UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE SISTEMAS DE INFORMAÇÃO – BACHARELADO

SISTEMA WEB DE AUXÍLIO AO PROJETO E EXECUÇÃO DE TESTE BASEADO NO MODELO MPT.BR

TAMARA FONTANELLA DE LIMA

BLUMENAU
2016

TAMARA FONTANELLA DE LIMA

SISTEMA WEB DE AUXÍLIO AO PROJETO E EXECUÇÃO DE TESTE BASEADO NO MODELO MPT.BR

Trabalho de Conclusão de Curso apresentado ao curso de graduação em Sistemas de Informação do Centro de Ciências Exatas e Naturais da Universidade Regional de Blumenau como requisito parcial para a obtenção do grau de Bacharel em Sistemas de Informação.

Prof. Everaldo Artur Grahl, Mestre - Orientador

BLUMENAU
2016

SISTEMA WEB DE AUXÍLIO AO PROJETO E EXECUÇÃO DE TESTE BASEADO NO MODELO MPT.BR

Por

TAMARA FONTANELLA DE LIMA

Trabalho de Conclusão de Curso aprovado para obtenção dos créditos na disciplina de Trabalho de Conclusão de Curso II pela banca examinadora formada por:

Presidente:	Prof. Everaldo Artur Grahl, Mestre – Orientador, FURB
Membro:	Prof. Samuel Cristhian Schwebel, Mestre – FURB
Membro:	Prof. Matheus Luan Krueger, Mestre – FURB

Blumenau, 7 de dezembro de 2016.

Dedico este trabalho a todos que acreditaram e ajudaram diretamente na realização deste.

AGRADECIMENTOS

A Deus, pela vida, saúde e por cada desafio superado.

À minha família, Osmar e Leda pelos ensinamentos, cobranças e por acreditarem na minha capacidade.

À minha amiga, Patrícia Gonsalves, pelo empenho e contribuição possibilitando a conclusão deste trabalho.

Aos meus amigos, pela compreensão nas ausências e apoio nos momentos difíceis.

Ao meu orientador, professor Everaldo Artur Grahl, que acreditou na conclusão deste trabalho.

The greatest thing is when you do put your heart and soul into something over an extended period of time, and it is worth it.

Steve Jobs

RESUMO

Para atingir índices cada vez maiores de produtividade e qualidade é fundamental que uma organização de software adote padrões e modelos de qualidade. Este trabalho apresenta o desenvolvimento de um sistema que suporta a área de processo de Projeto e Execução de Teste (PET) do modelo de Melhoria no Processo de Teste Brasileiro (MPT.BR), possibilitando o planejamento do caso de teste, execução, reporte de incidentes e respectivo acompanhamento. O sistema foi desenvolvido utilizando a linguagem Python através do *framework* web Django e banco de dados PostgreSQL. O sistema foi validado através de uma avaliação informal por usuários de uma organização de desenvolvimento de software que concluíram como de fácil manuseio e agilidade nos processos básicos de gerenciamento de um caso de teste.

Palavras-chave: Teste de software. MPT.BR. Projeto e execução de teste.

ABSTRACT

To achieve higher levels of quality and productivity, it is essential that a software organization adopts standards and quality models. This work presents the development of a system that supports the Test Design and Execution (PET) process field of the Improvement model in the Brazilian Testing Process (MPT.BR), allowing test case planning, execution, incident report and monitoring. The system was developed using Python Programming language through Django web framework and PostgreSQL database. The system was validated through an informal assessment by users of a software development company, who have concluded that such system is easy to handle and provides agility to test case is management processes.

Key-words: Software testing. MPT.BR. Test design and execution.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

FURB – Universidade Regional de Blumenau

GCC – Grupo de Controle da Configuração

GPT – Gerência de Projetos de Teste

IDE – Integrate Development Environment

JSP – Java Server Page

MER – Modelo Entidade Relacionamento

MPT.BR – Melhoria do Processo de Teste Brasileiro

MTV – Model Template View

MVC – Model View Controller

PET – Projeto e Execução de Teste

RF – Requisito Funcional

SGBDR – Sistema Gerenciador de Banco de Dados Relacional

SSL – Secure Socket Layer

TMM – Testing Maturity Model

TSM – Testability Support Model

SUMÁRIO

1 INTRODUÇÃO

A qualidade é encarada como um conjunto de atributos necessários à sobrevivência das organizações em um mercado altamente competitivo que envolve o estabelecimento de objetivos e metas de toda a organização (COSTA et al., 2013). Segundo Colombo e Guerra (2009, p. 38), qualidade de software compõe uma área cuja demanda cresce significativamente, em que os usuários cada vez mais exigem eficácia e eficiência. A qualidade do produto de software deve fazer parte do processo de desenvolvimento e manutenção de maneira intensa, diminuindo assim os problemas encontrados no produto final.

Atualmente muitas empresas não possuem um processo bem definido e encontram dificuldade em concluir os projetos dentro do prazo, com custo baixo e qualidade satisfatória. Os processos aderentes aos modelos de maturidade têm por objetivo auxiliar às organizações encontrarem os resultados desejados através da melhor execução das atividades planejadas (MELHORIA DO PROCESSO DE TESTE BRASILEIRO, 2011).

Para Bastos et al. (2012), o custo da correção dos defeitos encontrados no produto tende a subir quanto mais tarde são corrigidos. As correções dos defeitos na fase de desenvolvimento do projeto custam menos que a de defeitos encontrados por usuários. Sendo assim, o retorno de investimento será maior à medida que as organizações investirem em testes. Quanto maior a melhoria na atividade de teste, melhor serão os resultados financeiros.

Uma forma de identificar os defeitos é por meio do teste de software, cujo objetivo é avaliar sua qualidade ou possibilitar melhorias no software apontando seus defeitos. Gandara (2012, p. 6) descreve os benefícios sobre as atividades de teste de software, que são: controle no processo de teste, maior satisfação dos usuários/clientes e reduzir os custos entre o desenvolvimento e os testes antes de colocar o sistema em produção.

1.1 PROBLEMA

Para Koscianski e Soares (2006), as empresas buscam em planilhas de dados uma organização para o grande volume de informações geradas pelas atividades de teste e, ao longo do prazo, é provável que o uso das planilhas se torne inviável. Um mal controle na fase de teste desencadeia uma série de problemas no processo, como falta de documentação, acompanhamento falho desde a execução do teste até os resultados finais obtidos e em alguns casos a prioridade e criticidade não serem atribuídas de forma correta aos incidentes encontrados.

Com o mercado cada vez mais competitivo, pessoas cada vez mais informadas e clientes muito exigentes, torna-se necessário as empresas possuírem um controle de qualidade eficiente em seus produtos. Ferramentas para planejamento e acompanhamento de testes são fundamentais para agilizar e controlar o processo de teste.

1.2 JUSTIFICATIVA

Segundo Hirama (2012, p. 4), para atingir índices cada vez maiores de produtividade e qualidade, é fundamental que uma empresa fabricante de software adote padrões e modelos de qualidade. Estes modelos e padrões garantem uma melhora no processo e uma entrega do produto final com um grau de satisfação elevado.

Portanto, viu-se uma oportunidade de desenvolver um sistema que contemple processos da área de teste de uma empresa de software e utilize como base o modelo de referência em Melhoria do Processo de Teste Brasileiro (MPT.BR). Este modelo de referência reúne as melhores práticas organizadas conforme o grau de complexidade e nível de maturidade, proporcionando benefícios à empresa que aplica este modelo.

1.3 OBJETIVOS

O objetivo geral deste trabalho é o desenvolvimento de um sistema web que suporte o processo de Projeto e Execução de Teste definidos pelo modelo de referência MPT.BR.

Os objetivos específicos do trabalho são:

a) atender as práticas PET previstas no nível 1 do modelo;
b) permitir gerar relatórios de controle dos testes;
c) realizar uma avaliação informal do sistema proposto.

1.4 ESTRUTURA

Este trabalho está organizado em quatro capítulos. No primeiro capítulo tem-se a introdução do trabalho e os objetivos. O segundo capítulo apresenta a fundamentação teórica com conceitos de teste de software, MPT.BR e trabalhos correlatos. No terceiro capítulo é demonstrado o desenvolvimento do trabalho através de requisitos, especificação, implementação, resultados e operacionalidade da aplicação. O quarto capítulo apresenta as conclusões e as sugestões para trabalhos futuros.

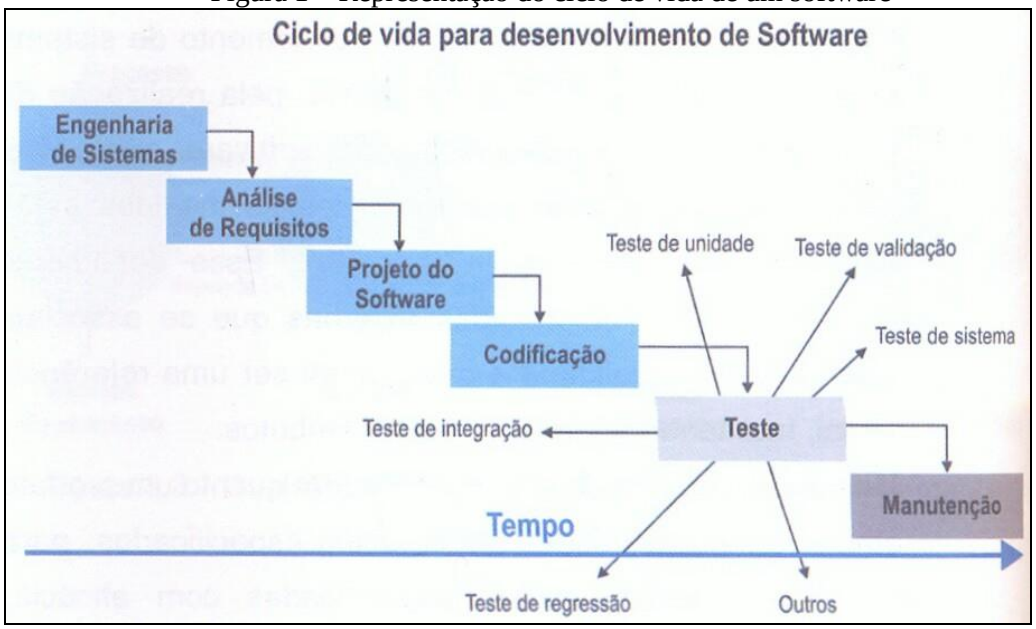
2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo aborda os conceitos aplicados no trabalho, tais como Teste de Software, Melhoria do Processo de Teste Brasileiro (MPT.BR), Projeto e Execução de Teste, além de Trabalhos Correlatos.

2.1 TESTE DE SOFTWARE

Durante a produção de um software, é necessário ter o conhecimento da avaliação do produto, se está de acordo com os requisitos especificados e também se obedece às normas de qualidade. Teste de software é a atividade de executar um software com o objetivo de encontrar diferenças entre os resultados obtidos e os resultados esperados. O ciclo de vida clássico de um software tem as fases demonstradas na Figura 1, onde uma das fases representa a fase de teste (COLOMBO; GUERRA, 2009).

Figura 1 – Representação do ciclo de vida de um software



Fonte: Colombo e Guerra (2009).

O processo de teste de software exibe uma estruturação em atividades bem definidas, etapas, artefatos (hardware e software), papéis e responsabilidades de cada integrante, que buscam desta forma a sistematização dos procedimentos e controle dos projetos (COLOMBO; GUERRA, 2009). O plano de teste utilizado deve considerar as técnicas mais adequadas à realidade da empresa desenvolvedora de software. Segundo Costa et al. (2013), não existe técnica de teste perfeita ou "melhor técnica" para todas as circunstâncias. A definição do plano de teste deve contemplar o ambiente que será testado o sistema, quais partes do sistema

serão testadas, módulos e a especificação de quais testes serão executados (GANDARA, 2012).

2.1.1 Caso de Teste

O principal objetivo da fase de elaboração de teste é a criação dos cenários de testes que serão executados na fase seguinte. Um cenário representa uma história hipotética com intuito de ajudar na solução de um problema, recriando um caminho para seguir. Define-se formalmente um caso de teste como uma especificação mais trabalhada do teste, estabelece as informações que serão empregadas durantes os testes dos cenários e os resultados esperados (BASTOS et. al., 2007).

Para Bartié (2002) cada caso de teste representa uma situação diferente e única de comportamento no software, com o objetivo de identificar um defeito não previsto durante o ciclo de desenvolvimento. Através dos casos de teste é possível monitorar os avanços da qualidade de um software, analisando os históricos de cobertura de teste no decorrer das contínuas interações do desenvolvimento de um software, tornando assim os casos de testes em elementos essenciais no processo de teste de software.

2.1.2 Execução de Teste

Segundo Bastos et al. (2007) em cada etapa do processo de teste é preciso executar os testes e analisar os resultados esperados. Todos os registros de execução de teste devem ser contidos em uma ferramenta de gestão de teste, permitindo que seja acompanhado o progresso desta execução.

O teste executado no software possui duas perspectivas diferentes: avaliação dos requisitos de software que são executados usando técnicas de projeto de casos de teste caixa-preta e a lógica interna do programa ou código que é executada utilizando técnicas de casos de teste de caixa-branca. Ambas formas de teste possuem o mesmo objetivo, encontrar o maior número de erros possíveis com a menor quantidade de esforço e tempo (COSTA et al., 2013).

2.1.3 Gestão de Defeitos

Defeitos são falhas no comportamento do software em relação aos requisitos estabelecidos e através dos defeitos é possível determinar a distância a ser percorrida até atingir o patamar de qualidade desejado. Devem ser analisados combinando diversas técnicas e ferramentas, desde simples contagens de defeitos até complexas análises estatísticas.

Através desta análise que decisões importantes serão tomadas, como a finalização de uma etapa no processo, implantação do software em produção ou até mesmo aumentar prazos e recursos no projeto (BARTIÉ, 2002).

Para Bastos et al. (2007) ao reportar um defeito, devemos fazê-lo com o objetivo de ajudar seu entendimento pelo desenvolvedor, descrevendo a situação na qual foi encontrado, os dados digitados e a operação que resultou o defeito, além disso deve-se indicar o impacto que ele representa para o sistema e para os negócios.

2.2 MELHORIA NO PROCESSO DE TESTE BRASILEIRO (MPT.BR)

O modelo Melhoria do Processo de Teste Brasileiro (MPT.BR) reúne as melhores práticas relativas às atividades desenvolvidas ao longo do ciclo de vida de teste do produto. O público alvo do modelo são pessoas interessadas em melhoria do processo com ênfase em teste de software focado em micro e pequenas empresas. Seus principais objetivos são (MELHORIA DO PROCESSO DE TESTE BRASILEIRO, 2011):

a) tornar-se um modelo de referência para definição, implantação e melhoria nos processos de teste;
b) abordar a melhoria contínua nos processos de teste conforme os objetivos organizacionais e nível de maturidade almejado;
c) fornecer uma base para avaliação e consequente identificação do grau de maturidade presente nas organizações;
d) reunir as melhores práticas e estruturá-las segundo o grau de complexidade versus o nível de maturidade que a mesma estará relacionada.

Para a criação do MPT.BR foram utilizados como base outros modelos de referência em teste de software e modelos de referência em melhoria de processo de software, como: Testability Support Model (TSM), Testing Maturity Model (TMM), Test Process Improvement (TPI), Test Organization Maturity (TOM™), Testing Assessement Program (TAP), Testing Improvement Model (TIM), Testing Maturity Model Integration (TMMI), Maturity Model for Automated Software Testing (MMAST), Modelo de Melhoria de Teste (MMT), Capability Maturity Model Integration (CMMI) e Melhoria de Processo de Software Brasileiro (MPS.BR) (MELHORIA DO PROCESSO DE TESTE BRASILEIRO, 2011).

Um dos modelos, o TMM que foi desenvolvido por Burnstein et al. em 1996 através do Illinois Institute of Technology (IIT) com o intuito de complementar a engenharia de software orientada a Capability Maturity Model (CMM) com a principal razão de não possuir

um modelo de maturidade que abordasse adequadamente as questões de teste. O objetivo do TMM é apoiar a avaliação e as unidades de melhoria dentro de uma organização através de níveis de maturidade. Os níveis são divididos em cinco (SWINKELS, 2000):

a) inicial;
b) definição da fase;
c) integração;
d) gestão e medição;
e) otimização / prevenção de defeitos e controle de qualidade.

Outro modelo existente é o TSM, segundo Swinkels (2000) foi desenvolvido por David Gelperin em 1996, com o objetivo de identificar questões para melhorar a testabilidade e o modelo consiste apenas em um modelo de maturidade. Possuir três níveis e seis áreas de suporte. Onde as áreas de suporte são (SWINKELS, 2000):

a) infra-estrutura de engenharia de software;
b) planos de projeto;
c) informação do produto;
d) desenho de software;
e) testware;
f) ambientes de teste.

