Ministério da
Ciência, Tecnologia
e Inovação

sid.inpe.br/mtc-m19/2012/03.02.02.29-TDI

VVTESTE: AMBIENTE DE GERAÇÃO E GERENCIAMENTO DE TESTES E DE DEFEITOS DE SOFTWARE COMO APOIO A PROCESSOS DE VERIFICAÇÃO E VALIDAÇÃO

Marcos Flávio de Souza Reis

Dissertação de Mestrado do Curso de Pós-Graduação em Engenharia e Tecnologia Espaciais/Gerenciamento de Sistemas Espaciais, orientada pelos Drs. Maurício Gonçalves Vieira Ferreira, e Ana Maria Ambrosio, aprovada em 30 de março de 2012.

URL do documento original:
<http://urlib.net/8JMKD3MGP7W/3BESFE5>

INPE
São José dos Campos
2012

PUBLICADO POR:

Instituto Nacional de Pesquisas Espaciais - INPE
Gabinete do Diretor (GB)
Serviço de Informação e Documentação (SID)
Caixa Postal 515 - CEP 12.245-970
São José dos Campos - SP - Brasil
Tel.:(012) 3208-6923/6921
Fax: (012) 3208-6919
E-mail: pubtc@sid.inpe.br

Ministério da
Ciência, Tecnologia
e Inovação

sid.inpe.br/mtc-m19/2012/03.02.02.29-TDI

VVTESTE: AMBIENTE DE GERAÇÃO E GERENCIAMENTO DE TESTES E DE DEFEITOS DE SOFTWARE COMO APOIO A PROCESSOS DE VERIFICAÇÃO E VALIDAÇÃO

Marcos Flávio de Souza Reis

Dissertação de Mestrado do Curso de Pós-Graduação em Engenharia e Tecnologia Espaciais/Gerenciamento de Sistemas Espaciais, orientada pelos Drs. Maurício Gonçalves Vieira Ferreira, e Ana Maria Ambrosio, aprovada em 30 de março de 2012.

URL do documento original:
<http://urlib.net/8JMKD3MGP7W/3BESFE5>

INPE
São José dos Campos
2012

Dados Internacionais de Catalogação na Publicação (CIP)

R277v

Reis, Marcos Flávio de Souza.

VVTESTE: ambiente de geração e gerenciamento de testes e de defeitos de software como apoio a processos de verificação e validação / Marcos Flávio de Souza Reis. – São José dos Campos : INPE, 2012.

xxii + 165 p. ; (sid.inpe.br/mtc-m19/2012/03.02.02.29-TDI)

Dissertação (Mestrado em Engenharia e Tecnologia Espaciais/Gerenciamento de Sistemas Espaciais) – Instituto Nacional de Pesquisas Espaciais, São José dos Campos, 2012.

Orientadores : Drs. Maurício Gonçalves Vieira Ferreira, e Ana Maria Ambrosio.

1. testes de software. 2. validação e verificação. 3. base de conhecimento. 4. qualidade de software. 5. automatização de processos. I.Título.

CDU 629.7:004.415.53

Aprovado (a) pela Banca Examinadora em cumprimento ao requisito exigido para obtenção do Título de Mestre em

Engenharia e Tecnologia Espaciais/Gerenciamento de Sistemas Espaciais

Dr. Nilson Sant'Anna

Presidente / INPE / SJCampos - SP

Dr. Mauricio Gonçalves Vieira Ferreira

Orientador(a) / INPE / SJCampos - SP

Dra. Ana Maria Ambrosio

Orientador(a) / INPE / São José dos Campos - SP

Dra. Maria de Fatima Mattiello Francisco

Membro da Banca / INPE / São José dos Campos - SP

Dra. Eliane Martins

Convidado(a) / UNICAMP / Campinas - SP

Este trabalho foi aprovado por:

( ) maioria simples

(x) unanimidade

Aluno (a): Marcos Flávio de Souza Reis

São José dos Campos, 30 de março de 2012

“Se eu tivesse oito horas para derrubar uma árvore, passaria seis afiando meu machado”.

Abraham Lincoln

A minha esposa Priscilla e a meus pais, Célio e Terezinha.

AGRADECIMENTOS

Primeiramente, gostaria de agradecer a Deus por me abençoar com Sabedoria, Coragem e Força. Sabedoria para tomar as decisões corretas, coragem para colocá-las em prática e força para suportar as consequências e desafios gerados.

Agradeço a minha esposa Priscilla pelo apoio incondicional, por aceitar meus momentos de ausência e principalmente por apoiar ao meu lado as dificuldades do dia-a-dia.

