste. A Figura 22 retratou a forma de adicionar um caso de teste específico ao plano de teste. Agora a Figura 23 expôs como adicionar vários casos de testes de um suíte, a um plano de teste e destaca também que é possível escolher a versão do caso de teste e a ordem de execução do suíte.

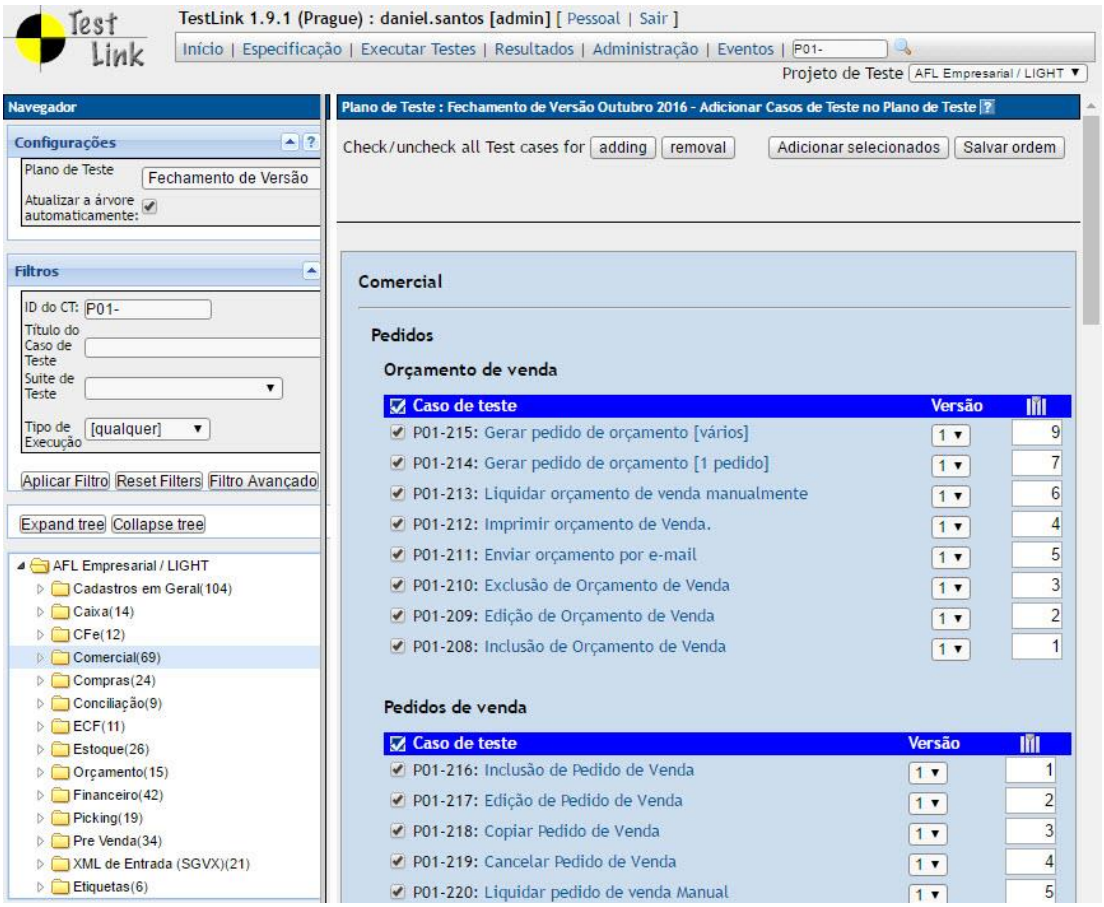


Figura 23 - Adicionando vários Casos de Teste ao Plano de Teste no TestLink (Fonte própria)

Depois disso, é momento de criar a *Baselines* ou *Releases*, que, de acordo com o conceito da ferramenta, são denominadas como controladores de versão do Plano de Teste. Em algumas literaturas, *baseline* é o conjunto de artefatos que compõem um release. Nesta situação, o Plano de Teste chama-se "Fechamento de Versão" e dentro deste plano de teste, pode haver diversos fechamentos que foram ocorrendo com o tempo.

A Figura 24 exprime a criação da primeira *Baseline* do plano de teste "Fechamento de Versão", que é definida por título, descrição e data, sendo definida também por dois atributos: ativo e aberto.

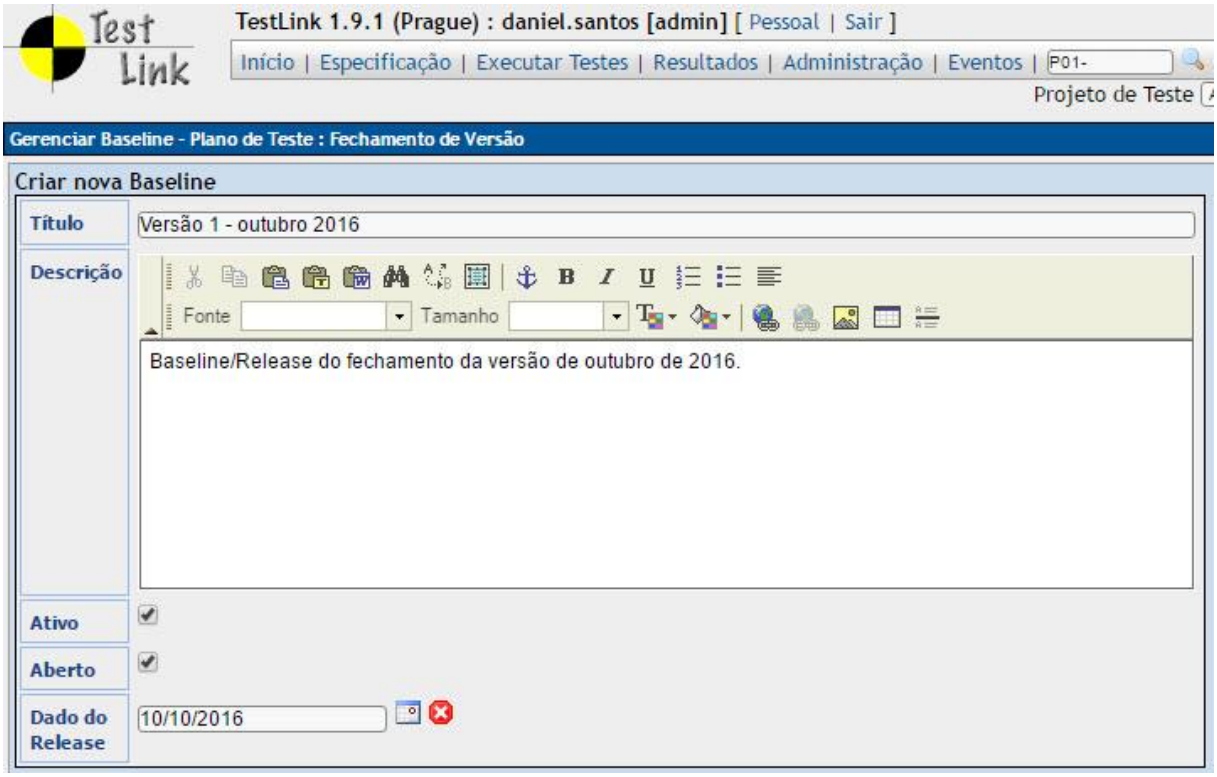


Figura 24 - Criação da *Baseline* no TestLink (Fonte própria)

Com a definição dos Casos de Testes, Plano de Teste e da *Baseline*, deve-se atribuir os casos de testes para os analistas. Isso pode ser feito de duas formas. A primeira seria atribuir o testador a cada caso de teste, ou seja, um por vez. A segunda seria atribuir em forma de lote, selecionando vários casos de teste ou suítes de testes e atribuir a um testador.