2.2.1 Níveis de Maturidade

O modelo de Melhoria do Processo de Teste Brasileiro (2011) possui cinco níveis de maturidade e cada nível representa patamares para a evolução do processo de teste de uma organização, conforme relacionado a seguir:

a) parcialmente gerenciado: este nível contém o mínimo que uma organização precisa para demonstrar que a disciplina de teste é aplicada e que esta aplicação ocorre de forma planejada e controlada nos projetos;
b) gerenciado: neste nível a aplicação do processo de teste possui maior visibilidade. O projeto é controlado pelo processo de gestão de mudanças, padrões são definidos e os processos são controlados e monitorados;
c) definido: o teste se torna organizacional no terceiro nível. São definidas responsabilidades para a organização de teste e um programa de medição é implantado. Processos padrões de teste são adotados e a garantia de qualidade é estabelecida de modo a auxiliar a definição dos processos. Neste nível o ciclo de vida do teste é integrado ao ciclo de vida do desenvolvimento;

d) prevenção de defeitos: nível focado na prevenção de defeitos e melhoria sistemática da qualidade do produto. Neste patamar um processo de gestão de defeitos existe na organização, em que defeitos encontrados no início do ciclo de vida são acompanhados e ações proativas são tomadas para evitar que novos defeitos sejam originados pelas mesmas causas raiz. Uma análise de risco dos atributos não-funcionais do produto e atividades de teste não-funcional são executados para minimizar estes riscos e uma análise para determinar a eficácia do teste;
e) automação e otimização: tem como objetivo estabelecer um processo de melhoria continua e automação do teste. O processo é controlado estatisticamente e está sob continua melhoria.

2.2.2 Áreas do processo do MPT.BR

Conforme o Quadro 1 a organização das áreas de processo do MPT.BR são divididas por nível de maturidade, área de processo, práticas genéricas e práticas específicas. As práticas devem ser implantadas para garantir a aderência nas áreas conforme o nível de maturidade. Para maiores informações pode-se verificar no modelo de Melhoria do Processo de Teste Brasileiro (2011).

Quadro 1 - Nível de maturidade e área de processo do MPT.Br

Nível de Maturidade	Área de Processo	Práticas Genéricas
Nível 1	GPT – Gerência de Projetos de Teste (práticas específicas GPT1 a GPT20). PET – Projeto e Execução de Teste (práticas específicas PET1 a PET4).	PG1 a PG6
Nível 2	GRT – Gerência de Requisitos de Teste (práticas específicas GRT1 a GRT5). GPT – Gerência de Projetos de Teste (práticas específicas GPT21 a GPT25) PET – Projeto e Execução de Teste (práticas específicas PET5 e PET6).	PG7 a PG9
Nível 3	FDT– Fechamento do Teste (práticas específicas FDT1 a FDT4). GDQ – Garantia da Qualidade (práticas específicas GDQ1 a GDQ3). MAT – Medição e Análise de Teste (práticas específicas MAT1 a MAT5). OGT – Organização do Teste (práticas específicas OGT1 a OGT10). TDA – Teste de Aceitação (práticas específicas TDA1 a TDA7). TES – Teste Estático (práticas específicas TES1 a TES5). TRE – Treinamento (práticas específicas TRE1 a TRE4). GPT – Gerência de Projetos de Teste (práticas específicas GPT26 a GPT28). PET – Projeto e Execução de Teste (prática específica PET7).	
Nível 4	AQP – Avaliação da Qualidade do Produto (práticas específicas AQP1 a AQP5). GDD – Gestão de Defeitos (práticas específicas GDD1 a GDD3). TNF – Teste Não-Funcional (práticas específicas TNF1 a TNF3). OGT – Organização do Teste (práticas específicas OGT11 e OGT12).	
Nível 5	AET – Automação da Execução do Teste (práticas específicas AET1 a AET6). CEP – Controle Estatístico do Processo (práticas específicas CEP1 a CEP5). GDF – Gestão de Ferramentas (práticas específicas GDF1 a GDF6).	

Fonte: Melhoria do Processo de Teste Brasileiro (2011).

O foco deste trabalho será o nível 1, na área de processo PET, que contempla as práticas PET1, PET2, PET3 e PET4 e serão descritas a seguir.

2.3 PROJETO E EXECUÇÃO DE TESTE – PET

Conforme o modelo de Melhoria do Processo de Teste Brasileiro (2011), o objetivo dessa área de processo de Projeto e Execução de Teste é identificar, elaborar e executar casos

de teste, realizar o registro da execução do teste e das divergências encontradas entre os resultados atuais e esperados na forma de incidentes.

Em uma organização de baixa maturidade em teste, as práticas apresentadas nesta área do processo propõem garantir que os testes estão sendo executados corretamente. Ou seja, para o nível 1 de maturidade não existe ainda a preocupação com o uso adequado da documentação e sim a garantia de que as atividades essenciais estão sendo cumpridas. Esta área de processo envolve (MELHORIA DO PROCESSO DE TESTE BRASILEIRO, 2011):

a) identificar os casos de teste (PET1);
b) executar os casos de teste (PET2);
c) reportar incidentes (PET3);
d) acompanhar incidentes (PET4);
e) estabelecer padrões de documentação de casos de teste (PET5);
f) estabelecer padrões de documentação de incidentes (PET6);
g) aplicar técnicas de projeto (design) de teste (PET7).

2.3.1 PET1 – Identificar Caso de Teste

O objetivo desta prática é identificar, priorizar e documentar os casos de teste. Deve-se saber o que está sendo verificado, as entradas, procedimentos e resultados que devem ser gerados e como deve ser executado o teste. Estas informações fazem parte do caso de teste (MELHORIA DO PROCESSO DE TESTE BRASILEIRO, 2011).

Para projetar um caso de teste deve-se encontrar o melhor compromisso entre (MELHORIA DO PROCESSO DE TESTE BRASILEIRO, 2011):

a) efetividade: possuir uma probabilidade razoável de encontrar erros;
b) exemplaridade: ser prático e possuir baixo nível de redundância;
c) economia: possuir um custo de desenvolvimento razoável e retorno de investimento;
d) evolução: ser flexível, estruturado e possuir fácil manutenção.

A documentação de um caso de teste deve possuir no mínimo (MELHORIA DO PROCESSO DE TESTE BRASILEIRO, 2011):

a) identificador único;
b) objetivo;
c) condições de teste que a serem observadas.

2.3.2 PET2 – Executar Caso de Teste

O objetivo desta prática é executar casos de teste identificados e registrar as informações de execução no log de teste. Para cada caso de teste executado, as entradas devem ser fornecidas para o sistema e os resultados gerados comparados com os resultados esperados descritos no caso de teste. Devem ser registradas as informações que compõe o log do teste durante a execução do caso de teste, que podem incluir (MELHORIA DO PROCESSO DE TESTE BRASILEIRO, 2011):

a) identificador do caso de teste executado;
b) resultado da execução do caso de teste;
c) identificadores de incidentes gerados a partir da execução do caso de teste;
d) autor da execução do teste;
e) data/hora e duração da execução do teste relacionada.

2.3.3 PET3 – Reportar Incidentes

O objetivo desta prática é garantir que as divergências de comportamento apresentadas na aplicação sejam reportadas na forma de incidentes. Um incidente é qualquer evento significante não planejado observado durante o teste. No modelo MPT.BR incidentes e anomalias são sinônimos.

Todos os incidentes devem ser registrados ou reportados. Quanto mais detalhado e uniforme for o registro dos incidentes, melhor serão as possibilidades para tomar as decisões corretas no ciclo de vida da gerência de incidentes e melhor será a análise das informações sobre produtos e processos.

Segundo o modelo, o registro de um incidente é composto das seguintes informações (MELHORIA DO PROCESSO DE TESTE BRASILEIRO, 2011):

a) identificador de incidente;
b) sumário;
c) descrição do incidente;
d) impacto.

2.3.4 PET4 – Acompanhar Incidentes

O objetivo desta prática é garantir que todos os incidentes sejam analisados e acompanhados até seu fechamento. Após seu registro, o incidente deve passar por uma análise detalhada junto ao grupo de controle da configuração (GCC), que possui a responsabilidade de analisar o incidente, identificar o impacto e atribuir uma prioridade e criticidade. O GCC

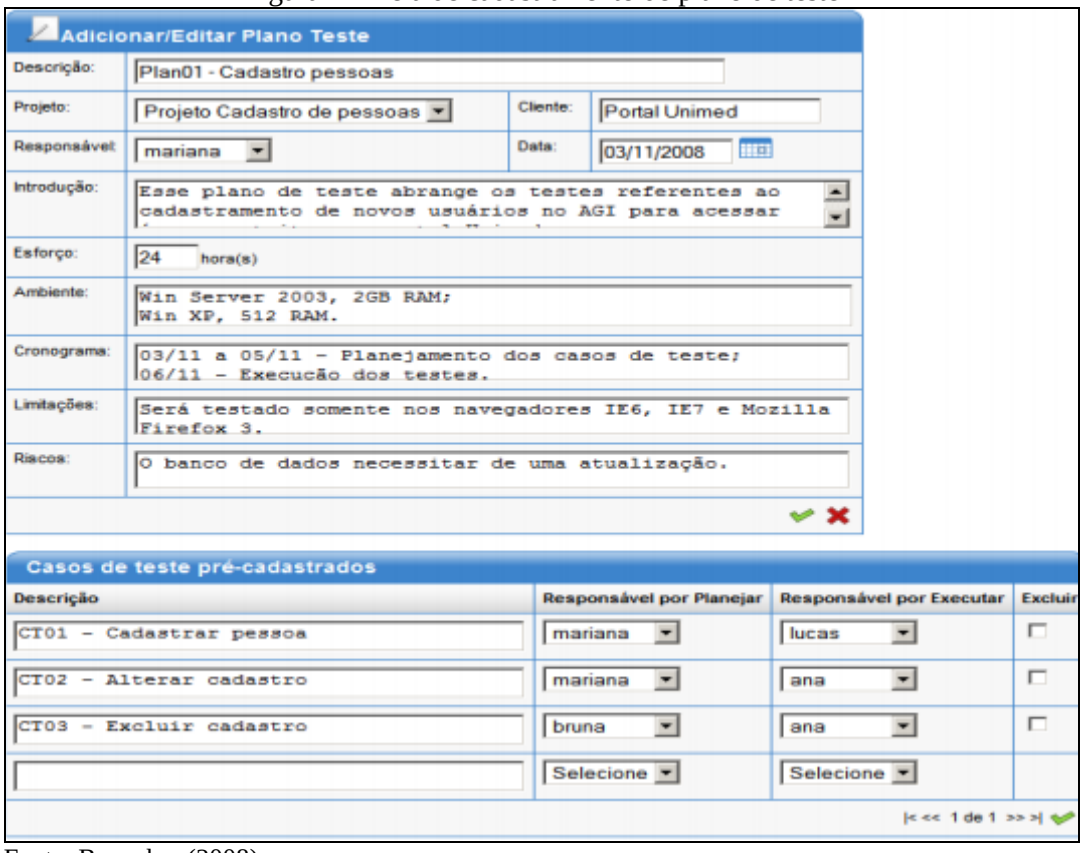
representa um grupo responsável por avaliar e aprovar mudanças propostas, assim como garantir que as mudanças aprovadas sejam implementadas de forma apropriada (MELHORIA DO PROCESSO DE TESTE BRASILEIRO, 2011).

2.4 TRABALHOS CORRELATOS

São descritos a seguir três trabalhos de conclusão de curso desenvolvidos na Universidade Regional de Blumenau (FURB).

O trabalho de conclusão de curso de Bonecher (2008) apresenta uma ferramenta web de apoio ao planejamento e controle dos testes de integração, sistema e regressão utilizando o processo OpenUP. Esta ferramenta foi desenvolvida em Java Server Page (JSP) com conexão ao banco de dados MySQL. A Figura 2 representa a tela de cadastramento do plano de teste.

Figura 2 – Tela de cadastramento de plano de teste



Fonte: Bonecher (2008).

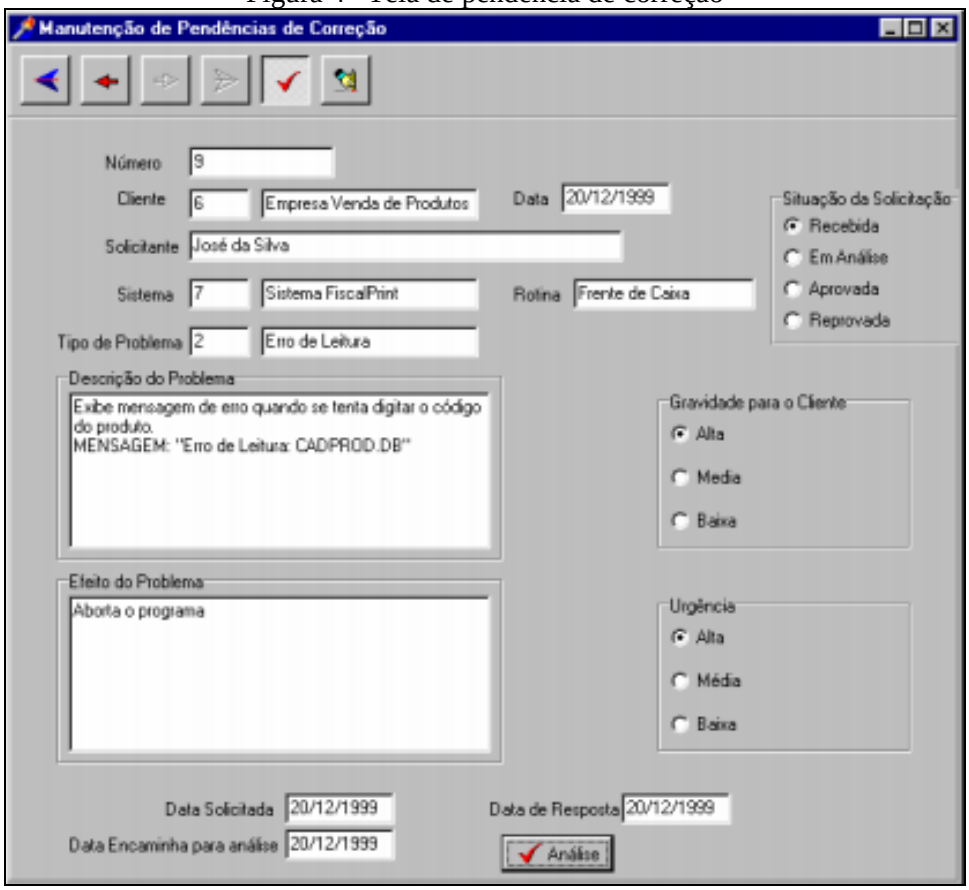
O trabalho de conclusão de curso de Depiné (2005) apresenta uma ferramenta de suporte ao gerenciamento de avaliações da qualidade de sistemas baseado na norma ISO/IEC 9126, ISO/IEC 14598 e NBR ISO/IEC 12119. Para a implementação foi utilizado o ambiente de desenvolvimento Delphi 7 e o banco de dados Interbase 6. A Figura 3 representa a tela de cadastro de plano de teste.

Figura 3 – Cadastro de plano de teste

Fonte: Depiné (2005).

O trabalho de conclusão de curso de Hoppe (1999) apresenta uma ferramenta de apoio a manutenção de sistemas baseado nas normas ISO/IEC 12207, ISO 9000-3 e SPICE. Na sua implementação foi utilizada a linguagem de programação Delphi 3.0. Na Figura 4 é apresentada a tela de pendência de correção contendo a descrição do problema, efeito do problema, cliente, gravidade, etc.

Figura 4– Tela de pendência de correção



Fonte: Hoppe (1999).

Todos estes trabalhos auxiliaram de alguma forma na concepção do sistema desde identificação de atributos, processos de teste e outras atividades correlatas, assim como atendimento a normas e padrões. O Quadro 2 representa a comparação entre os trabalhos correlatos.

Quadro 2 – Comparação entre trabalhos correlatos

	Bonecher (2008)	Depiné (2005)	Hoppe (1999)
Plataforma	Web	Desktop	Desktop
Linguagem	Java/ JSP	Delphi	Delphi
Banco de Dados	MySQL	Interbase	Baseado nos arquivos Paradox
Norma/Padrão/ Modelo de referência	OpenUP e IEEE-829	ISO/IEC 9126, ISO/IEC 14598 e NBR/IEC 12119	ISO/IEC 12207, ISO 9000-3 e SPICE
Permite cadastrar caso de teste?	Sim	Sim	Não
Permite cadastrar incidentes?	Sim	Sim	Sim
Permite cadastrar plano de teste?	Sim	Sim	Não

Fonte: Elaborada pela autora.

3 DESENVOLVIMENTO

Neste capítulo estão descritas as particularidades do sistema desenvolvido, tais como a descrição do levantamento das informações, apresentação dos requisitos funcionais e não funcionais, diagrama de caso de uso, diagrama de classes, diagrama de entidade e relacionamento, as técnicas e ferramentas utilizadas, a operacionalidade da implementação e resultados e discussões.

3.1 LEVANTAMENTO DE INFORMAÇÕES

Desenvolveu-se um sistema web para otimizar os processos no setor de qualidade em uma empresa de software utilizando o modelo MPT.BR baseado na área de processo Projeto e Execução de Teste (PET). A partir da experiência profissional da autora com os estudos de trabalhos correlatos e o próprio modelo MPT.BR foi criada uma solução básica para processo de teste de software.

O sistema permite o testador cadastrar casos de teste, informar os resultados obtidos em sua execução, reportar incidentes, cadastrar a análise dos incidentes e a tomada de decisão através do grupo de controle da configuração (GCC). A autenticação do usuário é feita através de um *login* e senha previamente cadastrados pelo administrador do sistema. O cadastro do projeto é responsabilidade do analista de teste, para assim vincular os casos de teste os casos de teste as execuções, incidentes e o acompanhamento dos incidentes.

Na construção do sistema foram utilizadas a ferramenta Django, que é um *framework* web baseado na linguagem de programação Python e o PostgreSQL para armazenamento das informações.