A meus pais, Célio e Terezinha, por sempre me guiarem no caminho da educação e da ética, servindo de exemplo moral na minha caminhada.

Aos amigos, Luiz Renato e Stefano, pelo apoio dado durante o desenvolvimento dos módulos deste trabalho.

Aos meus orientadores Maurício e Ana, que acreditaram no meu potencial, pelas palavras de apoio, de auxílio, pelos exemplos dados tanto profissionais quantos pessoais. Sem nenhuma dúvida, sem eles este trabalho não poderia ter sido concluído com sucesso.

Ao INPE, pela oportunidade de aprimorar os meus conhecimentos, disponibilizando excelentes profissionais durante todo o curso.

RESUMO

Os modelos de referência de processo de desenvolvimento de software definem vários objetivos a serem alcançados durante a realização das atividades de Verificação e Validação (V&V), porém, cabe à organização que vai adotá-las criar um ferramental para dar apoio à geração, execução e gerenciamento de dados de testes e de defeitos de software. Tais atividades de testes geram uma grande quantidade de informações, sendo que constantemente elas precisam ser registradas, consultadas e alteradas. Esta dissertação apresenta um ambiente para dar apoio a atividades de testes de processos de V&V de software. O ambiente proposto, chamado de VVTeste, utiliza as ferramentas livres: TestLink para gerenciamento dos testes, Mantis para gerenciamento de defeitos e Condado para geração automática de casos de testes a partir de máquinas de estados finitas estendidas. Essas ferramentas configuradas e integradas através da VVTeste formam uma base de conhecimento única com as informações geradas durante todas as atividades de testes. O ambiente VVTeste disponibiliza informações que podem ser utilizadas como estatísticas, lições aprendidas e comparações entre projetos de teste, atendendo, assim, se não completamente, pelo menos parte dos objetivos dos processos de Verificação e Validação esperados pelos modelos de referência de desenvolvimento de software. Detalhes da implementação do ambiente VVTeste e um estudo de caso são apresentados nesta dissertação.

VVTESTE: AN ENVIRONMENT FOR GENERATION AND MANAGEMENT OF TESTS AND DEFECTS OF SOFTWARE TO SUPPORT VERIFICATION AND VALIDATION PROCESSES

ABSTRACT

Reference Models of software development processes define several objectives to be achieved during the performance of activities of verification and validation (V&V), requiring an organization which will use them, to create tools to support generation, execution and management of software test data and defects. Such testing activities generate a large amount of information, and they constantly need to be registered, queried and changed. This work presents an environment to support testing activities of Software Verification and Validation Processes. The proposed environment, called VVTeste, integrates the free tools: *TestLink* for test management, *Mantis* for defect management and *Condado* to automatically generate test cases starting from extended finite state machines. These tools configured and integrated through VVTeste create an unique knowledge base with information generated during all testing activities. The VVTeste environment provides information that can be used as statistics, metrics, lessons learned and comparisons between test projects, meeting the objectives expected by the rules of software development, with regard to verification and validation. Details about the implementation of the VVTeste environment and a case study are also presented in this work.

LISTA DE FIGURAS

Pág.

LISTA DE TABELAS

Pág.

LISTA DE SIGLAS E ABREVIATURAS

ATIFS	Ambiente de Testes baseado em Injeção de Falhas por Software
CBERS	China-Brazil Earth Resources Satellite
CMMI	Capability Maturity Model Integration
CoFI	Conformance Test and Fault Injection
IEEE	Institute of Electrical and Electronics Engineers
INPE	Instituto Nacional de Pesquisas Espaciais
ISO	International Organization for Standardization
MEF	Máquina de estados finitos
MEFE	Máquina de estados finitos estendida
MPS.br	Melhoria do Processo de Software Brasileira
SEI	Software Enginnering Institute
SOFTEX	Associação para a Promoção da Excelência do Software Brasileiro
SUT	System under test
V&V	Verificação e Validação

SUMÁRIO

Pág.

1 INTRODUÇÃO

O Instituto Nacional de Pesquisas Espaciais (INPE) alcançou um resultado acima do esperado nos satélites SCD-1 e SCD-2 lançados em 1993 e 1998 respectivamente. A vida útil do SCD-1 prevista para seis meses alcançou a marca de 18 anos de uso e o SCD-2 previsto para cerca de um ano foi desativado apenas em 2009 (INPE, 2010). Estes resultados colocaram o Brasil em um seleto grupo de países com conhecimento em engenharia, gerenciamento e operação de sistemas espaciais.