A Figura 25 estampa como é feita a atribuição do testador a um caso de teste, em que é escolhido o caso de teste, por meio da interação com árvore de casos e suítes, sendo selecionado o testador/usuário no *combo box*.

Figura 25 - Atribuindo um caso de teste ao testador no TestLink (Fonte própria)

A Figura 26 expõe como é realizado a atribuição em lote, em que é selecionado um suíte de teste, pela interação com a árvore, que depois são selecionados os casos de testes e atribuído ao testador, efetuando em lote.

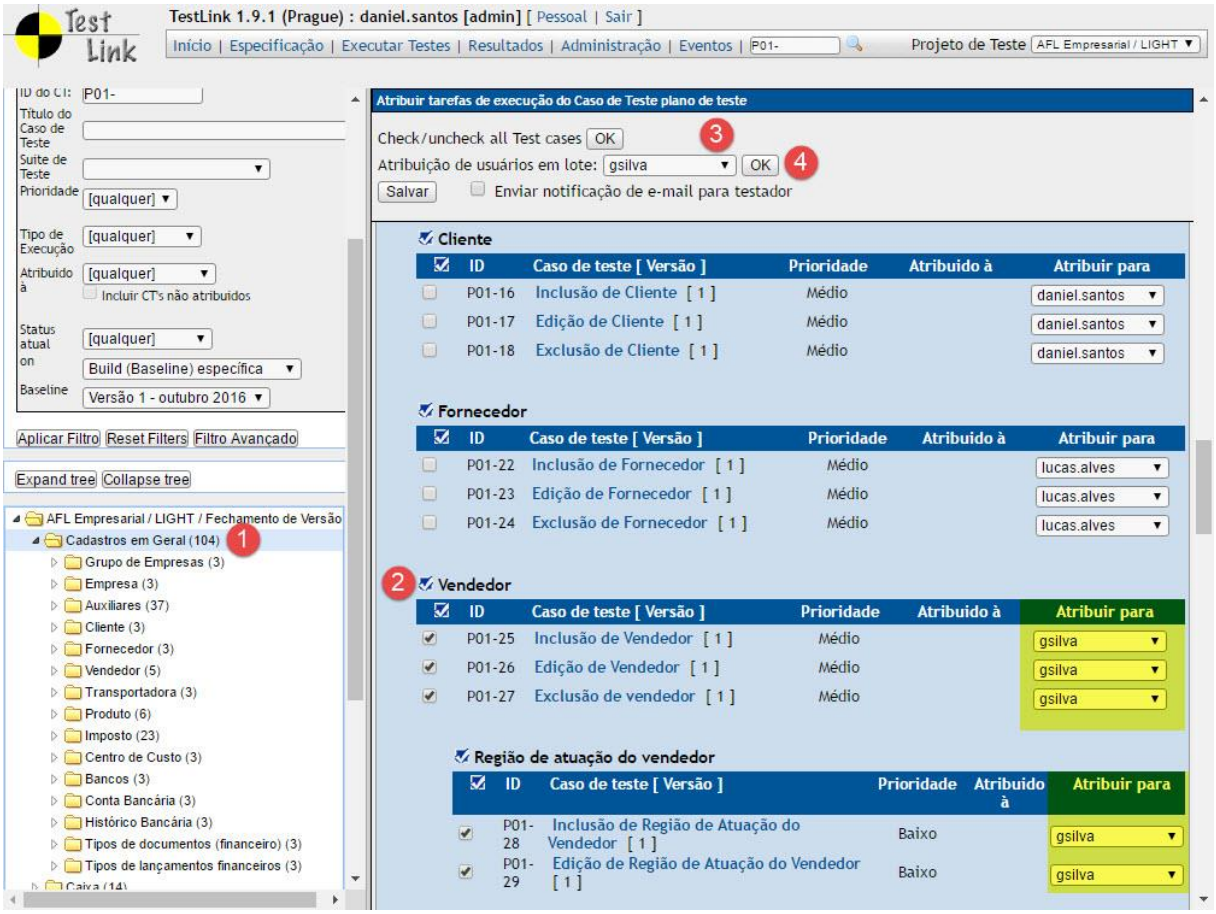


Figura 26 - Atribuindo o testador à vários Casos de Teste no TestLink (Fonte própria)

Para complementar, antes da execução dos testes, existe a possibilidade de priorização dos testes, dependendo da sua importância e da urgência, na qual o responsável atribui um conjunto de Casos de Teste com essas características ou de acordo com sua relevância, para que sejam testados primeiro. Isso ajuda a assegurar que o procedimento de teste irá cobrir os casos mais importantes.

A Figura 27 retrata a atribuição de prioridade alta para os casos de testes do suíte de teste “Cadastro em Geral / Clientes”.

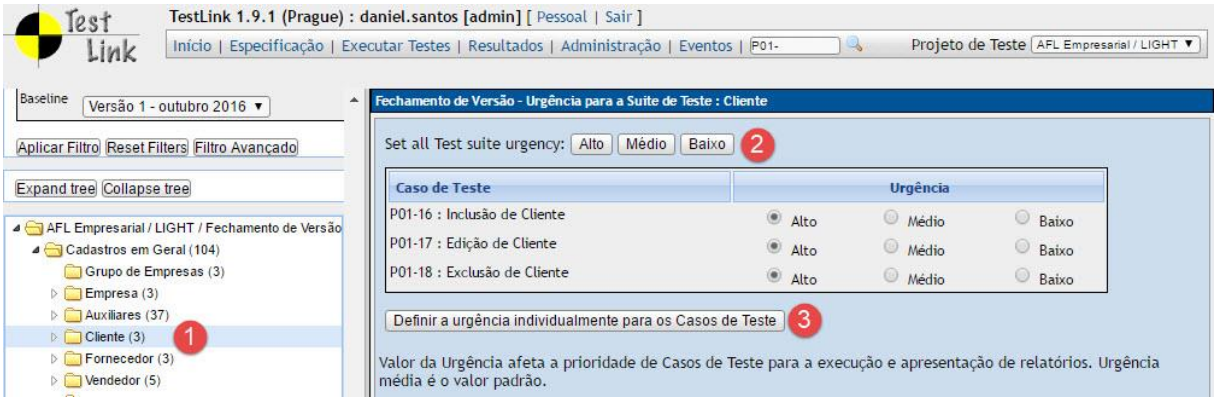


Figura 27 – Atribuindo prioridade em casos de teste no TestLink (Fonte própria)

Após os processos anteriores, é momento de executar os testes. Sendo assim, é possível visualizar os casos de testes atribuídos ao usuário autenticado no TestLink.

A Figura 28 destaca os casos de testes atribuídos ao usuário "daniel.santos", ordenados por prioridade, destacando também o status de cada caso de teste.