3.2 ESPECIFICAÇÃO

Nesta seção são apresentados os requisitos funcionais, requisitos não funcionais, diagrama de caso de uso, diagramas de atividades, e o modelo de entidade e relacionamento do sistema desenvolvido.

3.2.1 Requisitos Funcionais

O Quadro 3 apresenta os requisitos funcionais previstos para o sistema e sua rastreabilidade, ou seja, vinculação com os casos de uso.

Quadro 3 - Requisitos funcionais

Requisitos Funcionais	Caso de Uso
RF01: O sistema deverá permitir ao gestor realizar cadastramento de usuários.	UC01
RF02: O sistema deverá permitir o testador efetuar o *login* no sistema.	UC02
RF03: O sistema deverá permitir o analista de teste realizar cadastros de projetos.	UC03
RF04: O sistema deverá permitir o analista de teste cadastrar os casos de teste.	UC04
RF05: O sistema deve permitir o testador visualizar os casos de teste.	UC05
RF06: O sistema deverá permitir o testador informar o resultado do teste executado.	UC06
RF07: O sistema deverá permitir o testador cadastrar incidentes.	UC07
RF08: O sistema deverá permitir o GCC cadastrar tomada de decisão.	UC08
RF09: O sistema deverá permitir o gestor gerar relatório de detalhamento dos incidentes registrados.	UC09
RF10: O sistema deverá permitir o gestor gerar relatório dos testes executados.	UC10
RF11: O sistema deverá permitir o gestor gerar relatório de média de incidentes e aprovações por mês.	UC11

Fonte: Elaborada pela autora.

3.2.2 Requisitos não Funcionais

O Quadro 4 lista os requisitos não funcionais previstos para o sistema.

Quadro 4 - Requisitos não funcionais

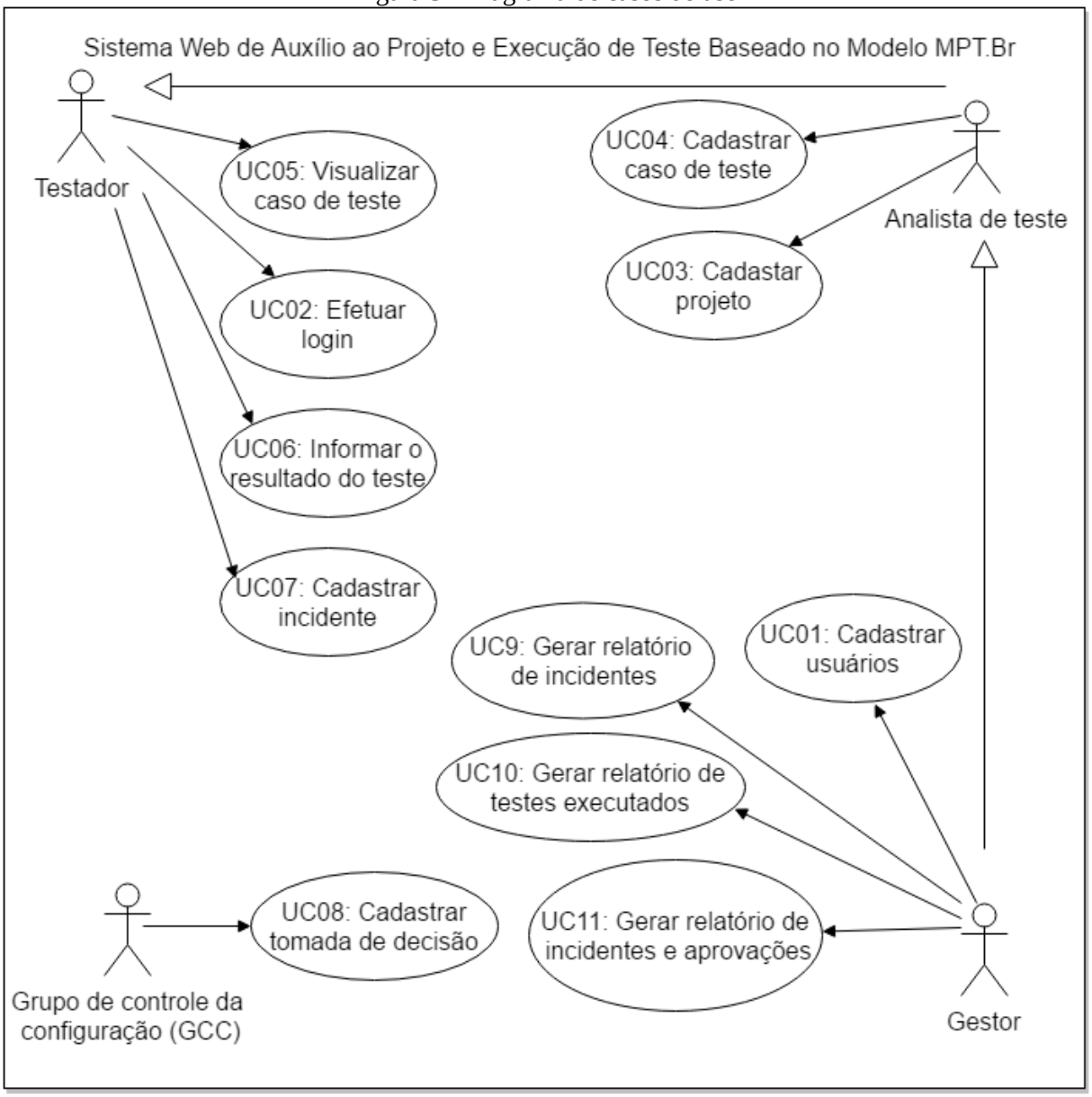
Requisitos Não Funcionais
RNF01: O sistema deverá utilizar o banco de dados PostgreSQL.
RNF02: O sistema deverá ser acessível via Google Chrome 34 ou superior, Mozilla Firefox 6.0 ou superior e Internet Explorer 9 ou superior.
RNF03: O sistema deverá ser desenvolvido para plataforma web.
RNF03: O sistema será implementado com a linguagem Python utilizando o *framework* Django versão 1.9.

Fonte: Elaborada pela autora.

3.2.3 Casos de Uso

Nesta subseção é apresentado o diagrama de caso de uso (UC) do sistema. Na Figura 5 são apresentadas as funcionalidades que os atores `testador`, `analista de teste`, `gestor` e `grupo de controle da configuração (GCC)` podem realizar no sistema.

Figura 5 - Diagrama de casos de uso



Fonte: Elaborada pela autora.

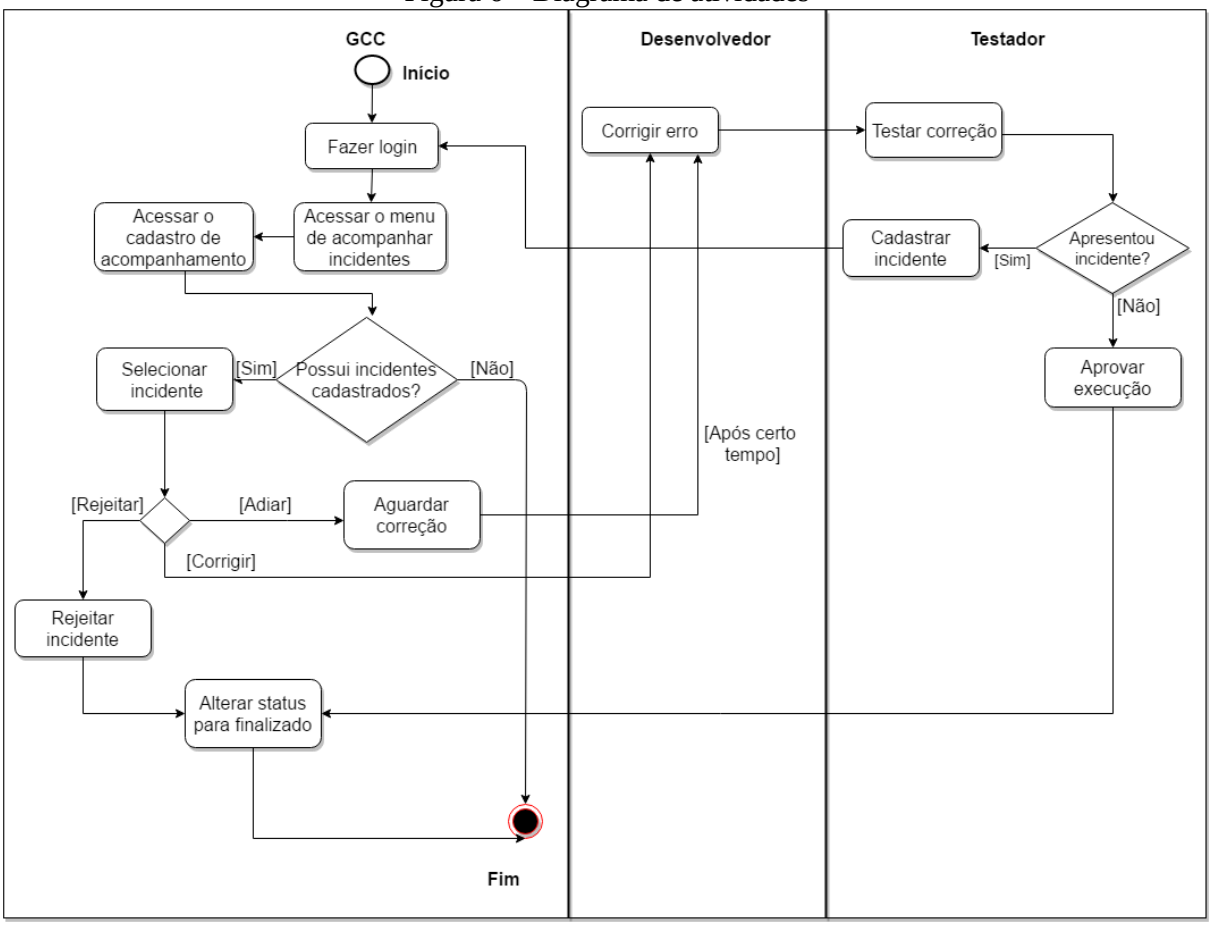
Conforme Figura 5 o `Testador` é responsável por cadastrar incidentes e informar os resultados encontrados na execução dos casos de teste, o ator do `Grupo de controle da configuração (GCC)` deve informar qual medida será tomada em cada incidente apontado pelo `Testador`.

O `analista de teste` possui todas as funcionalidades do `testador` e a responsabilidade de cadastrar os casos de teste e os projetos. O `gestor` representando o último ator, além de possuir todas as funcionalidades do analista de teste é responsável pelo cadastro de usuários e visualização dos relatórios gerados com base nas informações gravadas no sistema. No Apêndice A é possível visualizar o detalhamento dos casos de uso.

3.2.4 Diagrama de Atividades

O diagrama de atividades representado na Figura 6 demonstra o fluxo do processo de Acompanhar incidentes no sistema.

Figura 6 – Diagrama de atividades



Fonte: Elaborada pela autora.

A partir da Figura 6 pode-se verificar que para acessar a rotina de acompanhar incidentes o GGC deve realizar o *login* no sistema. Após realizar o *login* deve-se acessar o menu de acompanhar incidentes e a opção de cadastrar incidente, caso não exista incidente a operação é finalizada.

Para incidentes cadastrados deve-se selecionar o incidente e escolher uma tomada de decisão, ao rejeitar um incidente e alterar o status do acompanhamento para finalizado a operação é finalizada. Ao adiar uma correção o incidente fica aguardando a correção ser realizada e após certo tempo para ser retomada. Caso a decisão for corrigir o incidente será reportado para a área de desenvolvimento, após corrigido é encaminhado ao testador para realizar os testes.

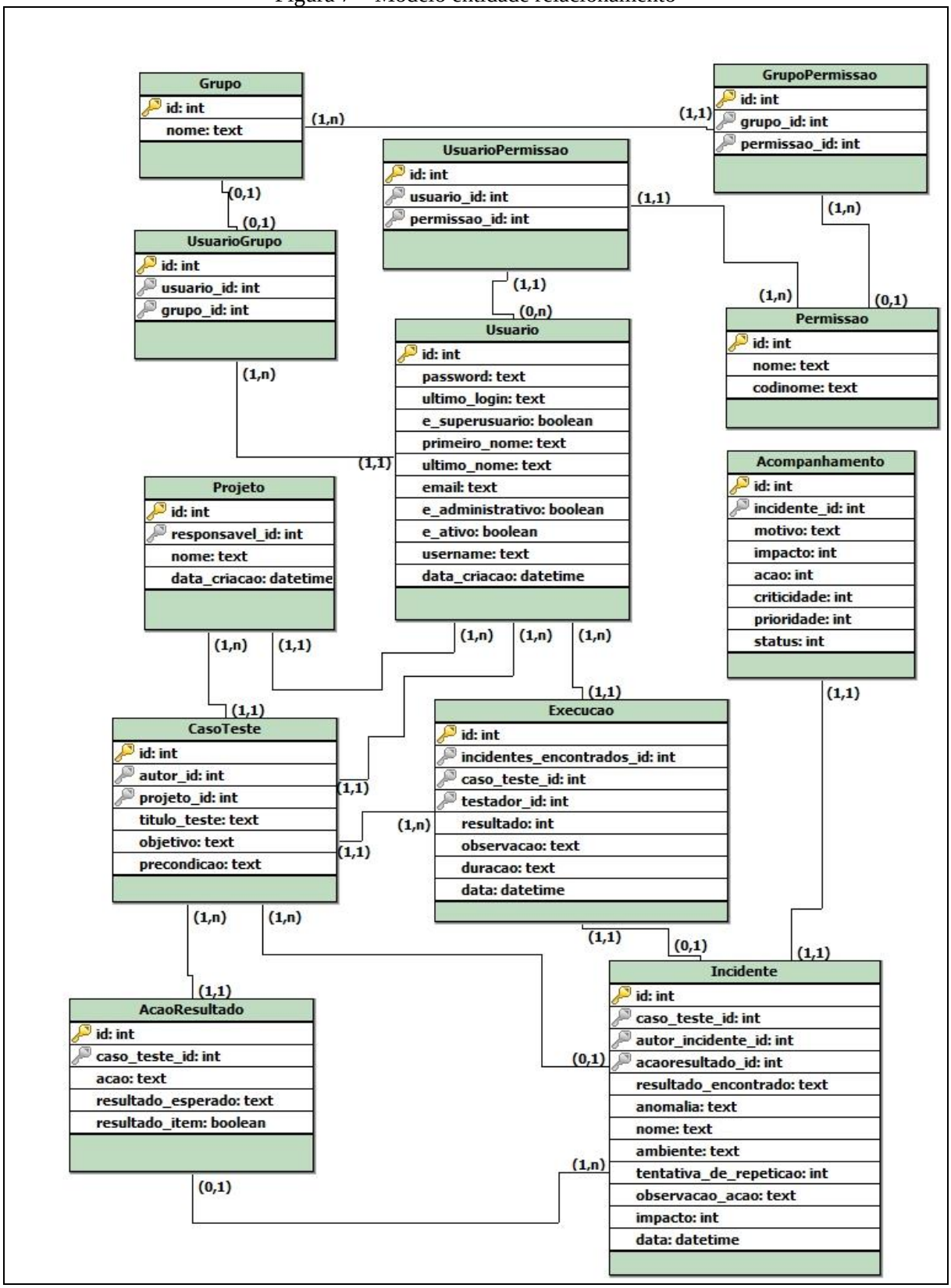
Se nos testes forem encontrados incidentes, estes serão cadastrados e a rotina de acompanhar incidentes voltará para o início, caso não ocorra problema no teste e for

aprovado, o status do acompanhamento deve ser alterado para finalizado e a operação é finalizada.

3.2.5 Modelo Entidade Relacionamento

A Figura 7 representa o Modelo Entidade Relacionamento (MER) da ferramenta desenvolvida. As entidades de autenticação e controle de usuários são geradas pelo próprio *framework* Django e o restante de entidades foram criadas para a gravação dos dados das informações populadas no sistema. O dicionário de dados das entidades está disponível no Apêndice B.

Figura 7 – Modelo entidade relacionamento



Fonte: Elaborada pela autora.

3.3 IMPLEMENTAÇÃO

A seguir são mostradas as técnicas e ferramentas utilizadas, assim como a operacionalidade da implementação.

3.3.1 Técnicas e ferramentas utilizadas

A aplicação foi desenvolvida na linguagem de programação Python através do *framework* Django, utilizando o Integrated Development Environment (IDE) Sublime Text 2 para edição do código fonte. O banco de dados escolhido foi o PostgreSQL, um Sistema Gerenciador de Banco de Dados Relacional (SGBDR) que suporta o armazenamento de grandes objetos binários e é compatível com os principais sistemas operacionais.

3.3.1.1 *Framework* web Django

O Django é um *framework* de alto nível que estimula o desenvolvimento rápido e design limpo baseado na linguagem de programação Python. Utiliza a arquitetura de software conhecido como Model Template View (MTV) redefinindo o padrão tradicional Model View Controller (MVC). M corresponde a Model e representa a camada de abstração de dados, esta camada contém tudo sobre dados, regra de negócio lógica e funções. T corresponde a Template e representa as páginas HyperText Markup Language (HTML), ou seja, quando algo deve ser mostrado em uma página web e como deve ser apresentado. V corresponde a View e representa a camada lógica de negócios, esta camada contém a lógica que acessa os dados do Model e submete a apresentação na Template apropriada (SILVA, 2015).