Essa condição alcançada pelo instituto abriu portas para novas parcerias, como a estabelecida com a China para construção da família de satélites *China-Brazil Earth Resources Satellite* (CBERS).

Como pode ser observado na Figura 1.1, o INPE tem em seu planejamento estratégico a participação em pelo menos 12 projetos de satélites nos próximos 10 anos.

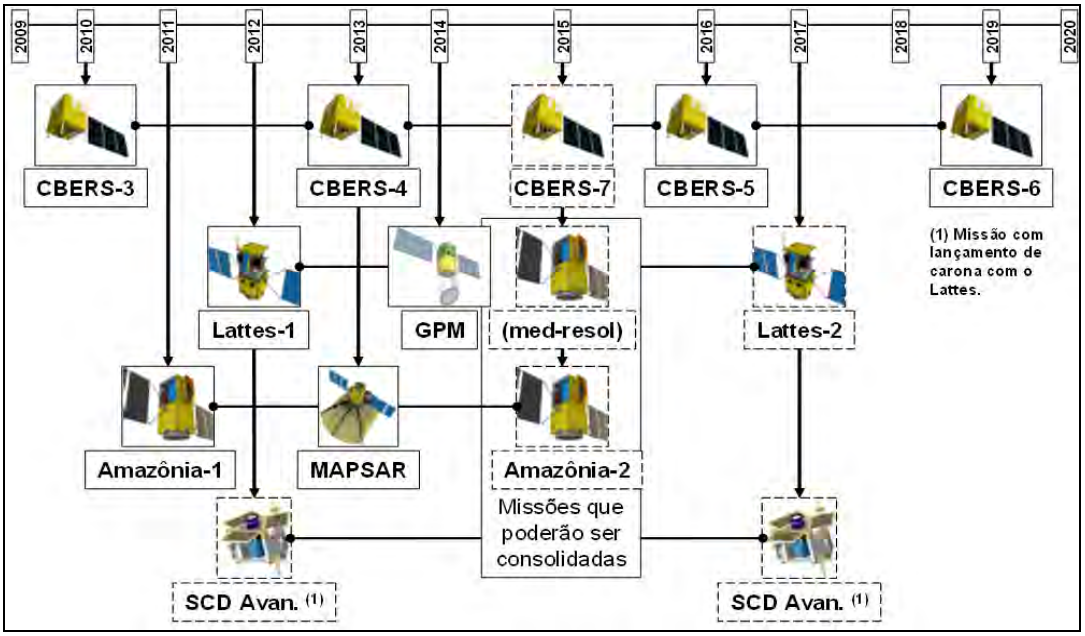

Figura 1.1 – Cronograma de lançamento de satélites do INPE.
Fonte: TOMINAGA (2010).

Para atender esta meta ambiciosa, uma grande gama de software para o controle dos satélites deverá ser desenvolvida; tais softwares apresentam um alto nível de complexidade e exigem alta confiabilidade.

Alguns softwares operam em solo e outros a bordo dos satélites. O software a bordo é chamado de embarcado ou embutido. Em vista da dificuldade no desenvolvimento desses softwares, as atividades de validação e verificação (V&V) também são complexas e caras. Alguns estudos como de (HECHT, et al., 2005), relatam que o custo de V&V em softwares de projetos espaciais varia de 0,1% a 3% do custo total das missões, valores considerados altos para um programa de satélites.

Diante desse cenário de demanda no desenvolvimento de softwares embarcado e da necessidade de diminuir os custos dos testes, sem a diminuição da qualidade e confiabilidade do software, o INPE tem trabalhado na implantação de um laboratório de V&V de softwares cujos processos foram desenvolvidos pela instituição em seus programas espaciais.

As principais atividades que este laboratório deverá apoiar são o planejamento, a execução e a analise dos resultados dos testes. O planejamento irá definir o que, como e quando os testes serão executados. Já a execução irá realizar as atividades planejadas no sistema em testes (SUT, do inglês *System under Test*), com a finalidade de encontrar erros (MYERS, 1979). A análise dos resultados pode requerer conhecimento prévio de resultado de outros projetos para comparações.

O gerenciamento de testes lida com uma grande quantidade de informação, que pode e deve ser reutilizada, derivando lições aprendidas e estatísticas para projetos futuros. Isso requer que o laboratório tenha uma estrutura que seja capaz de armazenar as informações geradas e auxiliar no gerenciamento das operações de testes, bem como na avaliação dos resultados.

Outra atividade custosa em um projeto de testes é a criação dos casos de testes, podendo chegar a 40% do esforço necessário no projeto (POL, et al., 2002). Além de custosa, essa atividade também não é trivial para sistemas complexos, pois requer um alto nível de detalhamento, conhecimento e cobertura do sistema.