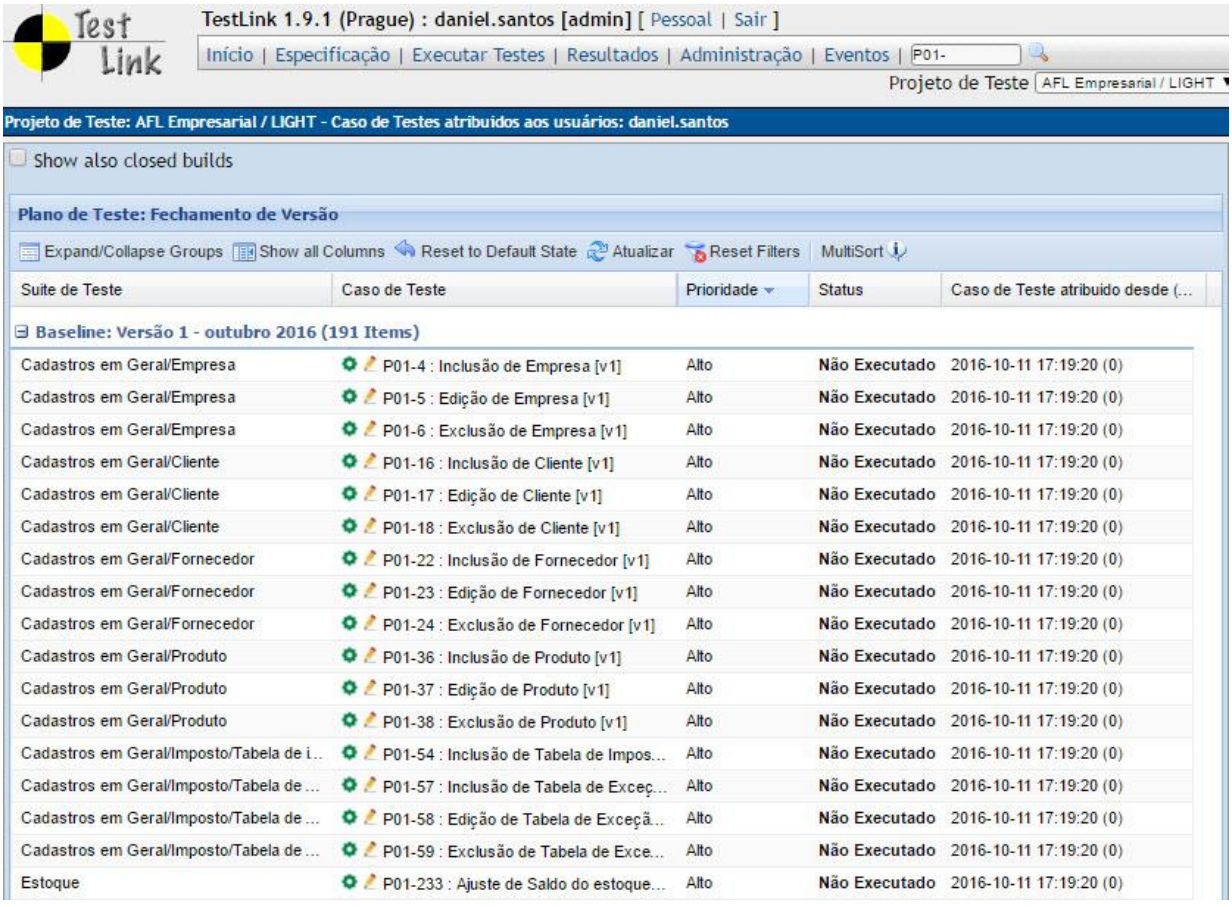


Figura 28 – Casos de testes atribuídos para o usuário "daniel.santos" no TestLink (Fonte própria)

A execução do teste é feita por meio da interação no ícone com o formato de engrenagem, da cor verde, conforme Figura 28. Essa interação faz com que uma página seja aberta e apresenta as informações do caso de teste, sendo estes, o sumário, pré-condições e os passos para a realização do teste. Também, se destaca o último resultado do teste para a *baseline*.

A Figura 29 destaca as informações que foram exibidas após a interação com a engrenagem do caso de teste "Inclusão de Clientes".

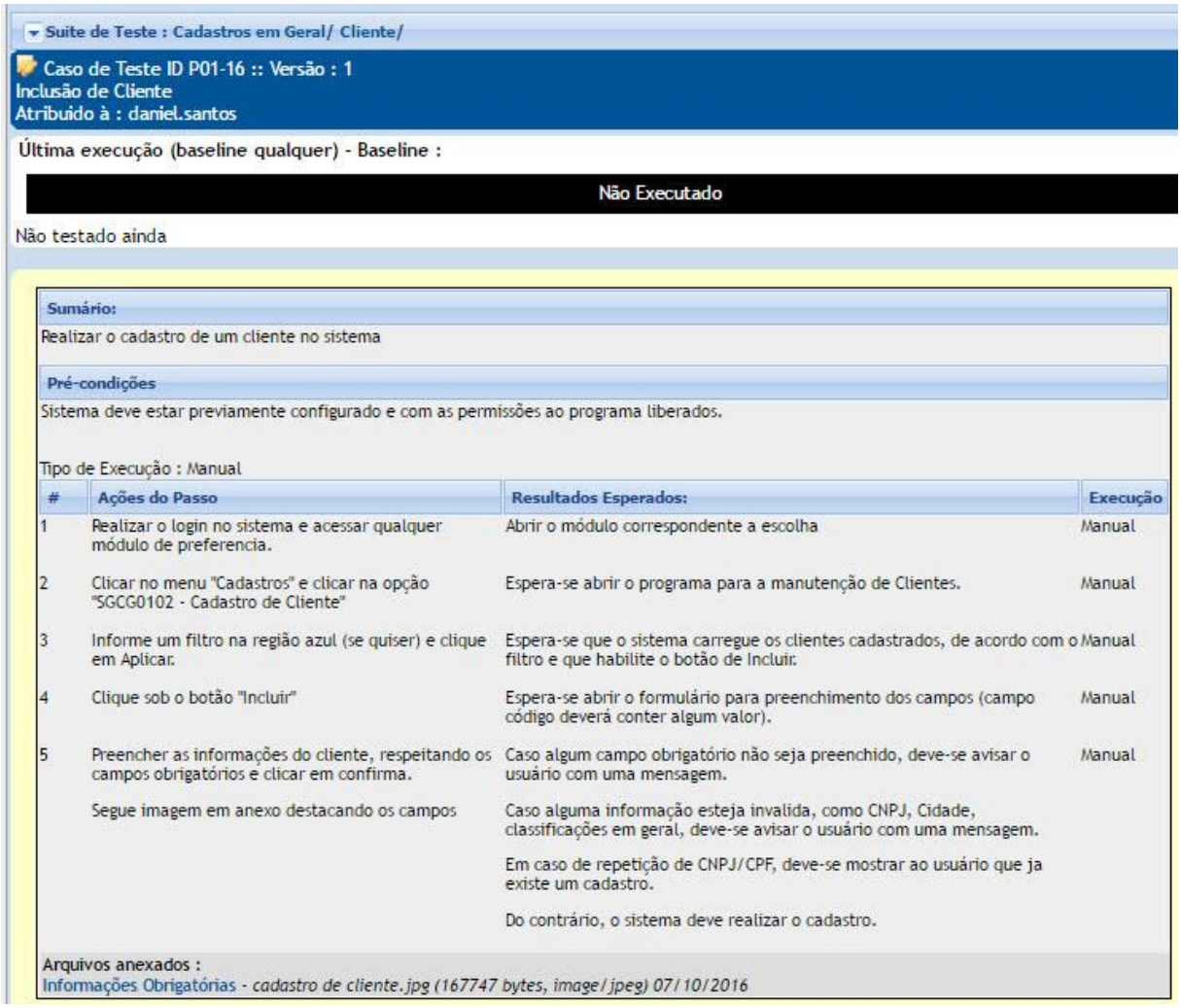


Figura 29 - Iniciando a execução do teste em um Caso de Teste no TestLink (Fonte própria)

Desta maneira, o analista responsável pela execução do teste deve realizar o procedimento de acordo com o que está descrito nos passos, respeitando as pré-condições descritas no caso de teste e avaliando se os resultados esperados foram obtidos.

Ao final da execução do teste, o analista deve informar o resultado obtido e uma descrição do resultado.

A Figura 30 expõe o caso de teste "Inclusão de Clientes" sendo aprovado pelo

analista responsável pelo teste.