Cada módulo do projeto trabalha com uma pasta específica e possui sua estrutura do código fonte de acordo com a estrutura recomendada pelo DjangoProjects. Utilizou-se o arquivo `models.py` para a definição dos dados, o arquivo `views.py` para a regra de negócio com a saída para um template HTML (DJANGOPROJECTS, 2013). O Quadro 5 apresenta parte do código fonte do arquivo `models.py` utilizado para o desenvolvimento do sistema.

Quadro 5 – Código fonte do arquivo models.py

```
from django.db import models
from django.utils import timezone
from django.contrib.auth.models import User
from django.core.exceptions import ValidationError
import datetime

class Projeto(models.Model):
    nome=models.CharField(max_length=200, verbose_name='Projeto')
    data_criacao=models.DateTimeField(verbose_name= 'Data')
    responsavel=models.ForeignKey(User, related_name='responsavel', null=True, blank=True,
        verbose_name='Responsável')

    class Meta:
        app_label='projeto'
        verbose_name_plural='Projeto'

    def __str__(self):
        return self.nome


class CasoTeste(models.Model):
    autor=models.ForeignKey(User, related_name='autor', null=True, blank=True, verbose_name='Autor')
    projeto=models.ForeignKey(Projeto)
    titulo_teste=models.CharField(max_length=100, verbose_name='Título')
    objetivo=models.TextField()
    precondicao=models.CharField(max_length=200, verbose_name='Pré-condição', null=True)

    class Meta:
        app_label='projeto'
        verbose_name_plural='Caso de teste'

    def __str__(self):
        return self.titulo_teste
```

Fonte: Elaborada pela autora.

No desenvolvimento das telas foi utilizada biblioteca de administração automática do Django que é responsável pelo tratamento da camada de visualização, onde realiza a leitura dos metadados dos modelos configurados e gera uma interface rápida ao usuário com base nas regras configuradas no arquivo `admin.py`. As telas de cadastro de execução, cadastro de incidente e relatórios necessitaram de customizações específicas, para isso os *templates* padrões foram copiados para a pasta do projeto, permitindo assim a realização das customizações necessárias utilizando arquivo `forms.py` para as regras específicas dos formulários (DJANGOPROJECTS, 2013).

3.3.1.2 PostgreSQL

O PostgreSQL é um Sistema Gerenciador de Banco de Dados Relacional (SGBDR) que possui mais de 15 anos de desenvolvimento e uma arquitetura que ganhou forte reputação de confiabilidade, integridade de dados e correção. Compatível com os principais sistemas operacionais e suporte o armazenamento de grandes objetos binários, incluindo imagens, sons e vídeo (POSTGRESQL, 2016).

Segundo Milani (2008) o PostgreSQL pode ser livremente incorporado sem nenhum custo para o desenvolvedor ou o fornecedor do software. Muitas micro e pequenas empresas tem optado por migrar suas plataformas de desenvolvimento para o software livre da mesma forma que organizações de médio e grande porte. Deste modo é possível conciliar o alto poder

de processamento do PostgreSQL e seu custo inexistente de licenças para desenvolvimento de software.

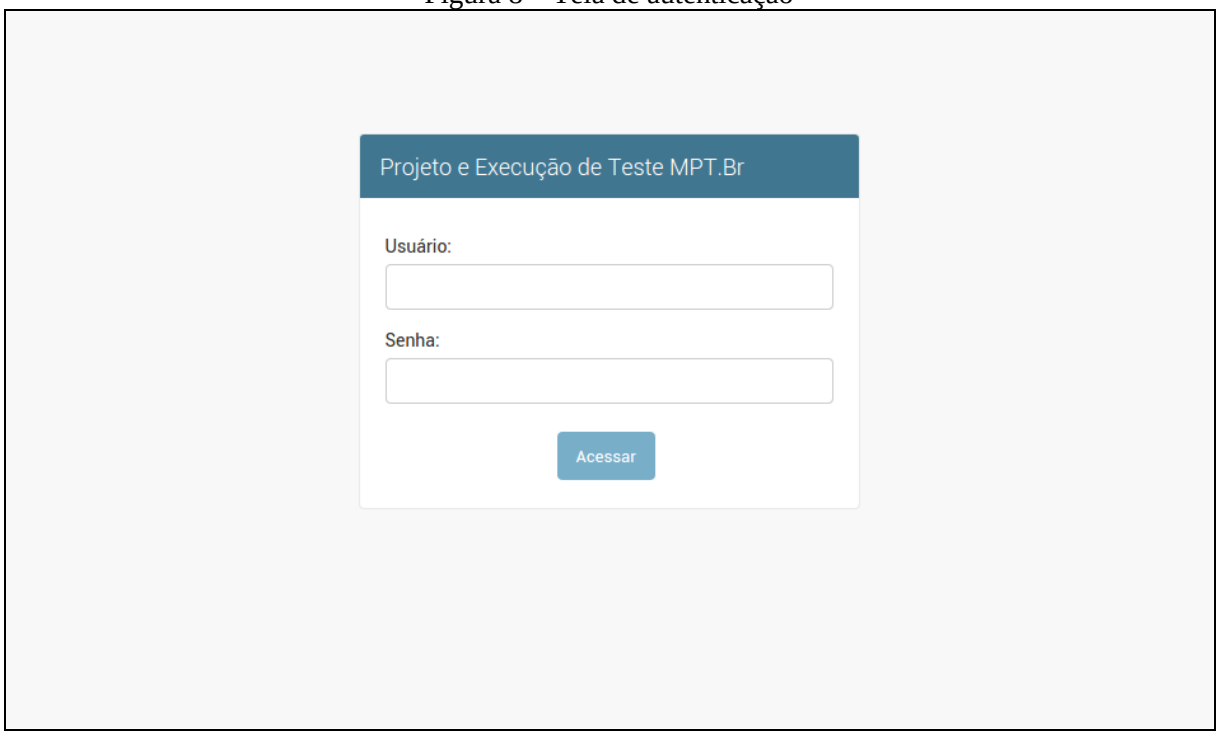
Para os desenvolvedores existem bibliotecas e drives de conexão para o PostgreSQL com as principais linguagens e plataformas, dentre elas: PHP, Python, Ruby, Java, Tcl e C. Apresenta suporte nativo a Secure Socket Layer (SSL) já embutido no PostgreSQL, criando conexões seguras para o tráfego de informações de *login* e informações sigilosas (MILANI, 2008).

3.3.2 Operacionalidade da Implementação

Nesta subseção apresentam-se as principais telas do sistema com uma apresentação sobre suas funcionalidades e trechos de código fonte relevantes para o entendimento das rotinas.

Ao abrir o sistema tem-se a tela de autenticação, onde o usuário que já possui cadastro deve informar seu usuário e senha para prosseguir. A Figura 8 contém a tela inicial do sistema.

Figura 8 – Tela de autenticação



Fonte: Elaborada pela autora.

Após efetuar a autenticação o gestor será redirecionado a tela principal do sistema, onde terá acesso a todos os menus, conforme Figura 9.

Figura 9 – Tela principal do gestor

Fonte: Elaborada pela autora.

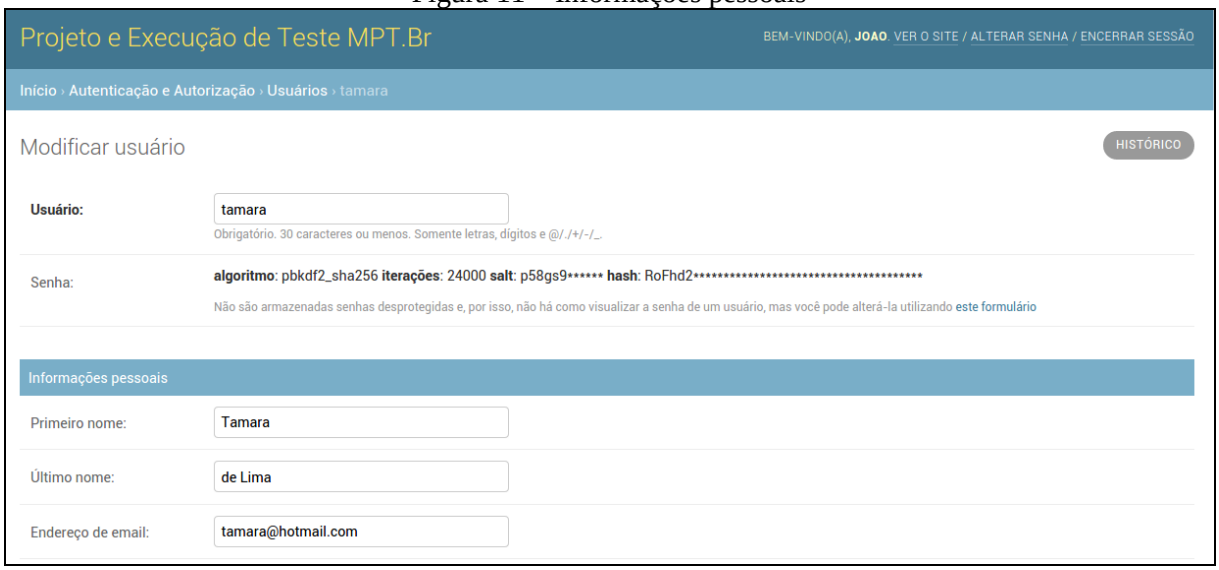
Na Figura 9 pode-se observar que a tela está subdividida em quatro áreas. Logo acima `AUTENTICAÇÃO E AUTORIZAÇÃO` refere-se à configuração de permissão e cadastro de usuários, abaixo o `PROJETO` que representa todos os cadastros para o planejamento e execução dos testes e ao lado direito a demonstração das `Ações Recentes` do usuário no sistema. A Figura 10 apresenta a tela de cadastro de usuário.

Figura 10– Tela de cadastro de usuário

Fonte: Elaborada pela autora.

Após informar os campos e salvar conforme Figura 10, o usuário será direcionado para a página que permite adicionar informações pessoais e permissões. Na Figura 11 é possível visualizar o cadastro das informações pessoais.

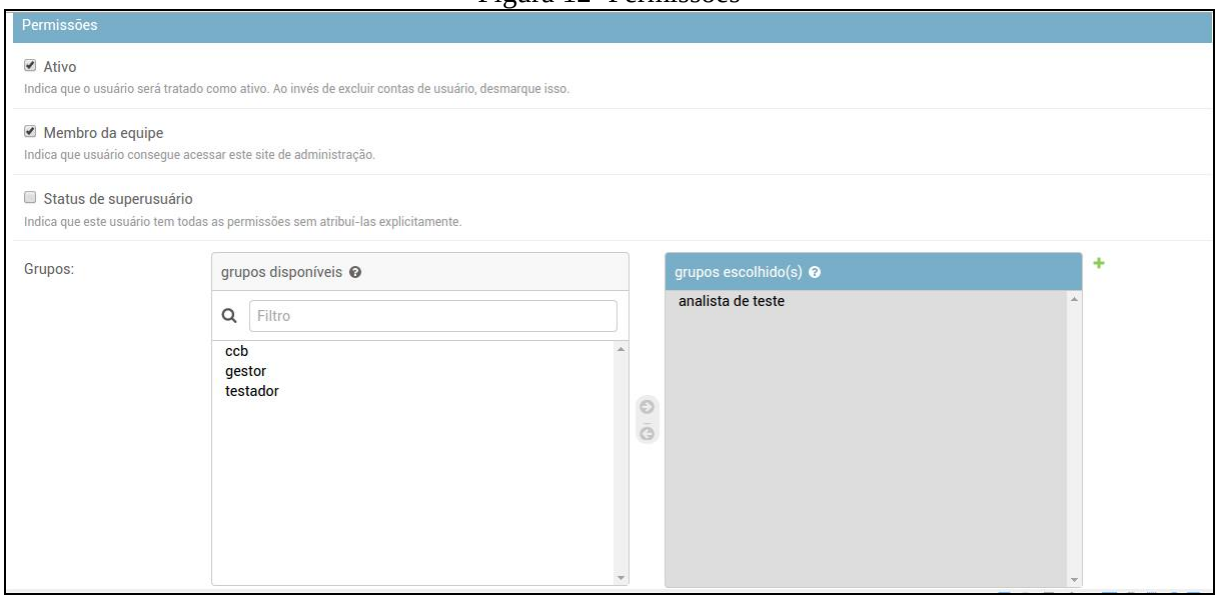
Figura 11 – Informações pessoais



Fonte: Elaborada pela autora.

Na Figura 12 é possível visualizar a parte de configuração de permissão do usuário.

Figura 12- Permissões



Fonte: Elaborada pela autora.

Para cadastrar um projeto o analista de teste deve acessar a opção `Projeto` localizada na área `PROJETO`, conforme Figura 13.

Figura 13 – Tela principal analista de teste

Fonte: Elaborada pela autora.

Ao pressionar esta opção o usuário será direcionado para a página principal do projeto, conforme Figura 14.

Figura 14 – Tela principal do projeto

Fonte: Elaborada pela autora.

A Figura 14 pode ser visto na parte superior esquerda o filtro que permite pesquisar por nome do projeto e data. Após tem-se `Ação` que permite excluir os projetos e abaixo os projetos que estão cadastrados no sistema. Na parte superior direita existe a opção `ADICIONAR PROJETO +` e abaixo pesquisar um projeto através do `FILTRO` por responsável.

O Quadro 6 representa o código fonte do arquivo admin.py onde é feita a configuração dos campos que são apresentados na Figura 14.

Quadro 6 – Código fonte tela principal do projeto

```
60  class ProjetoAdmin(admin.ModelAdmin):
61
62      model=Projeto
63      list_display=('nome','data_criacao', 'responsavel')
64      list_filter=['responsavel']
65      search_fields =['nome', 'data_criacao']
66      def get_id(self, obj):
67          return mark_safe("<span>%s</span>" % obj.id)
68      get_id.short_description = "Código"
```

Fonte: Elaborada pela autora.

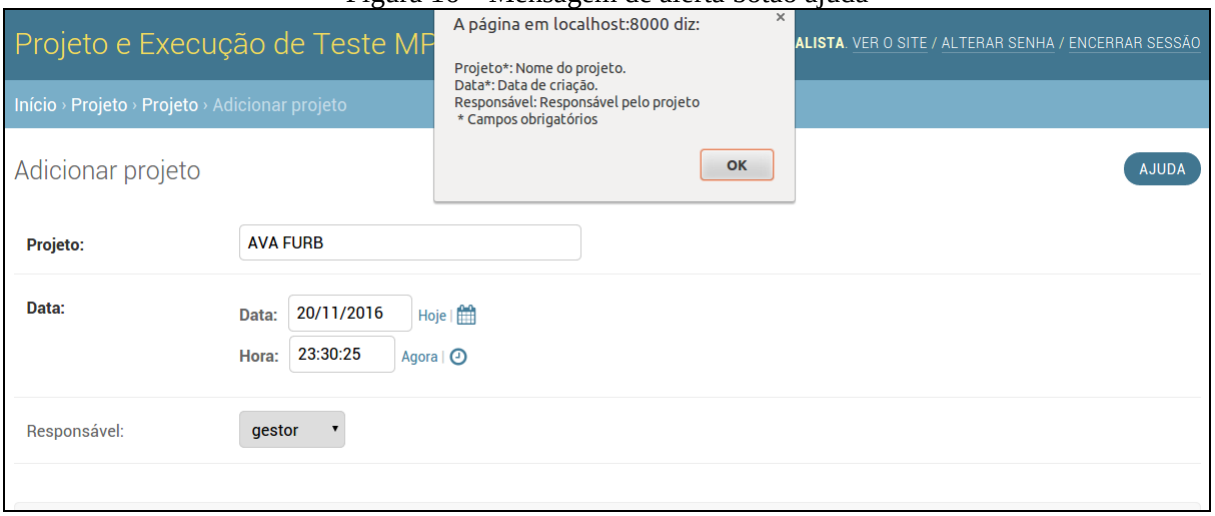
Ao selecionar a opção `ADICIONAR PROJETO +`, o analista de teste será direcionado para a tela de cadastro, conforme Figura 15.

Figura 15 – Adicionar projeto

Fonte: Elaborada pela autora.

Na Figura 15 o analista de teste deve informar os campos `Projeto` que representa o nome do projeto, `Data` que representa a data de criação e `Responsável` que representa o responsável pelo projeto. Em cada tela de cadastro do sistema existe o botão `AJUDA` que possui uma definição para cada campo do sistema auxiliando no preenchimento das informações. A Figura 16 representa a mensagem de alerta ao pressionar o botão `AJUDA`.

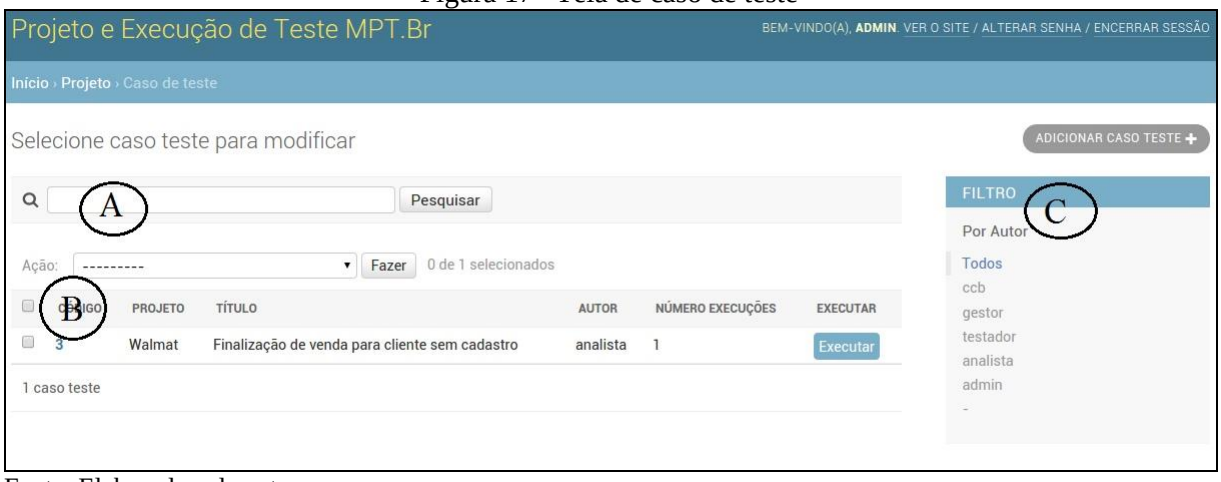
Figura 16 – Mensagem de alerta botão ajuda



Fonte: Elaborada pela autora.

Após o cadastro do projeto será cadastrado o caso de teste. Para realizar o cadastro deve-se acessar o menu `Caso Teste` apresentado na Figura 13, onde o analista de teste será direcionado para a tela de `Caso de Teste`, conforme Figura 17.

Figura 17– Tela de caso de teste

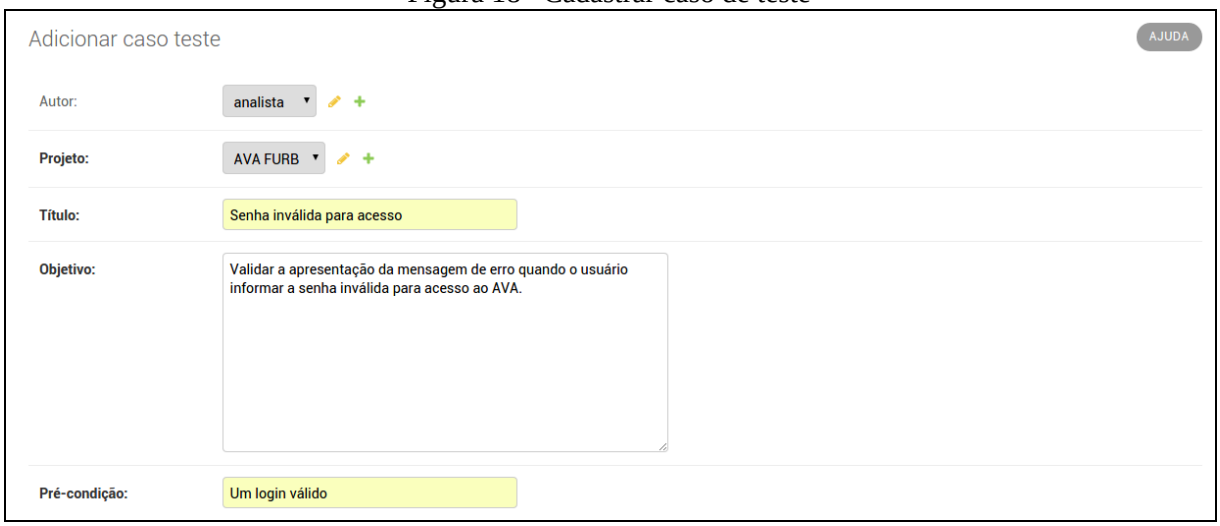


Fonte: Elaborada pela autora.

Na Figura 17 a parte superior esquerda representada pela letra `A`, apresenta o filtro que permite pesquisar por código, nome do projeto, título do caso de teste e responsável. Logo abaixo representado pela letra `B`, são apresentados os casos de testes cadastrados e o botão `Executar` para ser executado o caso de teste. Na parte superior esquerda representada pela letra `C` está a opção `ADICIONAR CASO TESTE +` e o `FILTRO` que permite filtrar os casos de teste por autor.

Para cadastrar um caso de teste o analista de teste deve selecionar a opção `ADICIONAR CASO TESTE +` onde será direcionado para a tela de cadastro, conforme a Figura 18. Nos dados de teste a fim de simplificação foram usadas informações do cenário de teste de *login* inválido do livro de Bastos et al. (2012).

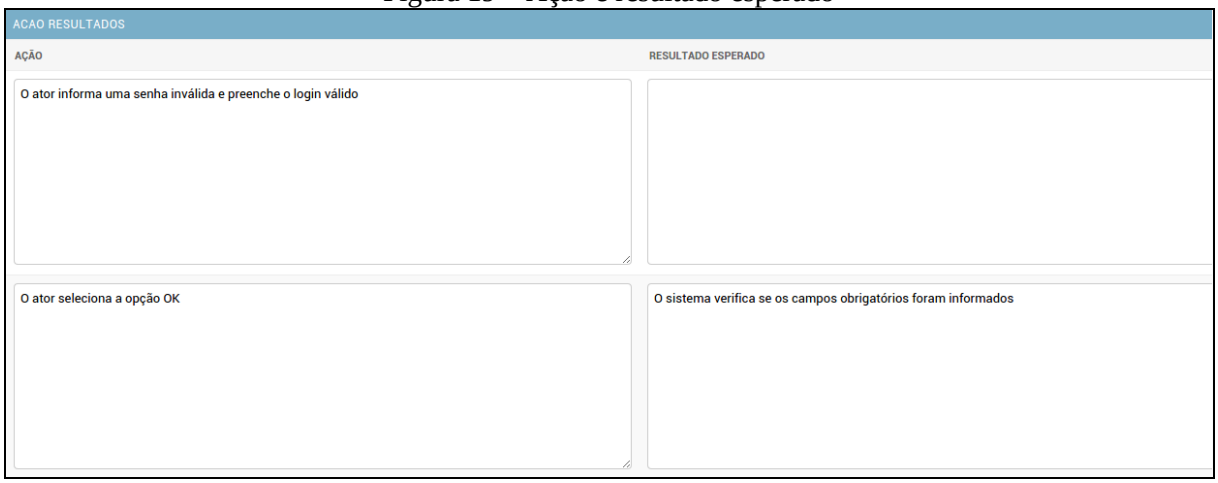
Figura 18– Cadastrar caso de teste



Fonte: Elaborada pela autora.

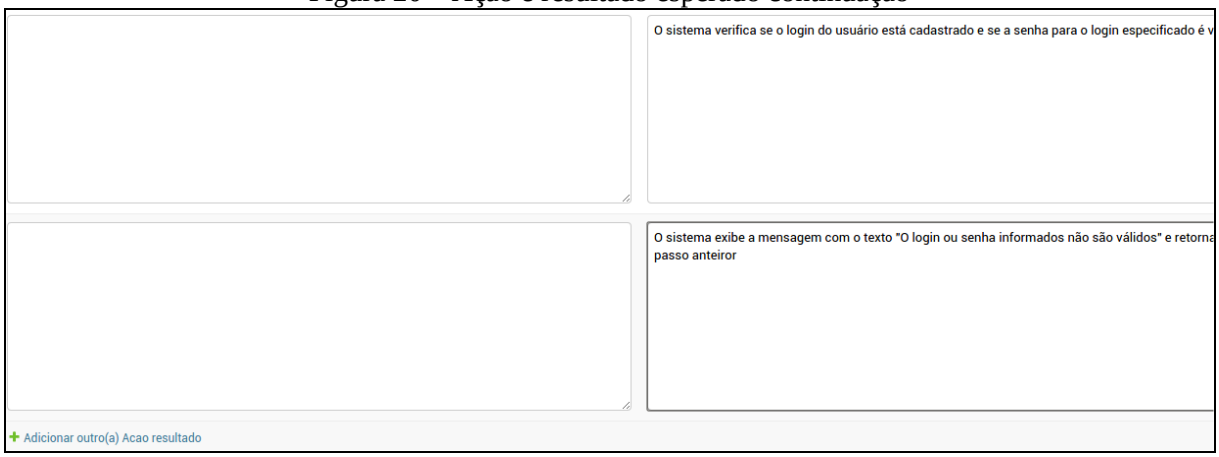
Na Figura 18 para cadastrar o caso de teste o analista de teste deve informar os campos `Autor`, `Projeto`, `Título`, `Objetivo`, `Pré-condição`, `Ação` e `Resultado esperado`. Os campos de `AÇÃO` e `RESULTADO ESPERADO` são apresentados na Figura 19 e Figura 20.

Figura 19 – Ação e resultado esperado



Fonte: Elaborada pela autora.

Figura 20 – Ação e resultado esperado continuação



Fonte: Elaborada pela autora.

Após salvar o cadastro o analista será direcionado para a página anterior, onde é possível realizar a execução do teste.

Caso a execução já tenha sido aprovada o botão de execução não será apresentado. No Quadro 7 tem-se o código fonte responsável pela apresentação do botão executar.

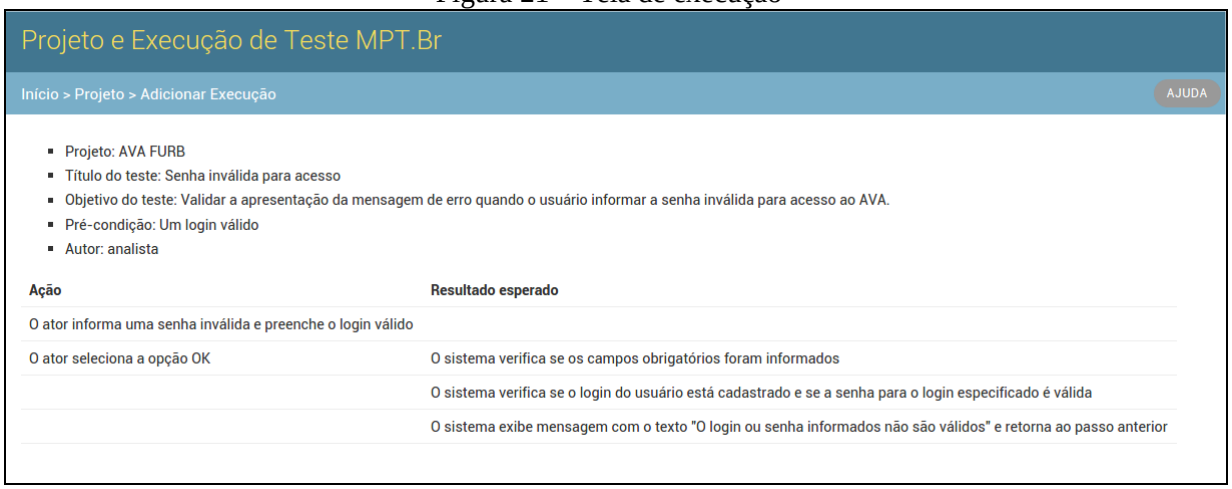
Quadro 7- Código fonte de validação do botão

```
36      def get_executar(self, obj):
37          meubotao = '<a href="/projeto/casoteste/{0}/executar/"'\
38              'class="button" type="button">Executar</a>'.format(obj.id);
39          if Execucao.objects.filter(caso_teste=obj.id, resultado='0').exists():
40              meubotao= '<button style= "display:none"></button>'
41
42          return mark_safe(meubotao)
43      get_executar.short_description= "Executar"
```

Fonte: Elaborada pela autora.

A Figura 21 apresenta as informações do caso de teste na tela de execução que se referem-se ao nome do projeto, título do teste, objetivo, pré-condição, autor, ação e resultado esperado.

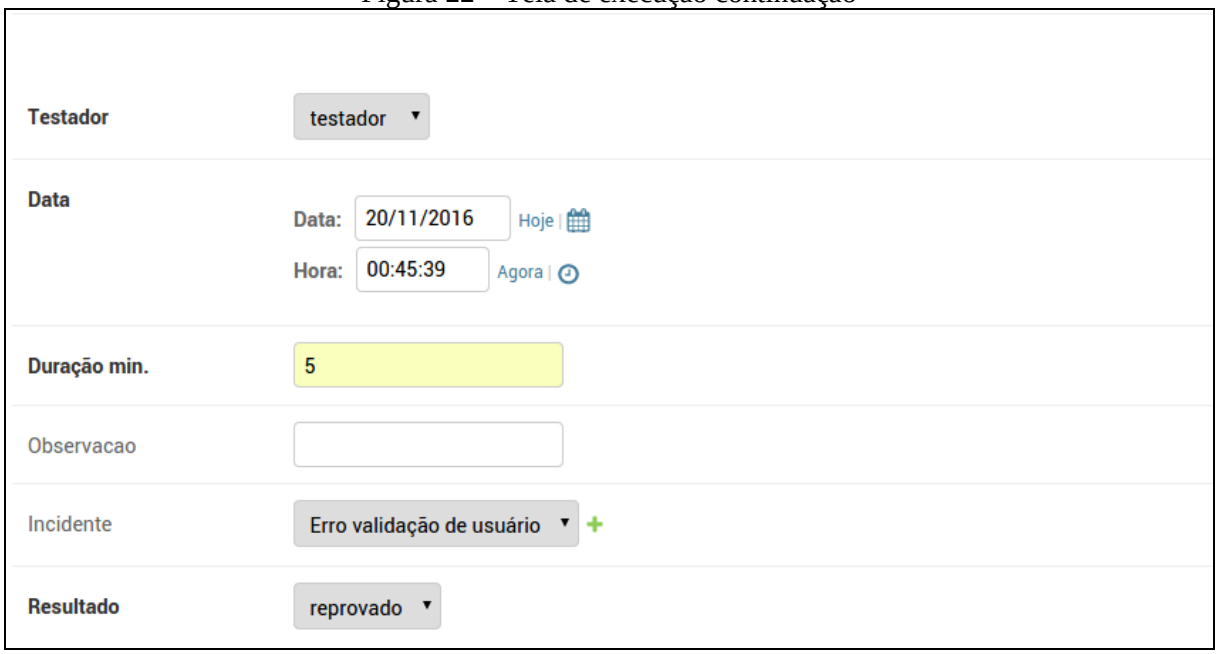
Figura 21 – Tela de execução



Fonte: Elaborada pela autora.

A Figura 22 representa as informações que devem ser preenchidas na execução.

Figura 22 – Tela de execução continuação



Fonte: Elaborada pela autora.

Na Figura 22 são apresentadas as informações a serem preenchidas, `Testador, Data, Duração min` que representa a duração em minutos, `Observação, Incidente` caso haja encontrado algum problema na execução dos passos e `Resultado` que pode ser aprovado ou reprovado. Para uma execução reprovada o cadastro do incidente é obrigatório, no Quadro 8 é possível ver o código fonte responsável por esta validação.

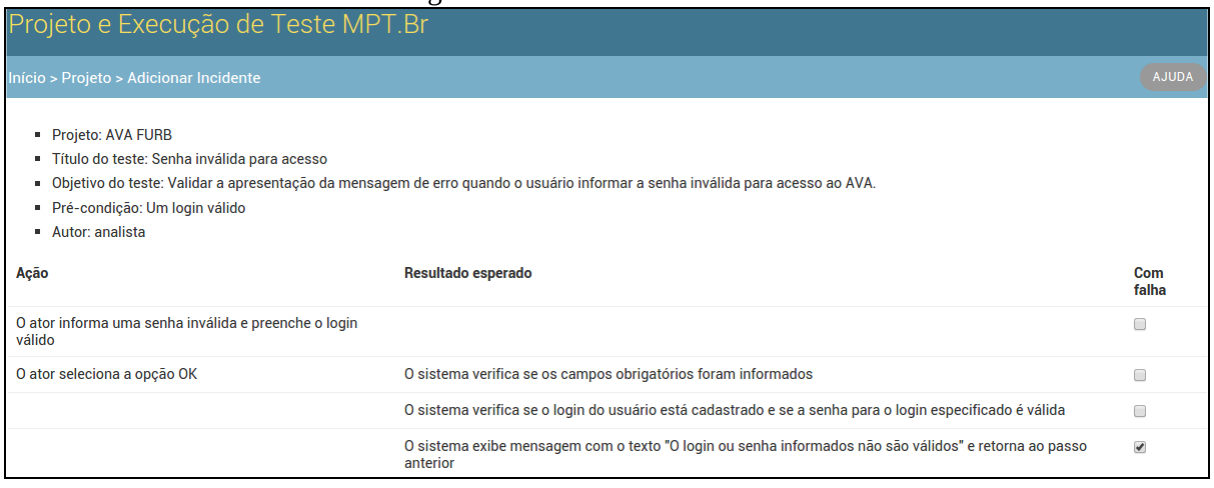
Quadro 8 – Código fonte validação incidente obrigatório

```
156     def clean(self):
157         if self.resultado==1 and self.incidentes_encontrados is None:
158             raise ValidationError("O cadastro do incidente é obrigatório")
159
```

Fonte: Elaborada pela autora.