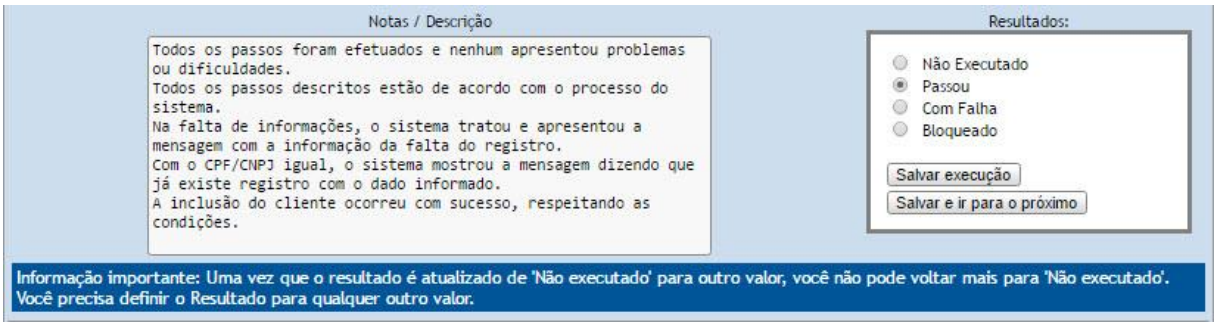


Figura 30 – Registrando o resultado da execução do teste de um Caso de Teste no TestLink (Fonte própria)

Por conseguinte, após ser atribuído um resultado, o caso de teste aparece como testado e assim é possível anexar arquivos, com os resultados dos testes.

A Figura 31 retrata como ficou após salvar a execução do caso de teste "Inclusão de Clientes".

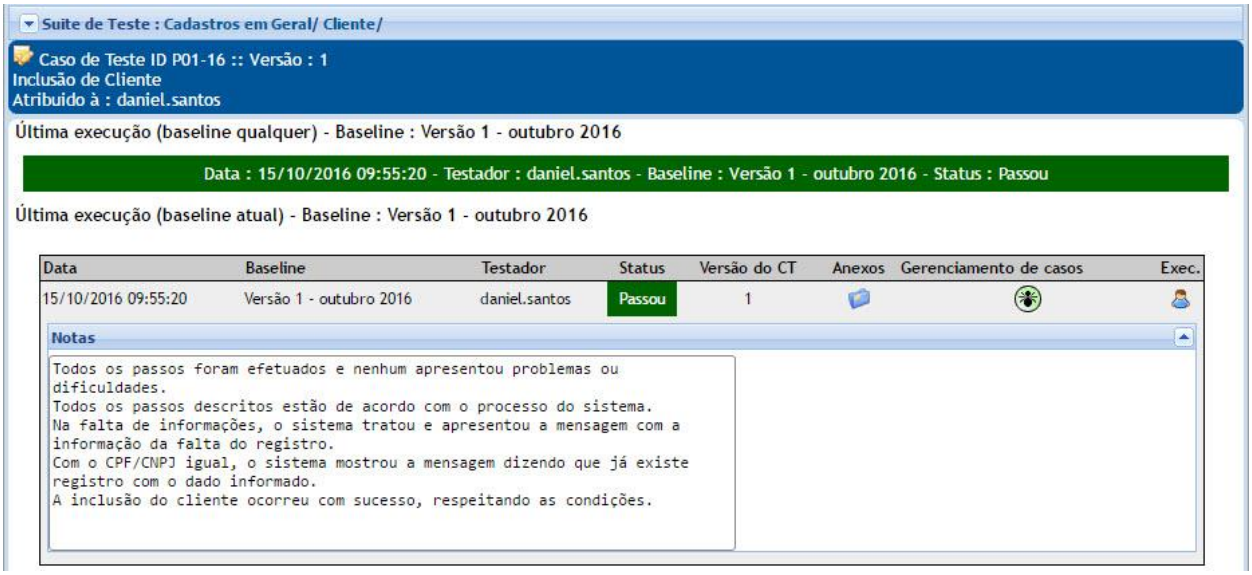


Figura 31 – Resultado após salvar a execução do teste a um Caso de Teste no TestLink (Fonte própria)

Quanto aos casos de teste que foram encontrados erros, é possível adicionar um caso do BugMantis (ou de outras ferramentas de gestão de defeitos), para registro e acompanhamento da correção do problema encontrado.

A Figura 32 estampa o caso de teste "Validação de XML [Validação automática]" com o status "com falha" e um caso relacionado a ele.

Figura 32 – Caso de teste registrado com falha e um caso do BugMantis relacionado (Fonte própria)

Assim sendo, o procedimento de teste foi padronizado nesta empresa, na qual não irá mais haver processos de testes *ad-hoc*. Isso faz com que os testes sejam mais confiáveis e mais precisos, podendo, depois, ser filtrados as situações de erros ou de sucesso e verificar quem foi o responsável pelo teste, se houve algum tipo de negligência ou algo relacionado a alguma falha no processo de teste.

A Figura 33 retrata o BMPN que descreve todo o fluxo que foi descrito neste item (3.2.5) desta monografia.