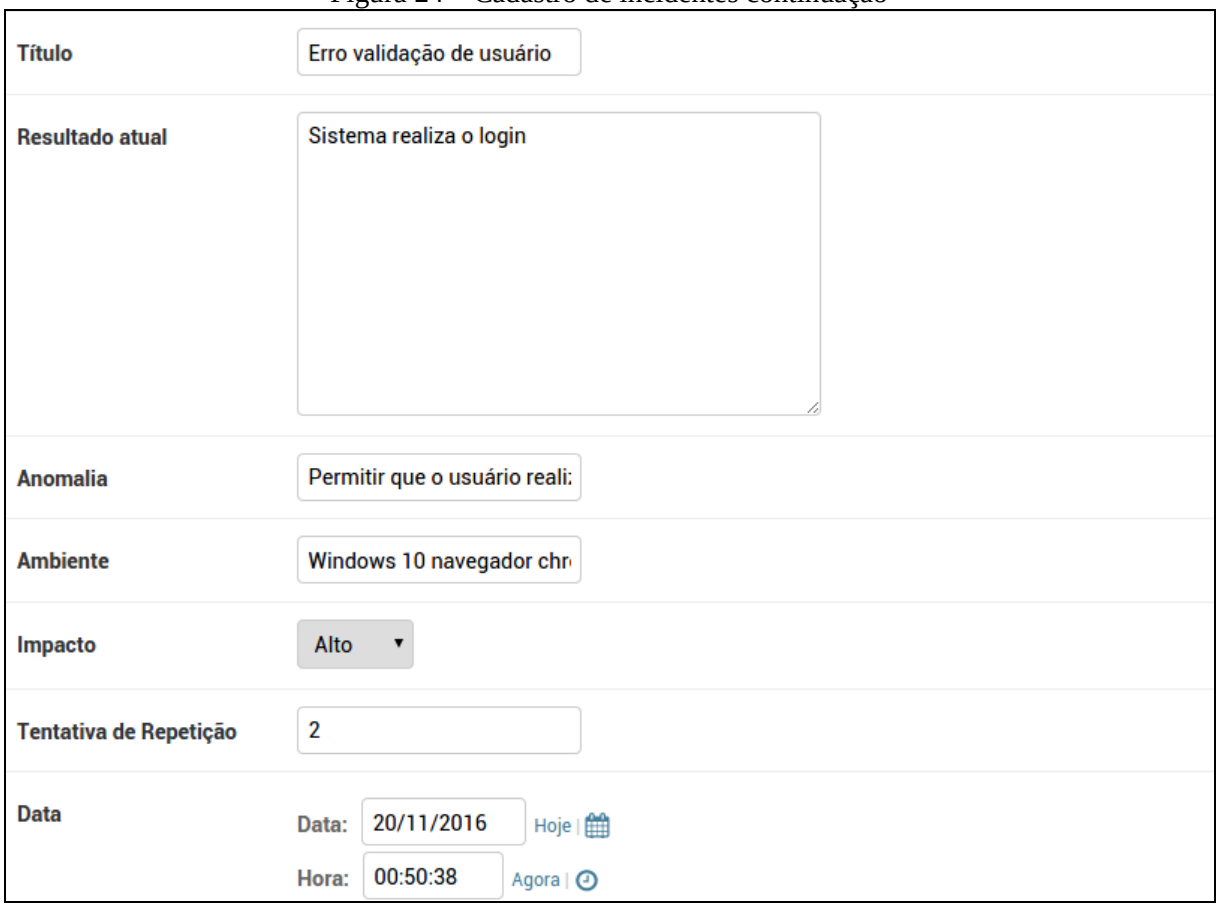
Na Figura 23 e Figura 24 são apresentadas a tela de cadastro de incidentes acessada pela tela de execução através do botão + no campo `Incidente`.

Figura 23 – Cadastro de incidentes



Fonte: Elaborada pela autora.

Figura 24 – Cadastro de incidentes continuação



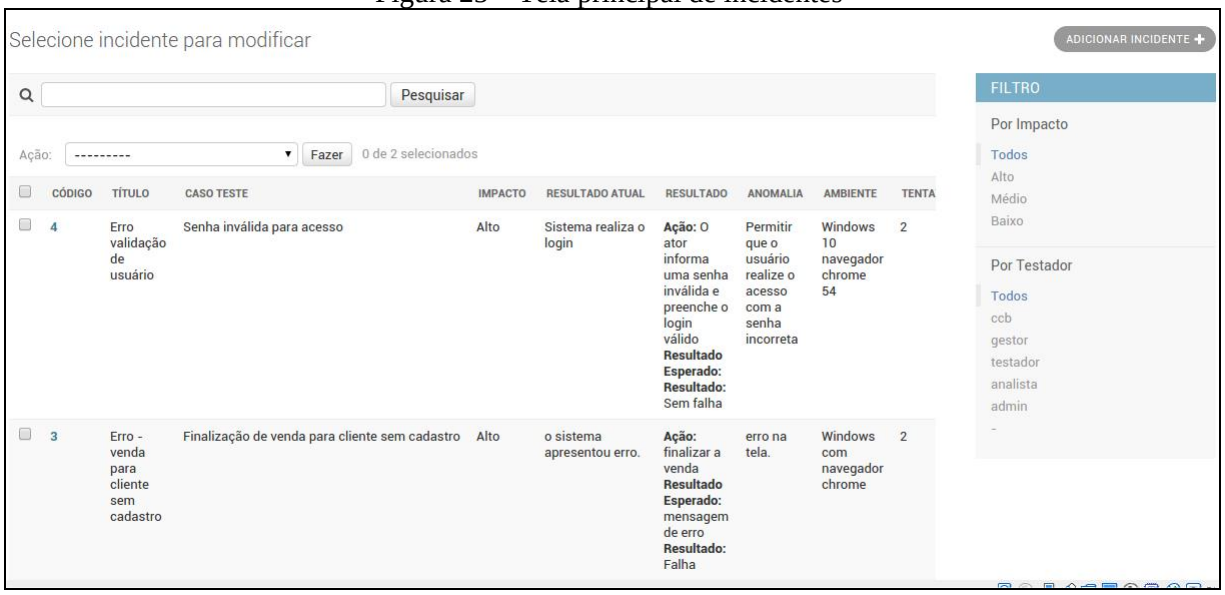
Fonte: Elaborada pela autora.

Na Figura 23 na parte superior são apresentadas informações do caso de teste, ou seja, nome do projeto, objetivo do teste, pré-condição, autor, ação e resultado esperado. Neste momento é possível selecionar o passo que ocorre o problema, apontando o que possui falha no campo `Com falha`.

O testador pode selecionar o item com falha ou apenas informar o `Resultado atual` e a `Anomalia`, seguido do `Título`, `Ambiente`, `Impacto`, `Tentativa de repetição`, `Data` e `Testador` conforme Figura 24.

O cadastro do incidente pode ser realizado em outro momento caso o caso de teste tenha sido aprovado. Este cadastro do incidente é acessado através da tela principal, ao selecionar opção `Incidente` o testador será direcionado para a tela de cadastro de incidentes conforme Figura 25.

Figura 25 – Tela principal de incidentes



Fonte: Elaborada pela autora.

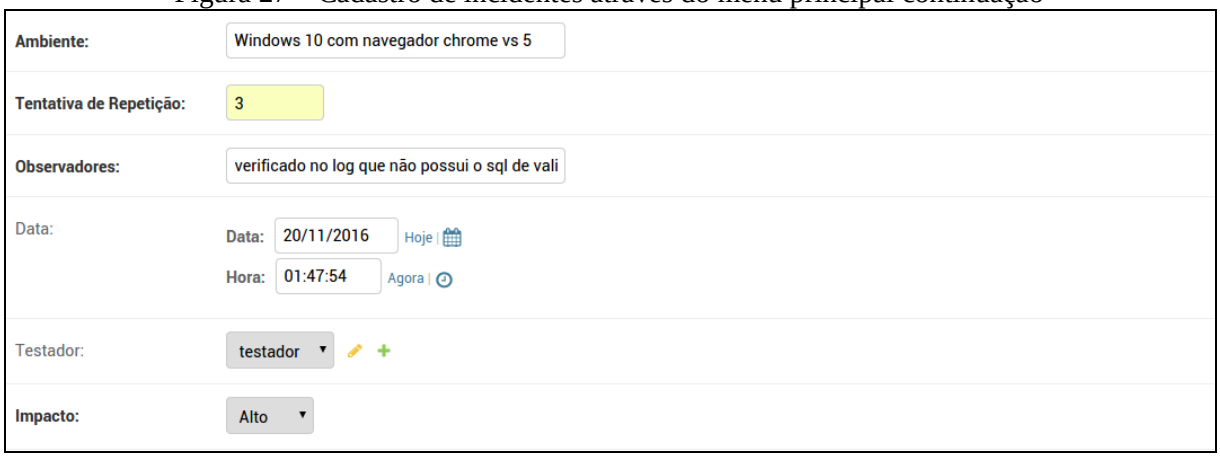
Na Figura 25 são apresentados os incidentes já cadastrados, detalhando código, título do incidente, caso de teste vinculado, impacto, resultado atual, campo resultado que representa a ação, resultado esperado e resultado final, anomalia que representa o problema encontrado na execução, ambiente de teste e quantidade de tentativas. Na parte superior esquerda é possível filtrar pelo código, título, caso de teste, impacto, resultado encontrado, anomalia, tentativa de repetição, data e testador do incidente, através do menu `FILTRO` é possível filtrar por impacto e testador.

Para cadastrar um incidente deve-se acessar a opção `ADICIONAR INCIDENTE +`, ao pressionar esta opção o testador é direcionado para a tela de cadastro de incidente conforme Figura 26 e Figura 27.

Figura 26– Cadastro de incidentes através do menu principal

Fonte: Elaborada pela autora.

Figura 27 – Cadastro de incidentes através do menu principal continuação

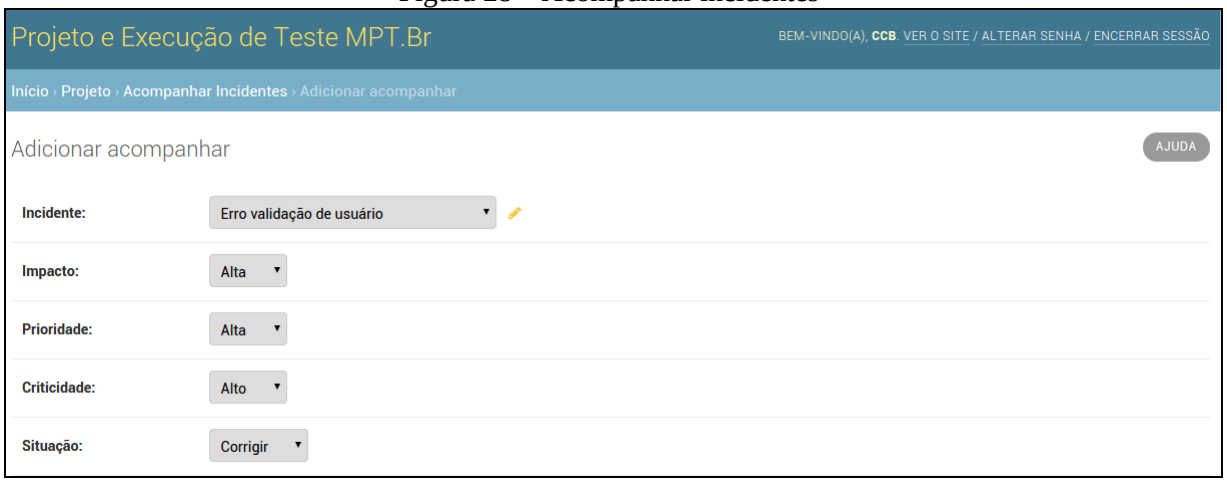


Fonte: Elaborada pela autora.

Neste tipo de cadastro de incidente o testador deve selecionar o caso de teste desejado e não é possível apontar qual passo apresentou falha. Na Figura 26 o testador deve selecionar o `Caso Teste`, informar o `Título`, `Resultado atual`, `Anomalia` e na Figura 27 `Ambiente`, `Tentativa de repetição`, `Observadores`, `Data`, `Testador` e `Impacto`.

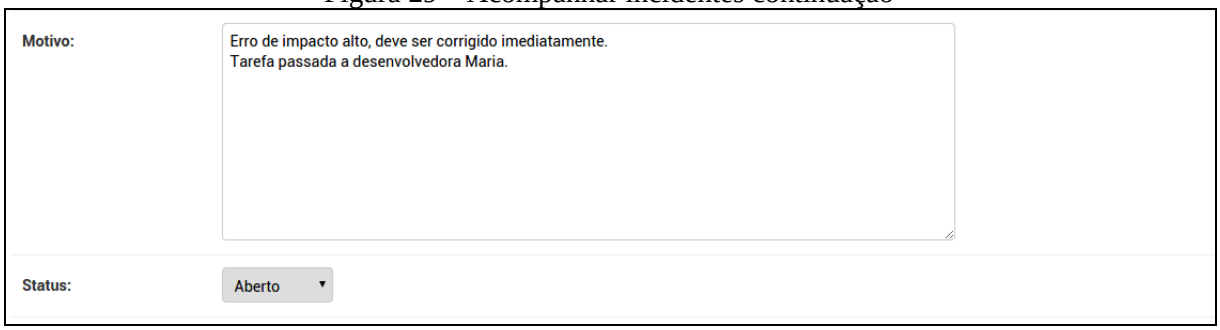
Após o cadastro do incidente o GCC deve tomar uma ação em relação ao incidente reportado. Através da tela acompanhar o GCC pode registrar a ação, conforme Figura 28 e Figura 29.

Figura 28 – Acompanhar incidentes



Fonte: Elaborada pela autora.

Figura 29 – Acompanhar incidentes continuação

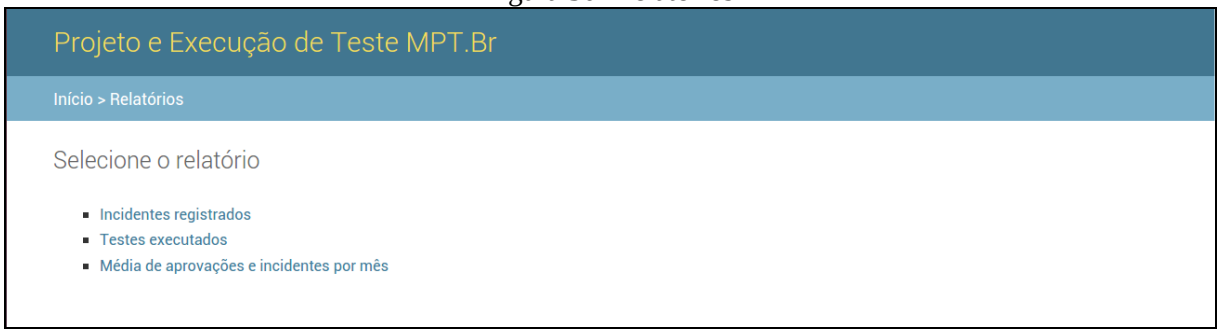


Fonte: Elaborada pela autora

Na tela da Figura 28 deve-se selecionar o incidente e informar os campos `Impacto`, `Prioridade`, `Criticidade` e `Situação`, `Motivo` e `Status` na tela da Figura 29. O status só deverá ser fechado quando o incidente for reprovado ou corrigido.

Para controle pode-se emitir os relatórios de incidentes registrados, execução dos testes e média de incidentes e aprovações por mês. O gestor deve acessar o menu `Relatórios` na tela principal, conforme Figura 30.

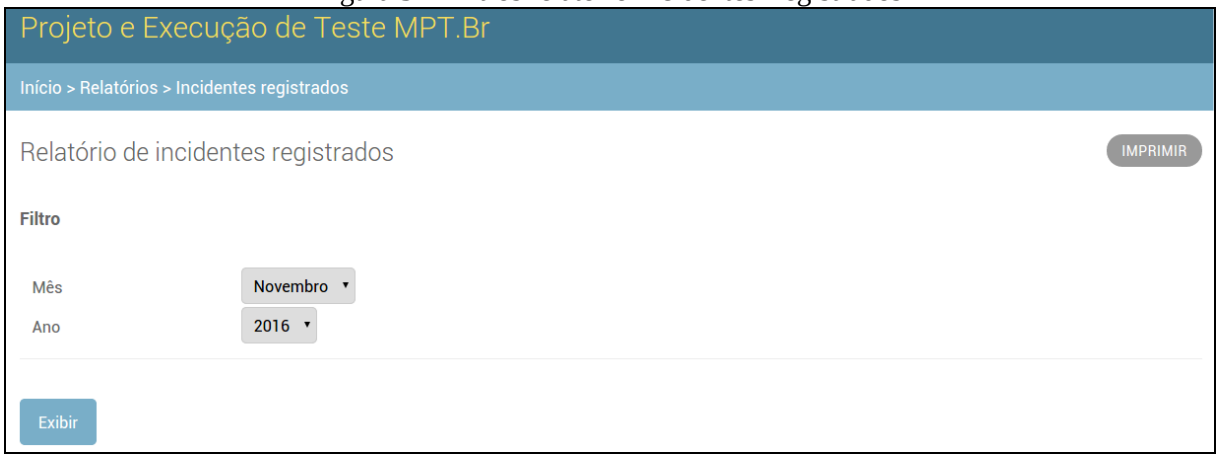
Figura 30- Relatórios



Fonte: Elaborada pela autora.

Na Figura 30 o gestor pode escolher o relatório que deseja visualizar. Ao selecionar na tela a opção `Incidentes Registrados` a seguinte tela é apresentada, conforme Figura 31.

Figura 31- Filtros relatório Incidentes Registrados

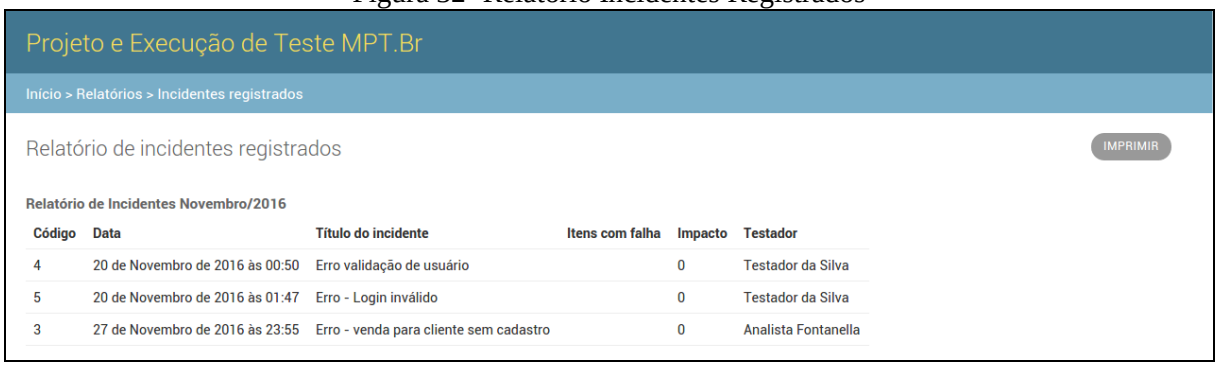


Fonte: Elaborada pela autora.

Na Figura 31 são apresentados os filtros `Mês` e `Ano` para demonstrar os incidentes registrados. Estes campos trazem por padrão o mês e ano atual. No canto superior direito o botão `IMPRIMIR` permite realizar a impressão trazendo as impressoras instaladas no sistema operacional.

Conforme Figura 32 ao pressionar `Exibir` as seguintes informações são apresentadas.

Figura 32- Relatório Incidentes Registrados

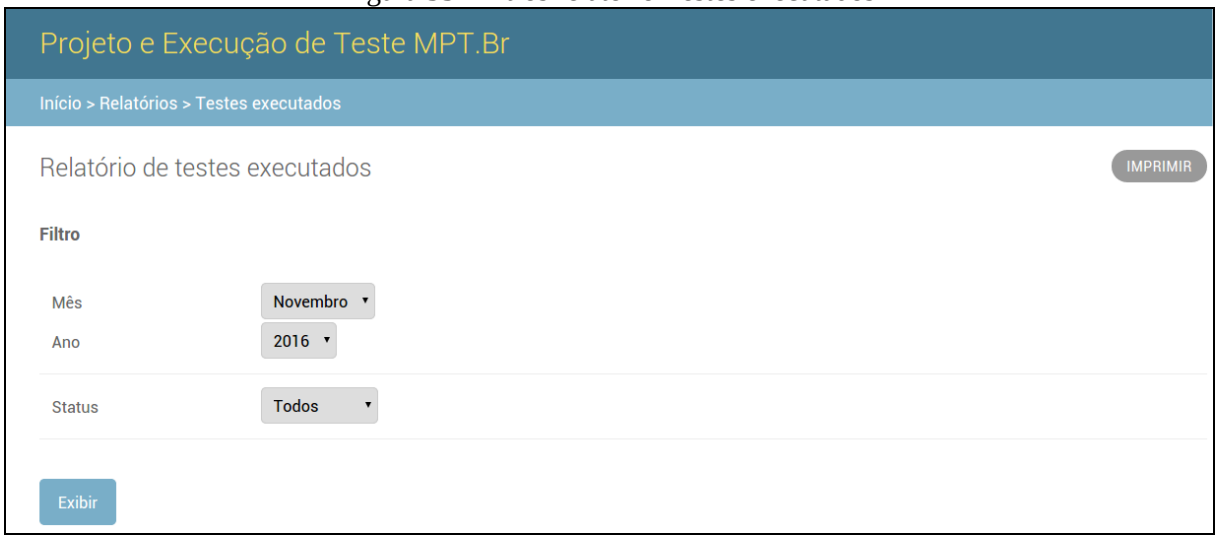


Fonte: Elaborada pela autora.

Na Figura 32 são apresentados o `Código` que representa código do incidente, `Data` que representa a data de registro, `Título` que representa o título do incidente, `Itens com falha` que representa quantidade de itens com falha, `Impacto` que representa o impacto do incidente e `Testador` que representa o testador que realizou o cadastro do incidente. No canto superior direito o botão `IMPRIMIR` permite realizar a impressão trazendo as impressoras instaladas no sistema operacional.

A Figura 33 representa a tela ao selecionar a opção `Testes executados`.

Figura 33- Filtros relatório Testes executados



Fonte: Elaborada pela autora.

Na Figura 33 é possível filtrar através dos campos `Mês`, `Ano` e `Status`. Mês e ano representam o mês e ano de registro da execução e trazem por padrão referente a data atual e

o status que pode ser todos, aprovado ou reprovado e por padrão será preenchido por todos. No canto superior direito o botão `IMPRIMIR` permite realizar a impressão trazendo as impressoras instaladas no sistema operacional. Ao pressionar `Exibir` são apresentadas as informações do relatório conforme Figura 34.

Figura 34- Relatório Testes executados

Fonte: Elaborada pela autora.

Na Figura 34 o relatório de testes executados apresenta o campo `Código`, `Data`, `Título do teste`, `Resultado`, `Testador` e `Duração (min)`. No canto superior direito o botão `IMPRIMIR` permite realizar a impressão trazendo as impressoras instaladas no sistema operacional.

3.4 RESULTADOS E DISCUSSÕES

O sistema desenvolvido neste trabalho atendeu as expectativas propostas, possibilitando desde o planejamento de um teste até a sua execução e registro de incidentes encontrados, conforme o MPT.BR ao que se refere ao Projeto e Execução de Teste (PET). No Quadro 9 tem-se um comparativo entre as funções do sistema e as funções estabelecidas no MPT.BR. A coluna resultado apresenta os seguintes valores:

a) A: Atende;
b) AP: Atende parcialmente;
c) N.A: Não atende

Quadro 9 – Comparativo entre as tarefas previstas pelo modelo e as funções do sistema

Tarefas MPT.BR Projeto e execução de teste	Aderência ao MPT.BR Projeto e execução de teste	Resultado
PET1 – Identificar casos de teste	O sistema permite o registro dos casos de teste.	A
PET2 – Executar casos de teste	O sistema permite o registro da execução dos casos de teste.	A
PET3 – Reportar incidentes	O sistema permite o registro de incidentes encontrados na execução do teste.	A
PET4 – Acompanhar incidentes	O sistema permite o registro da análise do incidente.	A

Fonte: Elaborada pela autora.

Baseado no Quadro 9 pode-se afirmar que o sistema desenvolvido atendeu satisfatoriamente os objetivos propostos. Além dos itens apontados no quadro, o sistema permite a geração de três relatórios para acompanhamento das informações cadastradas.

O sistema foi testado e validado através de uma avaliação informal por dois analistas de testes e um analista de sistemas representando o GCC. Na análise foi levado em consideração os pontos positivos, negativos e sugestões.

Para o analista de sistema o ponto positivo de possuir o acompanhamento de incidentes evita que erros se percam com o tempo por obrigar a tomada de decisão, não deixando o incidente cair no esquecimento e considerou o sistema de fácil manuseio.

Os analistas de testes reportaram que a ferramenta apresenta facilidade e agilidade nos processos básicos de gerenciamento de um caso de teste, apresentando rotinas simples de criação e vínculo para incidentes. Os ciclos de execução de casos de testes são rápidos para qualquer atitude tomada pelo testador, trazendo uma importância maior nos assuntos de grande impacto em um possível projeto. Como ponto negativo alguns campos possuem pouca usabilidade, podendo citar os campos obrigatórios anomalias e tentativa de repetição no cadastro de incidente.

Como melhorias para o trabalho desenvolvido foi sugerido uma forma de indicar se o incidente analisado trata de um erro pré-existente ou gerado a partir da nova implementação, trazendo assim um indicador para tomada de decisão pelo GCC. Ao adiar um incidente permitir cadastrar uma data indicando quando será analisando novamente, possuir cadastro de plano de teste com controle de tarefas e fornecer métricas de incidentes e tempo gasto em execução de testes.

Ao comparar o sistema desenvolvido com os trabalhos correlatos representado no Quadro 10, percebe-se que dois sistemas, Bonecher (2008) e Depiné (2005) possuem o

objetivo de gerenciar os testes, enquanto o sistema de Hoppe (1999) tem o objetivo de apoiar a manutenção de um software através de cadastros de erros e alterações.

Os trabalhos de Depiné (2005) e Hoppe (1999) foram desenvolvidos para a plataforma Desktop utilizando a linguagem Delphi e normas na sua maioria ISO/IEC, enquanto o sistema de Bonecher (2008) e o sistema proposto foram desenvolvidos para plataforma web. A principal diferença entre o sistema proposto e trabalhos correlatos é o padrão seguido baseado no modelo de referência MPT.BR, o banco de dados e a linguagem. O Quadro 10 representa a comparação entre a ferramenta desenvolvida neste trabalho e os trabalhos correlatos apresentados na seção 2.4.

Quadro 10 – Comparativo de resultados

	Bonecher (2008)	Depiné (2005)	Hoppe (1999)	Sistema Proposto
Plataforma	Web	Desktop	Desktop	Web
Linguagem	Java/ JSP	Delphi	Delphi	Python
Banco de Dados	MySQL	Interbase	Baseado nos arquivos Paradox	PostgreSQL
Norma/Padrão/ Modelo de referência	OpenUP e IEEE-829	ISO/IEC 9126, ISO/IEC 14598 e NBR/IEC 12119	ISO/IEC 12207, ISO 9000-3 e SPICE	MPT.Br
Permite cadastrar caso de teste?	Sim	Sim	Não	Sim
Permite cadastrar incidentes?	Sim	Sim	Sim	Sim
Permite cadastrar plano de teste?	Sim	Sim	Não	Não

Fonte: Elaborada pela autora.

4 CONCLUSÕES

Este trabalho apresentou o desenvolvimento de um sistema para suportar a área de processo Projeto e Execução de Teste utilizando o nível parcialmente gerenciado que representa o primeiro patamar de maturidade de uma organização baseado no modelo de referência MPT.BR.

O sistema foi implementado utilizando a linguagem de programação Python através do *framework* web Django e os resultados obtidos com o desenvolvimento deste sistema foram alcançados. Através do sistema é possível realizar o cadastro de caso de teste, execuções, incidentes, acompanhamento através da tomada de decisão e geração de relatórios por parte do gestor permitindo um gerenciamento disciplinado de forma planejada e controlada nos projetos de testes de uma empresa de software.

Com a avaliação informal dos analistas e com a percepção da autora foi possível concluir que a área do processo escolhido PET apresenta limitações na ferramenta ao necessitar de um controle maior na área de gerenciamento dos testes. Para possuir uma maior abrangência no sistema seria necessário a implementação de outras áreas de processos apresentados no modelo. Porém para projetos e aplicações de menor porte o sistema não apresenta limitações.

Por fim, conclui-se que o desenvolvimento deste trabalho proporcionou a autora um aprendizado e conhecimento na criação de sistema web e no modelo de Melhoria do Processo de Teste Brasileiro (MPT.BR).

4.1 EXTENSÕES

Como sugestão de extensões para o presente trabalho tem-se:

a) desenvolver novas funcionalidades através da área de processo Gerência de Projetos de Teste (GPT);
b) suportar os níveis de maturidade 2 e 3 referente a área de processo PET utilizando o modelo de referência MPT.BR;
c) desenvolver gráficos com os resultados obtidos através da execução dos testes;
d) implementar um controle para os incidentes gerados, sendo possível gerenciar a correção até o momento de envio da tarefa a qualidade;
e) aplicar um questionário formal através de uma amostra estatisticamente relevante.

REFERÊNCIAS

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

BASTOS, Aderson et al. **Base de conhecimento em teste de software**. São Paulo: Martins Editora Livraria Ltda, 2012.

BONECHER, Bruna Tatiane. **Ferramenta web de apoio ao planejamento e controle de teste de software.** 2008. 84f. Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) – Centro de Ciências Naturais e Exatas, Universidade Regional de Blumenau, Blumenau.

COLOMBO, Regina M. T; GUERRA, Ana Cervigni. **Tecnologia da informação:** qualidade de produto de software. Brasília: Ministério da Ciência e Tecnologia, 2009.

COSTA, Ivanir et al. **Qualidade em tecnologia da informação**. São Paulo: Atlas, 2013.

DEPINÉ, Juliano José. **Ferramenta de suporte e gerenciamento de avaliações da qualidade de sistemas**. 2005. 97f. Trabalho de Conclusão de Curso (Bacharelado em Sistemas de Informação) – Centro de Ciências Naturais e Exatas, Universidade Regional de Blumenau, Blumenau.

DJANGOPROJECT. **The web framework for perfectionists with deadlines.** [S.l.], 2016. Disponível em: <https://www.djangoproject.com/>. Acesso em: 20 nov. 2016.

GANDARA, Fernando. **Qualidade e teste em software**. [S.l.]: Clube dos autores, 2012.

HIRAMA, Kechi. **Engenharia de software:** qualidade e produtividade com tecnologia. Rio de Janeiro: Elsevier, 2012.

HOPPE, Charles. **Software de apoio a manutenção de sistemas baseado em normas de qualidade**. 1999. 110f. Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) – Centro de Ciências Naturais e Exatas. Universidade Regional de Blumenau, Blumenau.

KOSCIANSKI, André; SOARES, Michel S. **Qualidade de software**: Aprenda as metodologias mais modernas para o desenvolvimento de software. São Paulo: Novatec, 2006.

MELHORIA NO PROCESSO DE TESTE BRASILEIRO. **Guia de referência do modelo MPT.BR**. [Recife]: Softex, Riosoft, 2011. Disponível em: <http://mpt.org.br/mpt/wp-content/uploads/2013/05/MPT_Guia_de_referencia.pdf>. Acesso em: 03 mar. 2015.

MILANI, André. **PostgreSQL guia do programador**. São Paulo: Novatec Editora, 2008.

POSTGRESQL. **Sobre**. [S.l.], 2016. Disponível em: <http://www.postgresql.org/about/>. Acesso em: 24 nov. 2016.

PRESSMAN, S. R. **Engenharia de software**. São Paulo: Pearson Prentice Hall, 2006.

SILVA, Regis da. **Principais dúvidas de quem quer aprender Django**. [S.l.], maio 2014. Disponível em: <http://pythonclub.com.br/principais-duvidas-de-quem-quer-aprender-django.html#o-que-e-mtv>. Acesso em: 20 nov. 2016.

SWINKELS, Ron. **A comparison of TMM and other Test Process Improvement Models**. Frits Philips Institute, 2000. Disponível em: <http://www1.informatik.tu-muenchen.de/lehrstuhl_1/files/teaching/ws0708/ManagementSoftwareTesting/12-4-1-FPdef.pdf >. Acesso em: 13 dez. 2016.

APÊNDICE A – Descrição dos Casos de Uso

A seguir, são detalhados os principais casos de uso contemplados no diagrama apresentado na seção 3.3.1 deste trabalho. São os Quadros 11 ao 21.

Quadro 11- Descrição UC01

Caso de Uso	UC01 – Cadastrar usuários
Descrição	Permitir que o gestor realize o cadastro de usuários.
Ator	Gestor
Pré-condição	Usuário deve estar logado no sistema.
Pós-condição	O usuário é adicionado.
Cenário principal	1. Gestor acessa a opção “Usuários”; 2. Sistema apresenta a tela de usuários; 3. Gestor pressiona a opção “ADICIONAR USUÁRIO +”; 4. Sistema apresenta a tela de cadastro de usuário; 5. Gestor informa usuário, senha, confirmação de senha e pressiona salva; 6. Sistema salva, apresenta a mensagem de “Usuário adicionado com sucesso” e apresenta a tela de informações pessoais; 7. Gestor cadastra as informações pessoais, adiciona as permissões e pressiona salvar; 8. Sistema direciona para a tela de usuário e apresenta a mensagem “Usuário modificado com sucesso”
Cenário – alteração	No passo 2 do cenário principal o gestor opta por alterar um usuário. 2.1 Gestor pressiona o cadastro usuário; 2.2 Sistema apresenta a tela com as informações do usuário; 2.3 Gestor altera as informações necessárias e pressiona salvar; 2.4 Sistema direciona para a tela de usuário e apresenta a mensagem “Usuário modificado com sucesso”.
Cenário – exclusão	No passo 2 do cenário principal gestor opta por excluir o projeto. 2.1 Gestor seleciona um usuário através do check-box, seleciona a ação “Remover usuários selecionados” e pressiona o botão “Fazer”; 2.2 Sistema apresenta a tela de confirmação; 2.3 Gestor pressiona “Sim, tenho certeza” 2.4 Sistema direciona para a tela de usuários e apresenta a mensagem “Usuário removido com sucesso”.
Cenário – pesquisa	No passo 2 do cenário principal o gestor opta por pesquisar um usuário. 2.1 Gestor informa o que deseja pesquisar e pressiona o botão “Pesquisar”; 2.2 Sistema retorna as informações conforme a pesquisa.
Cenário – filtro	No passo 2 do cenário principal o gestor opta por realizar uma pesquisa através dos filtros. 2.1 Gestor seleciona o filtro que deseja; 2.2 Sistema retorna as informações conforme a pesquisa.

Fonte: Elaborada pela autora.