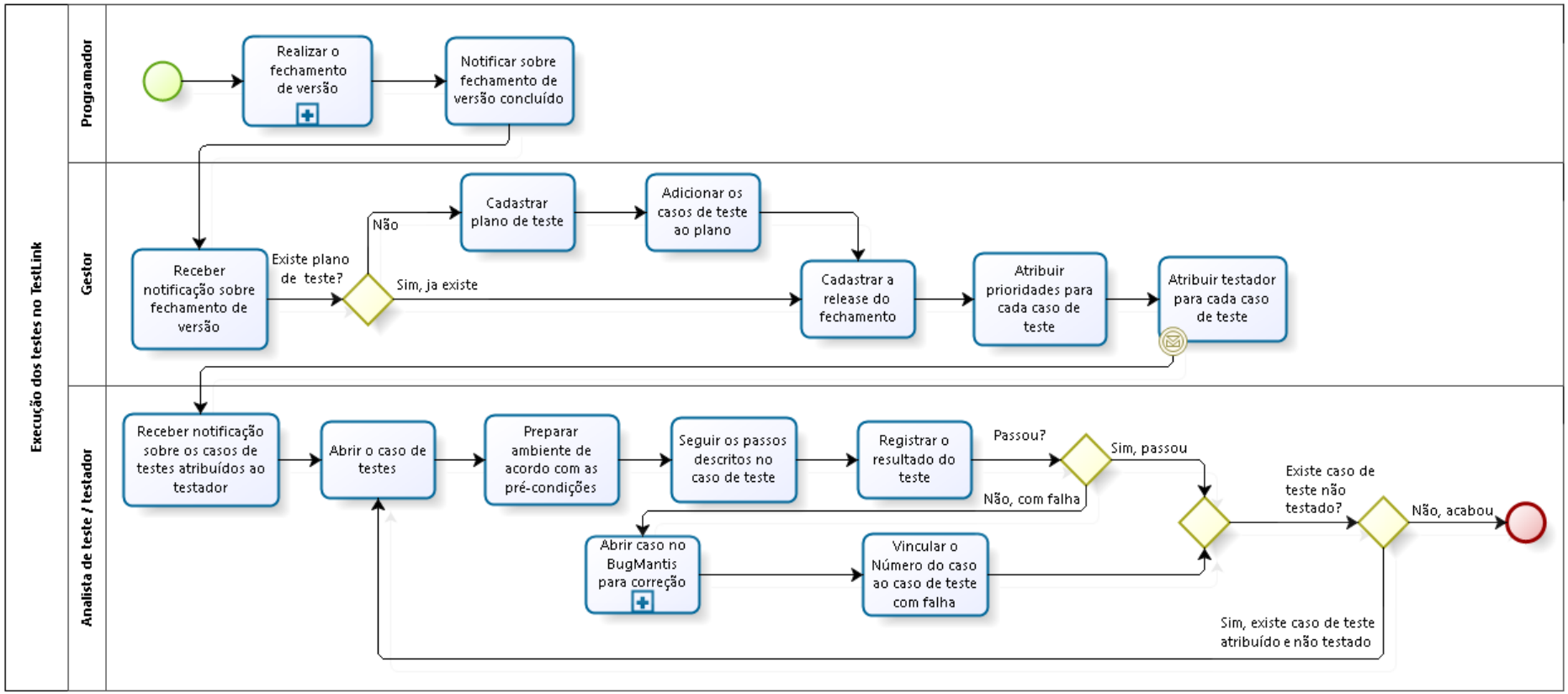


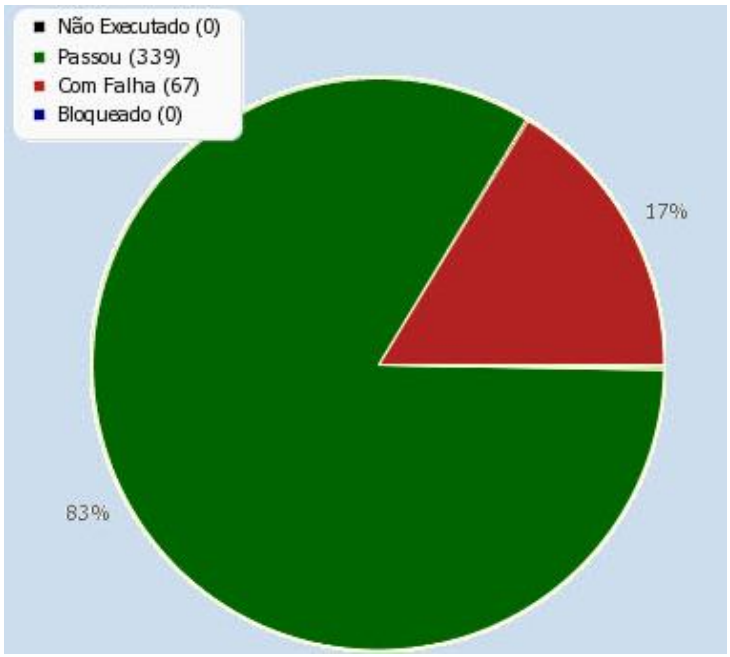
Figura 33 - BPMN do fluxo de teste, com o TestLink, Fechamento de Versão e o Mantis (Fonte Própria)

3.2.6 Resultados

Com a ferramenta TestLink padronizando os testes e com processo de o fechamento de versão acontecendo correta e periodicamente, todo o fluxo que envolve os testes tornou-se consistente, e com isso os testes no produto foram realizados com sucesso, bem como erros foram encontrados.

O Gráfico 6 destaca o resultado final dos testes efetuados, destacando que 17% (67) dos casos de testes registrados capturaram erros.

Gráfico 6 - Resultado final depois da execução dos testes (Fonte própria)



Em relação a prioridade dos casos de testes, processos importantes que antes envolviam todos os analistas para testar, o novo processo de teste encontra vários erros. A Figura 34 estampa o resultado final, agrupando por prioridade dos casos de teste.

Prioridade	Total	Não Executado	[%]	Passou	[%]	Com Falha	[%]	Bloqueado	[%]	[%] Completado
Alto	67	0	0.00	63	94.03	4	5.97	0	0.00	100.00
Médio	114	0	0.00	103	90.35	11	9.65	0	0.00	100.00
Baixo	225	0	0.00	173	76.89	52	23.11	0	0.00	100.00

Figura 34 - Resultado dos testes de acordo com prioridade (Fonte própria)

Também foi possível visualizar que a maioria dos módulos do sistema possuem ao

menos um erro. A Figura 35 exprime o resultado de acordo com o primeiro nível de suítes.

Suite de Teste	Total	Não Executado	Passou	Com Falha	Bloqueado	[%] Completado
Cadastros em Geral	104	0	78	26	0	100.0
Caixa	14	0	13	1	0	100.0
CFe	12	0	8	4	0	100.0
Comercial	69	0	64	5	0	100.0
Compras	24	0	21	3	0	100.0
Conciliação	9	0	9	0	0	100.0
ECF	11	0	9	2	0	100.0
Estoque	26	0	21	5	0	100.0
Orçamento	15	0	7	8	0	100.0
Financeiro	42	0	41	1	0	100.0
Picking	19	0	14	5	0	100.0
Pre Venda	34	0	31	3	0	100.0
XML de Entrada (SGVX)	21	0	17	4	0	100.0
Etiquetas	6	0	6	0	0	100.0

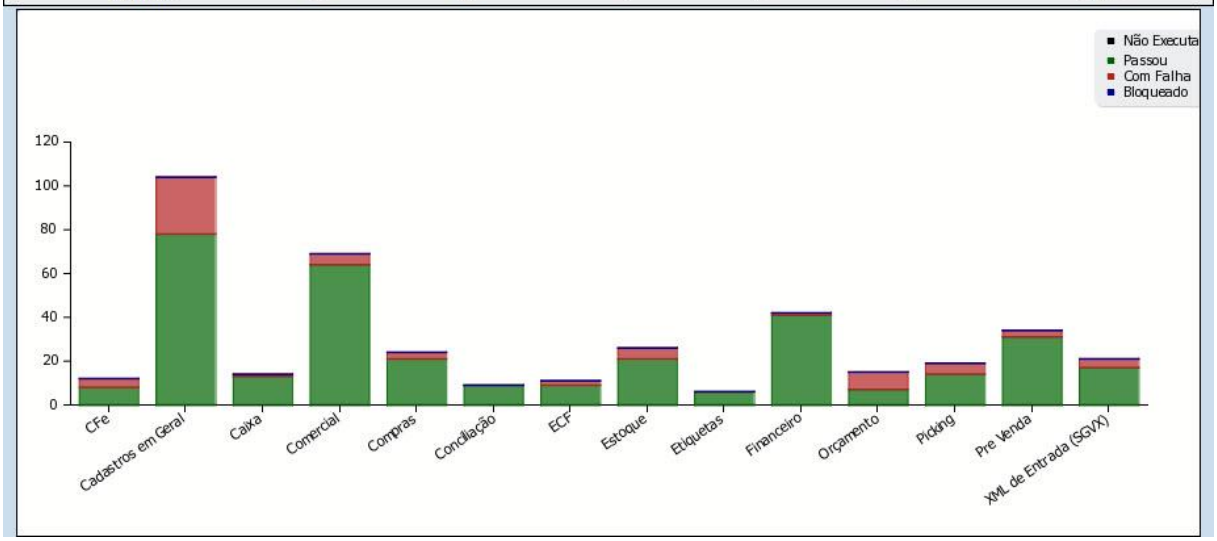


Figura 35 - Resultado de acordo com os Suítes de Teste do primeiro nível (fonte própria)

A Figura 36 destaca a lista com os casos de testes, destacando o seu status e o link do BugMantis.