Quadro 12- Descrição UC02

Caso de Uso	UC02 – Efetuar *login*
Descrição	Permitir que o testador efetue *login*.
Ator	Testador
Pré-condição	Testador possuir cadastro no sistema.
Pós-condição	O usuário tem acesso as informações.
Cenário principal	1. Testador preenche seu usuário e senha; 2. Sistema valida os dados; 3. Sistema direciona o usuário para a página inicial.
Cenário de exceção	Nome do usuário e senha inválidos; Campo obrigatório não informado.

Fonte: Elaborada pela autora.

Quadro 13- Descrição UC03

Caso de Uso	UC03 – Cadastrar projeto
Descrição	Permitir que o analista de teste realize o cadastro do projeto.
Ator	Analista de teste
Pré-condição	Analista de teste deve estar logado no sistema.
Pós-condição	Cadastro do caso de teste realizado.
Cenário principal	1. Analista de teste acessa a opção “Projeto”; 2. Sistema apresenta a tela de projeto; 3. Analista de teste clica na opção “ADICIONAR PROJETO +”; 4. Sistema apresenta a tela de cadastro de projeto; 5. Analista preenche os campos e pressiona salvar; 6. Sistema valida os campos informados; 7. Sistema direciona para a tela de projeto e apresenta a mensagem “Projeto foi adicionado com sucesso”.
Cenário – alteração	No passo 2 do cenário principal o analista de teste opta por editar um projeto. 2.1 Analista de teste clica em um projeto; 2.2 Sistema apresenta as informações para edição; 2.3 Analista de teste realiza as alterações e pressiona salvar; 2.4 Sistema valida os campos informados; 2.5 Sistema direciona para tela de projetos e apresenta a mensagem “Projeto foi alterado com sucesso”.
Cenário – exclusão	No passo 2 do cenário principal o analista de teste opta por excluir um projeto. 2.1 Analista de teste seleciona um projeto e seleciona a ação “Remover Projeto Selecionados” e pressiona “Fazer”; 2.3 Sistema apresenta a tela questionando; 2.4 Analista de confirma a operação”; 2.5 Sistema direciona para a tela de projetos e apresenta a mensagem “Projeto removido com sucesso”.
Cenário – pesquisa	No passo 2 do cenário principal o analista de teste opta por pesquisar um projeto. 2.1 Analista de teste informa o que deseja pesquisar e pressiona “Pesquisar”; 2.2 Sistema retorna as informações conforme a pesquisa.
Cenário – filtro	No passo 2 do cenário principal o analista de teste opta por filtrar por responsável do projeto. 2.1 Analista de teste seleciona o responsável; 2.2 Sistema retorna as informações conforme a pesquisa.
Cenário de exceção	Campo obrigatório não informado.

Fonte: Elaborada pela autora.

Quadro 14 – Descrição UC04

Caso de Uso	UC04 – Cadastrar caso de teste
Descrição	Permitir que o analista de teste realize o cadastro de caso de teste.
Ator	Analista de teste
Pré-condição	Analista de teste deve estar logado no sistema.
Pós-condição	Cadastro do caso de teste realizado.
Cenário principal	1. Analista de teste acessa a opção “Caso de teste”; 2. Sistema apresenta a tela de caso de teste; 3. Analista de teste clica na opção “ADICIONAR CASO TESTE +; 4. Sistema apresenta a tela de cadastro de caso de teste; 5. Analista de teste preenche os campos e pressiona salvar; 6. Sistema valida os campos informados; 7. Sistema apresenta a tela de casos de teste cadastrados e a mensagem “Caso de teste adicionado com sucesso. ”
Cenário – alteração	No passo 2 do cenário principal o analista de teste opta por editar um caso de teste. 2.1 Analista de teste seleciona um caso de teste; 2.2 Sistema mostra as informações para edição; 2.3 Analista de teste realiza alterações necessárias e seleciona a opção salvar; 2.4 Sistema valida os campos informados; 2.5 Sistema direciona para a tela de casos de teste cadastrados e apresenta a mensagem “Caso teste modificado com sucesso. ”
Cenário – exclusão	No passo 2 do cenário principal o analista de teste opta por apagar um caso de teste. 2.1 Analista de teste seleciona um caso de teste; 2.2 Sistema mostra as informações para exclusão; 2.3 Analista de teste seleciona a opção apagar; 2.4 Sistema questiona a opção; 2.5 Analista de teste seleciona a opção “Sim, tenho certeza”; 2.6 Sistema apresenta a tela de caso de teste cadastrados e a mensagem “Caso de teste excluído com sucesso. ”
Cenário de exceção	Campo obrigatório não informado.

Fonte: Elaborada pela autora.

Quadro 15 - Descrição UC05

Caso de Uso	UC05 – Visualizar caso de teste
Descrição	Permitir que o usuário visualize o caso de teste.
Ator	Testador
Pré-condição	Usuário deve estar logado no sistema.
Pós-condição	O caso de teste é visualizado.
Cenário principal	1. Testador acessa o menu “Caso de teste”; 2. Sistema apresenta a tela de caso de teste; 3. Testador seleciona o caso de teste que deseja; 4. Sistema apresenta o caso de teste.

Fonte: Elaborada pela autora.

Quadro 16- Descrição UC06

Caso de uso	UC06 – Informar o resultado do teste
Ator	Testador
Pré-condição	O sistema deve possuir um caso de teste cadastrado ou reprovado.
Pós-condição	O cadastro do resultado do teste é realizado.
Cenário principal	1. Testador acessa o menu “Caso de Teste” 2. Sistema apresenta a tela de caso de teste; 3. Testador pressiona o botão “Executar”; 4. Sistema apresenta a tela de execução; 5. Testador informa o resultado do teste; 6. Testador seleciona a opção salvar; 7. Sistema valida os campos informados; 8. Sistema direciona para a tela de casos de teste e apresenta a mensagem de “Execução salva com sucesso. ”
Cenário alternativo	No passo 5 o testador opta por cadastrar um incidente. 5.1 Testador pressiona o botão “+” no campo “Incidentes”; 5.2 Sistema apresenta a tela de cadastro de incidente; 5.3 Testador informa todos os campos obrigatórios e salva; 5.4 Sistema fecha a página de incidente e vincula o incidente na tela de execução.
Cenário de exceção	Campo obrigatório não informado.

Fonte: Elaborada pela autora.

Quadro 17 – Descrição UC07

Caso de uso	UC07 – Cadastrar incidente
Descrição	Permitir o testador cadastrar incidentes
Ator	Testador
Pré-condição	Possuir caso de teste sem incidente vinculado.
Pós-condição	O incidente é cadastrado.
Cenário principal	1. Testador acessa o menu “Incidentes”; 2. Sistema apresenta a tela de incidentes; 3. Testador preenche os campos e pressiona salvar; 4. Sistema valida os campos informados; 5. Sistema direciona para a tela de incidentes e apresenta a mensagem de “Incidente adicionado com sucesso”.
Cenário de exceção	Campo obrigatório não informado.

Fonte: Elaborada pela autora.

Quadro 18- Descrição UC08

Caso de uso	UC08 – Cadastrar tomada de decisão
Descrição	Permitir que o GCC cadastre a tomada de decisão.
Ator	Grupo de Controle da Configuração - GCC
Pré-condição	Possuir incidente cadastrado no sistema.
Pós-condição	Registro da tomada de decisão.
Cenário principal	1. GCC acessa o menu “Gerenciar incidentes”; 2. Sistema apresenta a tela de gerenciar incidentes; 3. GCC preenche os campos e pressiona salvar; 4. Sistema valida os campos informados; 5. Sistema direciona para a tela de gerenciar incidentes e apresenta a mensagem de “Gerenciar incidentes adicionado com sucesso. ”
Cenário – alteração	No passo 2 do cenário principal o GCC opta por editar uma tomada de decisão. 2.1 GCC seleciona uma tomada de decisão; 2.2 Sistema mostra as informações para edição; 2.3 GCC realiza alterações necessárias e seleciona a opção salvar; 2.4 Sistema valida os campos informados; 2.5 Sistema direciona para a tela de gerenciar incidentes cadastrados e apresenta a mensagem “Gerenciar incidentes modificado com sucesso. ”
Cenário – exclusão	No passo 2 do cenário principal GCC opta por apagar uma tomada de decisão. 2.1 GCC seleciona uma tomada de decisão; 2.2 Sistema mostra as informações para exclusão; 2.3 GCC seleciona a opção apagar; 2.4 Sistema questiona a opção; 2.5 GCC seleciona a opção “Sim, tenho certeza”; 2.6 Sistema apresenta a tela de caso de gerenciar incidentes e a mensagem “Gerenciar incidentes excluído com sucesso. ”
Cenário de exceção	Campo obrigatório não informado.

Fonte: Elaborada pela autora.

Quadro 19- Descrição UC09

Caso de uso	UC09 – Gerar relatórios de incidentes
Descrição	Permitir o gestor gerar o relatório de incidentes
Ator	Gestor
Pré-condição	Possuir incidentes registrados no sistema
Pós-condição	Relatório gerado.
Cenário principal	1. Gestor acessar o menu “Relatórios”; 2. Sistema apresenta a tela de relatórios; 3. Gestor seleciona o relatório “Incidentes registrados”; 4. Sistema direciona para a página de filtros; 5. Gestor informar os filtros e pressiona “Exibir”; 6. Sistema carrega o relatório com as informações.

Fonte: Elaborada pela autora.

Quadro 20- Descrição UC10

Caso de uso	UC10 – Gerar relatório de testes executados
Descrição	Permitir o gestor gerar o relatório de testes executados
Ator	Gestor
Pré-condição	Possuir execuções registradas no sistema.
Pós-condição	Relatório gerado
Cenário principal	1. Gestor acessar o menu “Relatórios”; 2. Sistema apresenta a tela de relatórios; 3. Gestor seleciona o relatório “Testes executados”; 4. Sistema direciona para a página de filtros; 5. Gestor informar os filtros e pressiona “Exibir”; 6. Sistema carrega o relatório com as informações.

Fonte: Elaborada pela autora.

Quadro 21- Descrição UC11

Caso de uso	UC11 – Gerar relatório de incidentes e aprovações
Descrição	Permitir o gestor gerar o relatório de média de aprovações e incidentes por mês.
Ator	Gestor
Pré-condição	Possuir execuções aprovadas e incidentes registrados no sistema.
Pós-condição	Relatório gerado.
Cenário principal	1. Gestor acessar o menu “Relatórios”; 2. Sistema apresenta a tela de relatórios; 3. Gestor seleciona o relatório “Média de aprovações e incidentes por mês”; 4. Sistema direciona para a página de filtros; 5. Gestor informar os filtros e pressiona “Exibir”; 6. Sistema carrega o relatório com as informações.

Fonte: Elaborada pela autora.

APÊNDICE B – Dicionário de Dados

Este Apêndice apresenta a descrição das tabelas criadas no banco de dados para armazenamento de informações geradas pelo sistema. Foram utilizados quatro tipos de atributos:

a) `boolean`: para atributos de verdadeiro e falso;

b) `datetime`: para atributos de data e hora;

c) `int`: para atributos numéricos;

d) `text`: para atributos de texto;

Os Quadro 22 ao 33 representam o detalhamento das tabelas.

Quadro 22 – Descrição da tabela `AcaoResultado`

Entidade: Acaoresultado				
Descrição: Armazena as informações de ação e resultado do caso de teste				
Atributo	**Tipo**	**PK**	**FK**	**Descrição**
id	int	Sim	Não	Identificação única
caso_teste_id	int	Não	Sim	Identificação do caso de teste
acao	text	Não	Não	Ação
resultado_esperado	text	Não	Não	Resultado esperado da ação
resultado_item	boolean	Não	Não	Resultado da ação

Fonte: Elaborada pela autora.

Quadro 23 – Descrição tabela `Acompanhar`

Entidade: Acompanhar				
Descrição: Armazena as informações de tomada de decisão sobre o incidente				
Atributo	**Tipo**	**PK**	**FK**	**Descrição**
id	int	Sim	Não	Identificação única
incidente_id	int	Não	Sim	Identificação do incidente
motivo	text	Não	Não	Motivo
impacto	int	Não	Não	Impacto
acao	int	Não	Não	Ação
criticidade	int	Não	Não	Criticidade
prioridade	int	Não	Não	Prioridade
status	int	Não	Não	Prioridade

Fonte: Elaborada pela autora.

Quadro 24 – Descrição tabela `CasoTeste`

Entidade: CasoTeste				
Descrição: Armazena as informações do caso de teste.				
Atributo	**Tipo**	**PK**	**FK**	**Descrição**
id	int	Sim	Não	Identificação única
projeto_id	int	Não	Sim	Identificação do projeto
titulo_teste	text	Não	Não	Título
objetivo	text	Não	Não	Objetivo
precondicao	text	Não	Não	Pré-condição

Fonte: Elaborada pela autora.

Quadro 25– Descrição da tabela `Execucao`

Entidade: Execucao				
Descrição: Armazena a execução				
Atributo	**Tipo**	**PK**	**FK**	**Descrição**
id	int	Sim	Não	Identificação única
incidentes_encontrados	int	Não	Sim	Identificação do incidente
caso_teste_id	int	Não	Sim	Identificação do caso de teste
testador_id	int	Não	Sim	Identificação do testador
resultado	text	Não	Não	Resultado
observacao	text	Não	Não	Observação
duracao	int	Não	Não	Duração
data	datatime	Não	Não	Data e hora

Fonte: Elaborada pela autora.

Quadro 26– Descrição tabela `Grupo`

Entidade: Grupo				
Descrição: Armazena o grupo de usuário				
Atributo	**Tipo**	**PK**	**FK**	**Descrição**
id	int	Sim	Não	Identificação única
nome	text	Não	Não	Nome

Fonte: Elaborada pela autora.

Quadro 27 – Descrição tabela `GrupoPermissao`

Entidade: GrupoPermissao				
Descrição: Armazena o grupo de usuário e a permissão				
Atributo	**Tipo**	**PK**	**FK**	**Descrição**
id	int	Sim	Não	Identificação única
grupo_id	int	Não	Sim	Identificação do grupo
permissao_id	int	Não	Sim	Identificação da permissão

Fonte: Elaborada pela autora.

Quadro 28 – Descrição tabela `Incidente`

Entidade: Incidente				
Descrição: Armazena as informações de incidente				
Atributo	**Tipo**	**PK**	**FK**	**Descrição**
id	int	Sim	Não	Identificação única
caso_teste_id	int	Não	Sim	Identificação do caso de teste
autor_incidente_id	int	Não	Sim	Identificação do autor
acao_resultado_id	int	Não	Sim	Identificação ação resultado
resultado_encontrado	text	Não	Não	Resultado encontrado
anomalia	text	Não	Não	Anomalia
nome	text	Não	Não	Título
ambiente	text	Não	Não	Ambiente
tentativa_de_repeticao	int	Não	Não	Tentativa de repetição
observacao_acao	text	Não	Não	Observação
impacto	int	Não	Não	Impacto
data	datetime	Não	Não	Data e hora

Fonte: Elaborada pela autora.

Quadro 29 – Descrição tabela `Permissao`

Entidade: Permissao				
Descrição: Armazena as permissões				
Atributo	**Tipo**	**PK**	**FK**	**Descrição**
id	int	Sim	Não	Identificação única
nome	text	Não	Não	Nome
codinome	text	Não	Não	Codinome

Fonte: Elaborada pela autora.

Quadro 30 – Descrição tabela `Projeto`

Entidade: Projeto				
Descrição: Armazena as informações do projeto				
Atributo	**Tipo**	**PK**	**FK**	**Descrição**
id	int	Sim	Não	Identificação única
responsavel_id	int	Não	Sim	Identificação do responsável
nome	text	Não	Não	Nome
data_criacao	datetime	Não	Não	Data e hora

Fonte: Elaborada pela autora.

Quadro 31 – Descrição tabela `Usuario`

Entidade: Usuario				
Descrição: Armazena as informações do usuário				
Atributo	**Tipo**	**PK**	**FK**	**Descrição**
id	int	Sim	Não	Identificação única
password	text	Não	Não	Senha
ultimo_login	text	Não	Sim	Data e hora do último acesso
e_superusuario	boolean	Não	Não	Super usuário
primeiro_nome	text	Não	Não	Primeiro nome
ultimo_nome	text	Não	Não	Último nome
email	text	Não	Não	Endereço de correio eletrônico
e_administrativo	boolean	Não	Não	Membro da equipe administrativa
e_ativo	boolean	Não	Não	Ativo
username	text	Não	Não	Usuário
date_criacao	datetime	Não	Não	Data e hora de criação

Fonte: Elaborada pela autora.

Quadro 32 – Descrição tabela `UsuarioGrupo`

Entidade: UsuarioGrupo				
Descrição: Armazena as informações do vínculo do usuário e do grupo				
Atributo	**Tipo**	**PK**	**FK**	**Descrição**
id	int	Sim	Não	Identificação única
usuario_id	int	Não	Sim	Identificação do usuário
grupo_id	int	Não	Sim	Identificação do grupo

Fonte: Elaborada pela autora.

Quadro 33 – Descrição tabela `UsuarioPermissao`

Entidade: UsuarioPermissao				
Descrição: Armazena as informações do usuário e suas permissões				
Atributo	**Tipo**	**PK**	**FK**	**Descrição**
id	int	Sim	Não	Identificação única
usuario_id	int	Não	Sim	Identificação do usuário
permissao_id	int	Não	Sim	Identificação da permissão

Fonte: Elaborada pela autora.