Caso de teste	Baseline	Testador	Data/Hora	Status	Descrição	Bugs
P01-203: Validação de XML [Validar Automático]	Versão 1 - outubro 2016	daniel.santos	16/10/2016 10:22:07	Com Falha	Houve um problema com a Validação Automática, em relação ao item. Na atualização de relacionamentos, o sistema esta criando novos relacionamentos e mantendo os antigos, quando deveria retira-los. O sistema não esta registrando o relacionamento, quando o banco de dados está parametrizado para usa código do fabricante e o código do fornecedor é o mesmo do cadastro de produto.	6115 : Erro na validação automática no SGVX
P01-204: Validação de XML [Cancelar Validação]	Versão 1 - outubro 2016	daniel.santos	16/10/2016 17:18:14	Passou	O sistema consegue cancelar corretamente o xml da nf de entrada e retorna o status corretamente.	
P01-205: Efetivação de XML	Versão 1 - outubro 2016	daniel.santos	16/10/2016 17:15:49	Passou	O processo de efetivação ocorre corretamente, registrando as devidas movimentações de estoque e financeiro, se houverem.	
P01-305: Efetivação de XML [Alterar um centro de custo]	Versão 1 - outubro 2016	daniel.santos	16/10/2016 17:10:01	Com Falha	Encontrado falha no processo. Esta sendo possível informar centro de custo 9999 para rateio, sem informar rateio, a explosão dos valores aos centros de custos.	6116 : Erro na alteração do centro de custo no SGVX
P01-306: Efetivação de XML [Alterar todos centro de custo]	Versão 1 - outubro 2016	daniel.santos	16/10/2016 17:13:15	Com Falha	Encontrado falha no processo. Esta sendo possível informar centro de custo 9999 para rateio, sem informar rateio, a explosão dos valores aos centros de custos.	6116 : Erro na alteração do centro de custo no SGVX

Figura 36 - Casos de Teste com o *status* e caso do BugMantis vinculado (Fonte própria)

Outrossim, o novo procedimento de teste contribuirá para que a empresa tenha um produto melhor no mercado, com menos problemas e mais estável. Desta forma, o número de chamados ao suporte relacionado à erro deve cair, ocasionando em menos casos de erro para o desenvolvimento, reduzindo o custo com retrabalho e com suporte.

CONCLUSÃO

O presente trabalho baseou-se na implantação de uma metodologia de trabalho para teste e qualidade de software, buscando o baixo custo, com o uso da ferramenta *Open Source* TestLink, que proporcionou melhoras no processo de teste da empresa onde se passou o estudo de caso.

O objetivo foi atingido, sendo possível implantar a metodologia de trabalho, com o uso da ferramenta para o gerenciamento dos testes, visualizando os processos de teste, as formas de testar, não antes efetuados, que agora passaram a ser registrados e testados corretamente. Segundo relatos dos analistas, erros no sistema que foram encontrados com o novo processo de teste, não seriam encontrados facilmente pelo processo antigo.

Concluiu-se também que a gestão dos testes e dos casos de erros abertos na ferramenta de gestão de defeitos, no caso o BugMantis, houve uma melhora na padronização do fluxo de um caso do Mantis, conforme item 3.2.3 deste documento e também com alguns relatórios do TestLink.

Observou-se que a ferramenta TestLink pode ser utilizada para outros fins, como por exemplo, ter informações do que o seu produto oferece e o que não oferece. Isso porque os casos de testes acabam descrevendo todos os processos do produto. Em sendo assim, a ferramenta ajudou com a elaboração de manuais, pois os passos de cada caso de teste, teoricamente, seria o que um usuário comum do sistema teria que fazer para a execução do processo.

Observou-se também que houve uma certa resistência com a implantação do novo processo de teste, de início, por parte dos colaboradores, tendo em vista que todo o processo de trabalho que já existia na empresa foi afetado. Esta resistência ocorreu não só com a mudança dos processos de teste e de fechamento de versão, mas também com o uso da ferramenta, que demandou tempo para registrar cada Caso de Teste e realizar a execução do teste, lançando tudo no TestLink.

Para trabalhos futuros, sugere-se evoluir o processo de teste, utilizando ferramentas para deixar os testes automatizados. Além disso, pode ser possível aplicar a técnica de teste estrutural, voltada em realizar os testes direto no código fonte. Porém, a aplicação desta técnica pode gerar resistência por parte dos colaboradores do setor de desenvolvimento.

Ademais, outro projeto futuro é o de melhorar os processos da empresa, como por exemplo o registro de atendimentos e solicitações, fluxo de implantação de sistemas, entre outros, demonstrando, posteriormente, em artigos ou outros veículos, expondo os resultados, para que outras empresas de pequeno porte também possam ter seus processos melhorados.

Em relação à comunidade, como projeto futuro, é aprimorar a ferramenta *Web* que foi criado, na qual o usuário poderá até manipular a *select* e o próprio *dashboard*.

REFERÊNCIAS

AFL SISTEMAS. AFL Sistemas – MantisBT. Disponível em <http://www.aflsistemas.com.br/mantis/>. Acesso em 10 abr. 2016.

ANDRADE, Wagner. Afinal, o que é HTTP?. Disponível em <http://imasters.com.br/artigo/11513/redes-e-servidores/afinal-o-que-e-http/?trace=1519021197&source=single>. Acesso em 26 jun. 2016.

BARTIÉ, Alexandre. Garantia de Qualidade de Software: adquirindo maturidade organizacional. Rio de Janeiro: Editora Campus, 2002.

BUGZILLA. The Bugzilla Guide - 4.2.16+ Release. 2015. Disponível em <https://www.bugzilla.org/docs/4.2/en/html/>. Acesso em 09 dez. 2016.

CAETANO, Cristiano. Conhecendo os testes exploratórios – Revista Engenharia de Software Magazine 53 - Parte 1. Disponível em <http://www.devmedia.com.br/conhecendo-os-testes-exploratorios-revista-engenharia-de-software-magazine-53-parte-1/26286>. Acesso em 10 nov. 2016.

CAMPOS, Fabrício Ferrari. Teste de Desempenho/Carga/Stress | QualidadeBR Disponível em <https://qualidadebr.wordpress.com/2010/08/29/teste-de-desempenhocargastress/>. Acesso em 19 jun. 2016.

COMITÉ FRANÇAIS DES TESTS LOGICIELS. Glossaire CFTL/ISTQB des termes utilisés em tests de logiciels. Disponível em <http://en.gasq.org/fileadmin/user_upload/redaktion/en/Data/ISTQB_Glossary_French_v2_0.pdf>. Acesso em 10 nov. 2016.

COSTA, Pedro. 4 tendências para testes de software no Brasil em 2015. Disponível em: http://crowdtest.me/4-tendencias-testes-software-2015/. Acesso em 12 nov. 2015.

COSTA, Pedro. Os tipos de teste de software. Disponível em: http://crowdtest.me/tipos-teste-software/. Acesso em 12 nov. 2015

CRAIG, R.D., JASKIEL, S. P., “Systematic Software Testing”, Artech House Publishers, Boston, 2002.

CRAIG, R. D., JASKIEL, S. P., Systematic Software Testing. Boston: Artech House Publishers, 2002.

DA SILVA, Letícia Dias; DA SILVA, Evaldo de Oliveira. Artigo Engenharia de Software 20 - Introdução ao Mantis. Disponível em <http://www.devmedia.com.br/artigo-engenharia-de-software-20-introducao-ao-mantis/15462>. Acesso em 18 jun. 2016.

DE BITTENCOURT, Marcelo. #1 Ferramenta Testlink - Visão Geral: Parte 1 de 2. Disponível em: <https://www.youtube.com/watch?v=2dj6DIIVY4E>. Acesso em 19. Jun. 2016.

DE BITTENCOURT, Marcelo. #2 Ferramenta Testlink - Visão Geral: Parte 2 de 2. Disponível em: <https://www.youtube.com/watch?v=St6r9TQP8jQ>. Acesso em 19. Jun. 2016.

DELAMARO, Márcio Eduardo; MALDONADO, José Carlos; JINO, Mario; DE SOUZA, Simone do Rocio Senger; BARBOSA, Ellen Francine. Introdução ao Teste de *software*. São Carlos: Instituto de Ciências Matemáticas e de Computação, 2004.

DeMILLO, R. A; LIPTON, R, J.; SAYWARD, F.G. *Hints on test data selection: Help for the practicing programmer*. IEEE Computer, 1978. In: SIMÃO, Adenilso da Silva. Aplicação de análise de mutantes no contexto do teste e validação de redes de Petri coloridas. Tese, Universidade de São Paulo, São Carlos, SP: novembro de 2004. Disponível em: <http://www.teses.usp.br/teses/disponiveis/55/55134/tde-13042005-152434/pt-br.php>. Acesso em: 22 mai. 2016.

DEVMEDIA. Introdução ao Business Process Modeling Notation (BPMN). Disponível em <http://www.devmedia.com.br/introducao-ao-business-process-modeling-notation-bpmn/29892>. Acesso em 20 out. 2016.

DEVMEDIA. TestLink: Gerenciando atividades de teste. Disponível em <http://www.devmedia.com.br/testlink-gerenciando-atividades-de-teste/32281>. Acesso em 19 jun. 2016.

DEVMEDIA. Testes de Desempenho, Carga e Stress - Revista Java Magazine 110. Disponível em <http://www.devmedia.com.br/testes-de-desempenho-carga-e-stress-revista-java-magazine-110/26546>. Acesso em 21 jun. 2016.

EIS, Diego. O que é o HTML e para que ele serve?. Disponível em <http://tableless.com.br/o-que-html-basico/>. Acesso em 26 jun. 2016.

ELIZA, Renata. Introdução a diferentes tipos de teste. Disponível em <http://www.devmedia.com.br/introducao-a-diferentes-tipos-de-teste/29799>. Acesso em 21 jun. 2016.

ELIZA, Renata. Ferramentas de suporte ao Teste de Software. Disponível em <http://www.devmedia.com.br/ferramentas-de-suporte-ao-teste-de-software/28642>. Acesso em 10 jun. 2016.

FELIZARDO, Katia Romero. Técnicas de VV&T - Validação, Verificação e Teste. Disponível em <http://www.linhadecodigo.com.br/artigo/492/tecnicas-de-vvampt-validacao-verificacao-e-teste.aspx>. Acesso em 21 mai. 2016.

HOWDEN, W. E. Software Engineering and Technology: Functional Program Testing and Analysis. Nova Iorque: McGrall-Hill, 1987.

IEEE, IEEE - About IEEE. Disponível em <https://www.ieee.org/about/index.html>. Acesso em 23 abr. 2016.

IEEE STANDARDS COORDINATING COMMITTE. IEEE Standard Glossary of Software Engineering Terminology (IEEE Std 610.12-1990). Los Alamitos. CA. IEEE Computer Society, 1990.

INTHURN, Cândida. Qualidade & Teste de Software. Florianópolis: Visual Books, 2001.

LUFT, Cristiane Carline. Teste de Software: Uma necessidade das empresas. Trabalho de Conclusão de Curso, Universidade Regional do Noroeste do Estado do Rio Grande do Sul, RS: julho de 2006. Disponível em: <http://bibliodigital.unijui.edu.br:8080/xmlui/handle/123456789/1307>. Acesso em 02 fev. 2016.

MALDONADO, José Carlos; DELAMARO, Márcio Eduardo; JINO, Mario. Introdução ao Teste de *software*. Rio de Janeiro: Elsevier, 2007.

MANTIS BT TEAM. Mantis Bug Tracker. Disponível em <https://www.mantisbt.org/>. Acesso em 18 jun. 2016.

MYERS, Glenford J.; SANDLER, Corey; BADGETT, Tom. The art of software testing. John Wiley & Sons, Nova York, NY, EUA, 2 ed., 2011.

NETO, Arilo Claudio Dias. Artigo Engenharia de Software – Introdução a Teste de Software. Disponível em <http://www.devmedia.com.br/artigo-engenharia-de-software-introducao-a-teste-de-software/8035>. Acesso em 23 abr. 2016.

NETO, Arilo Claudio Dias. Casos de Teste: Aprimore seus casos e procedimentos de teste. Disponível em <http://www.devmedia.com.br/casos-de-teste-aprimore-seus-casos-e-procedimentos-de-teste/30526>. Acesso em 21 mai. 2016.

NOGUEIRA, Elias. Interação do TestLink com o Mantis – Elias Nogueira. Disponível em <http://eliasnogueira.com/integracao-do-testlink-com-mantis-modo-atual/>. Acesso em 14 nov. 2016.

PRESSMAN, R. S., “Software Engineering: A Practitioner’s Approach”, McGraw-Hill, 6th ed, Nova York, NY, 2005.

PRIMÃO, Aline Pacheco; RIBEIRO, Patric da Silva; KREUTZ, Diego Luís. Estudo de Caso: Técnicas de Teste como parte do Ciclo de Desenvolvimento de Software. Universidade Federal do Pampa - UNIPAMPA, Alegrete, RS, 2010. Disponível em: <http://www.sirc.unifra.br/artigos2010/4.pdf>. Acesso em: 21 jun. 2016.

RAMOS, Ricardo A. Introdução ao Teste de Software. Disponível em <http://www.univasf.edu.br/~ricardo.aramos/disciplinas/ESI2009_2/Aula19_V&VTesteSoftware.pdf>. Acesso em 22 mai. 2016.

RIBEIRO, Rafael Dias. Teste de Partição de Equivalência. Disponível em <http://www.rafaeldiasribeiro.com.br/downloads/testesw3.pdf>. Acesso em 21 mai. 2016.

SOFTPEDIA. Bugzilla Download Mac. Disponível em <http://mac.softpedia.com/get/Developer-Tools/Bugzilla.shtml>. Acesso em 18 jun. 2016.

SOFTEX. Fundamentos de Teste de Software. 2011. Disponível em <http://ava.nac.softex.br/pluginfile.php/598/mod_resource/content/2/Aula1.pdf>.

SOUSA, Markelly. O Processo de Teste: Testes de Carga. Disponível em <http://thetestingprocess.blogspot.com.br/2012/08/testes-de-carga.html>. Acesso em 21 jun. 2016.

SOFTWARE TESTING GENIUS. Ad-hoc testing. Disponível em <http://www.softwaretestinggenius.com/ad-hoc-testing>. Acesso em 10 nov. 2016.

TEODORO, Danilo Santos; SILVEIRA, Sarah Gonçalves. O cenário da Qualidade de Software ← TI & Gestão. Disponível em <http://www.tiegestao.com.br/2014/12/14/o-cenario-da-qualidade-de-software>. Acesso em 12 nov. 2015.

TOZELLI, Paulo. Teste de Software -. Disponível em <http://imasters.com.br/artigo/9572/software/teste-de-software?trace=1519021197&source=single>. Acesso em 21 jun. 2016.

TUTORIALS POINT. Adhoc Testing. Disponível em <https://www.tutorialspoint.com/software_testing_dictionary/adhoc_testing.htm>. Acesso em 10 nov. 2016.

TONI, Adriana Pinheiro; FRANQUEIRA, Raphael Valery Leal. Common Gateway Interface – CGI. Disponível em <http://homepages.dcc.ufmg.br/~mlbc/cursos/internet/cgi/cgi.html>. Acesso em 18 jun. 2016.

W3C. Extensible Markup Language (XML), 2011. Disponível em <https://www.w3.org/XML/>. Acesso em 10 nov. 2016.

W3C. CGI - Common Gateway Interface, 2009. Disponível em <https://www.w3.org/CGI/>. Acesso em 18 jun. 2016.

WITCZAK, Ariel Bolzan. Guia Técnica de Teste. Celepar Informática do Paraná, PR: agosto de 2009. Disponível em: <http://www.documentador.pr.gov.br/documentador/pub.do?action=d&uuid=@gtf-escriba@d9bbb686-a5cd-432d-ad3f-ba77d6c427f2>. Acesso em: 21 jun. 2016.

APÊNDICE A – *SELECTS* UTILIZADAS NOS *DASHBOARDS*

Este apêndice contém as *selects* que foram utilizadas nos *dashboards*.

```
select mantis_bug_table.id as "Código",
       mantis_project_table.name as "Projeto",
       mantis_custom_field_string_table.value as "Cliente"
       mantis_bug_table.summary as "Descrição",
       mantis_category_table.name as "Categoria",
       CASE mantis_bug_table.priority
         WHEN 10 THEN "Nenhum"
         WHEN 20 THEN "Baixo"
         WHEN 30 THEN "Normal"
         WHEN 40 THEN "Alto"
         WHEN 50 THEN "Urgente"
         WHEN 60 THEN "Imediato"
         ELSE "Não atribuido"
       END AS "Prioridade",
       CASE mantis_bug_table.severity
         WHEN 10 THEN "Recurso"
         WHEN 20 THEN "Trivial"
         WHEN 30 THEN "Texto"
         WHEN 40 THEN "Minimo"
         WHEN 50 THEN "Pequeno"
         WHEN 60 THEN "Grande"
         WHEN 70 THEN "Travamento"
         WHEN 80 THEN "Obstaculo"
         ELSE "Não atribuido"
       END AS "Severidade",
       CASE mantis_bug_table.status
         WHEN 10 THEN "Novo"
         WHEN 20 THEN "Retorno"
         WHEN 30 THEN "Admitido"
         WHEN 40 THEN "Confirmado"
         WHEN 50 THEN "Atribuído"
         WHEN 80 THEN "Resolvido"
         WHEN 90 THEN "Fechado"
       END AS "Status",
       CASE mantis_bug_table.resolution
         WHEN 10 THEN "Aberto"
         WHEN 20 THEN "Fixado"
         WHEN 30 THEN "Reaberto"
         WHEN 40 THEN "Incapaz de reproduzir"
         WHEN 50 THEN "Não fixado"
         WHEN 60 THEN "Duplicado"
         WHEN 70 THEN "Não é um bug"
         WHEN 80 THEN "Suspenso"
         WHEN 90 THEN "Não será corrigido"
       END AS "Resolução",
       FROM_UNIXTIME(mantis_bug_table.date_submitted) as "Data de abertura",
       FROM_UNIXTIME(mantis_bug_table.last_updated) as "Ultima atualização",
       case
         when mantis_bug_table.status in (80, 90) then
          DATEDIFF(FROM_UNIXTIME(mantis_bug_table.last_updated),
                   FROM_UNIXTIME(mantis_bug_table.date_submitted))
         else
          DATEDIFF(now(), FROM_UNIXTIME(mantis_bug_table.date_submitted))
       end as "Tempo aberto ou para resolver"
  from mantis_bug_table
  LEFT join mantis_project_table
    on mantis_project_table.id = mantis_bug_table.project_id
  LEFT join mantis_category_table
    on mantis_category_table.id = mantis_bug_table.category_id
  left join mantis_custom_field_string_table
    on mantis_custom_field_string_table.bug_id = mantis_bug_table.id
 where mantis_custom_field_string_table.field_id = 6
```

Figura 37 - *Select* usada no *dashboard* com todos os casos (Fonte própria)

```
select mantis_bug_table.id as "Código",
       mantis_project_table.name as "Projeto",
       mantis_custom_field_string_table.value as "Cliente"
       mantis_bug_table.summary as "Descrição",
       mantis_category_table.name as "Categoria",
       CASE mantis_bug_table.priority
         WHEN 10 THEN "Nenhum"
         WHEN 20 THEN "Baixo"
         WHEN 30 THEN "Normal"
         WHEN 40 THEN "Alto"
         WHEN 50 THEN "Urgente"
         WHEN 60 THEN "Imediato"
         ELSE "Não atribuido"
       END AS "Prioridade",
       CASE mantis_bug_table.severity
         WHEN 10 THEN "Recurso"
         WHEN 20 THEN "Trivial"
         WHEN 30 THEN "Texto"
         WHEN 40 THEN "Minimo"
         WHEN 50 THEN "Pequeno"
         WHEN 60 THEN "Grande"
         WHEN 70 THEN "Travamento"
         WHEN 80 THEN "Obstaculo"
         ELSE "Não atribuido"
       END AS "Severidade",
       CASE mantis_bug_table.status
         WHEN 10 THEN "Novo"
         WHEN 20 THEN "Retorno"
         WHEN 30 THEN "Admitido"
         WHEN 40 THEN "Confirmado"
         WHEN 50 THEN "Atribuído"
         WHEN 80 THEN "Resolvido"
         WHEN 90 THEN "Fechado"
       END AS "Status",
       CASE mantis_bug_table.resolution
         WHEN 10 THEN "Aberto"
         WHEN 20 THEN "Fixado"
         WHEN 30 THEN "Reaberto"
         WHEN 40 THEN "Incapaz de reproduzir"
         WHEN 50 THEN "Não fixado"
         WHEN 60 THEN "Duplicado"
         WHEN 70 THEN "Não é um bug"
         WHEN 80 THEN "Suspenso"
         WHEN 90 THEN "Não será corrigido"
       END AS "Resolução",
       FROM_UNIXTIME(mantis_bug_table.date_submitted) as "Data de abertura",
       FROM_UNIXTIME(mantis_bug_table.last_updated) as "Ultima atualização",
       case
         when mantis_bug_table.status in (80, 90) then
          DATEDIFF(FROM_UNIXTIME(mantis_bug_table.last_updated),
                   FROM_UNIXTIME(mantis_bug_table.date_submitted))
         else
          DATEDIFF(now(), FROM_UNIXTIME(mantis_bug_table.date_submitted))
       end as "Tempo aberto ou para resolver",
       count(mantis_bug_history_table.id) as "Qtde Retornos"
  from mantis_bug_history_table
  LEFT join mantis_bug_table
    on mantis_bug_history_table.bug_id = mantis_bug_table.id
  LEFT join mantis_project_table
    on mantis_project_table.id = mantis_bug_table.project_id
  LEFT join mantis_category_table
    on mantis_category_table.id = mantis_bug_table.category_id
  left join mantis_custom_field_string_table
    on mantis_custom_field_string_table.bug_id = mantis_bug_table.id
 where mantis_bug_history_table.field_name = "status"
   and mantis_bug_history_table.new_value = 20
   and mantis_custom_field_string_table.field_id = 6
 group by mantis_bug_table.id,
          mantis_project_table.name,
          mantis_custom_field_string_table.value,
          mantis_bug_table.summary,
          mantis_category_table.name,
          CASE mantis_bug_table.priority,
          CASE mantis_bug_table.severity,
          CASE mantis_bug_table.status,
          CASE mantis_bug_table.resolution,
          FROM_UNIXTIME(mantis_bug_table.date_submitted),
          FROM_UNIXTIME(mantis_bug_table.last_updated),
          case
              when mantis_bug_table.status in (80, 90) then
             DATEDIFF(FROM_UNIXTIME(mantis_bug_table.last_updated),
                      FROM_UNIXTIME(mantis_bug_table.date_submitted))
            else
             DATEDIFF(now(), FROM_UNIXTIME(mantis_bug_table.date_submitted))
          end
 order by mantis_bug_table.id asc
```

Figura 38 - *Select* usada no *dashboard* com os casos de retorno (Fonte própria)