Para auxiliar a geração de casos de testes, alguns sistemas dedicados conseguem disponibilizar automaticamente um conjunto de casos de testes a partir de um modelo, como por exemplo, uma máquina de estados finita. Este tipo de ferramenta recebe como entrada um modelo com transições e estados, algumas restrições e disponibilizam um conjunto de testes a serem executados contra o sistema.

Avaliando os esforços que serão realizados em um laboratório de testes, é perceptível a necessidade de buscar ferramentas que sejam capazes de aumentar a produtividade e a eficiência dos trabalhos realizados. Estas ferramentas devem estar inter-relacionadas a um processo. O processo de V&V de um software requer um esforço que percorre grande parte do ciclo de vida de um projeto de software. As atividades passam por avaliação dos requisitos, definição dos requisitos de testes, elaboração de um plano de testes, criação de casos de testes, preparação do ambiente de testes, execução dos testes e elaboração de relatos das validações.

Para minimizar o custo das atividades de teste é importante inovar. (SAKAR, 2007) explica que a palavra “inovar” vem do latim *in* + *novare*, que de maneira simples significa fazer novo, alterar ou renovar. Ele ainda afirma que inovação significa ter uma nova ideia ou aplicar as ideias de outras pessoas em novidades ou de uma forma nova.

O pensamento de Sakar e a necessidade apresentada pelo laboratório na criação de um ambiente para apoio aos processos de V&V, com ênfase nos

testes de software, motivaram a busca de soluções que pudessem colaborar com a instituição.

Neste sentido o trabalho desta dissertação buscou subsídios para a criação de um ambiente para apoio as atividades de V&V relacionadas à criação automática de teste, gerenciamento dos testes e dos defeitos de software, definindo uma estrutura para armazenamento e obtenção de informações sobre lições aprendidas para serem usadas em projetos futuros.

1.1. Objetivo

O objetivo deste trabalho de pesquisa é propor um ambiente capaz de apoiar as atividades de V&V relativas à criação de casos de teste, gerenciamento de testes e de defeitos, utilizando de ferramentas livres integradas, com a finalidade de aumentar a capacidade e eficiência no armazenamento das informações de teste geradas, bem como permitir o levantamento de dados estatísticos e métricas dos testes aplicados.

Para tanto, define-se as seguintes metas para este trabalho:

- Identificar objetivos dos processos de V&V de modelos de referência de desenvolvimento de software (O que?) que possam ser apoiados por um ambiente operacional eficiente (Como).
- Realizar um estudo de ferramentas livres que sejam capazes de auxiliar um laboratório de teste a atingir objetivos dos processos de V&V.
- Integrar e configurar as ferramentas utilizando como principio a Interoperabilidade, para que o ambiente possa ser expandido no futuro.
- Criar uma base de conhecimento de testes de software capaz de armazenar dados históricos, permitir o levantamento de estatísticas e auxiliar em planejamento de novos projetos de testes.
- Demonstrar a eficiência do ambiente proposto através de um estudo de caso.

1.2. Metodologia

Para a realização da concepção do ambiente de geração e gerenciamento de teste necessário para um laboratório de verificação e validação de software, foi elaborado um estudo dos conceitos, normas e ferramentas envolvidas em tais atividades. A Figura 1.2 apresenta todos estes pontos e como eles estão relacionados.

Inicialmente o estudo buscou em modelos de referência de desenvolvimento de software as melhores práticas e os objetivos que precisam ser alcançados durante as atividades de Verificação e Validação. Entre os vários modelos existentes, o CMMI e a MPS.br foram adotadas como referências neste trabalho.

Com relação aos conceitos de testes de softwares e a aplicação dos testes no seu ciclo de desenvolvimento, foram explorados às práticas de geração automática de casos de testes, de gerenciamento de testes e do gerenciamento de defeitos, que formam a estrutura deste trabalho.

Com relação à geração automática de casos de testes, foi escolhida a ferramenta CONDADO, por estar disponível e ter sido usada em vários projetos no INPE. Esta ferramenta auxilia a atividade de geração de teste considerando as características de um software embarcado no que concerne ao seu comportamento através da técnica de modelagem em Máquina de Estados Finita Estendida (MEFE).

Para o gerenciamento de todas as informações geradas e utilizadas durante a realização dos testes, a ferramenta, dedicada em gerenciamento de teste, *TestLink* foi selecionada como candidata a este ambiente. No gerenciamento de defeitos, a ferramenta *Mantis* apresentou como opção ao ambiente.

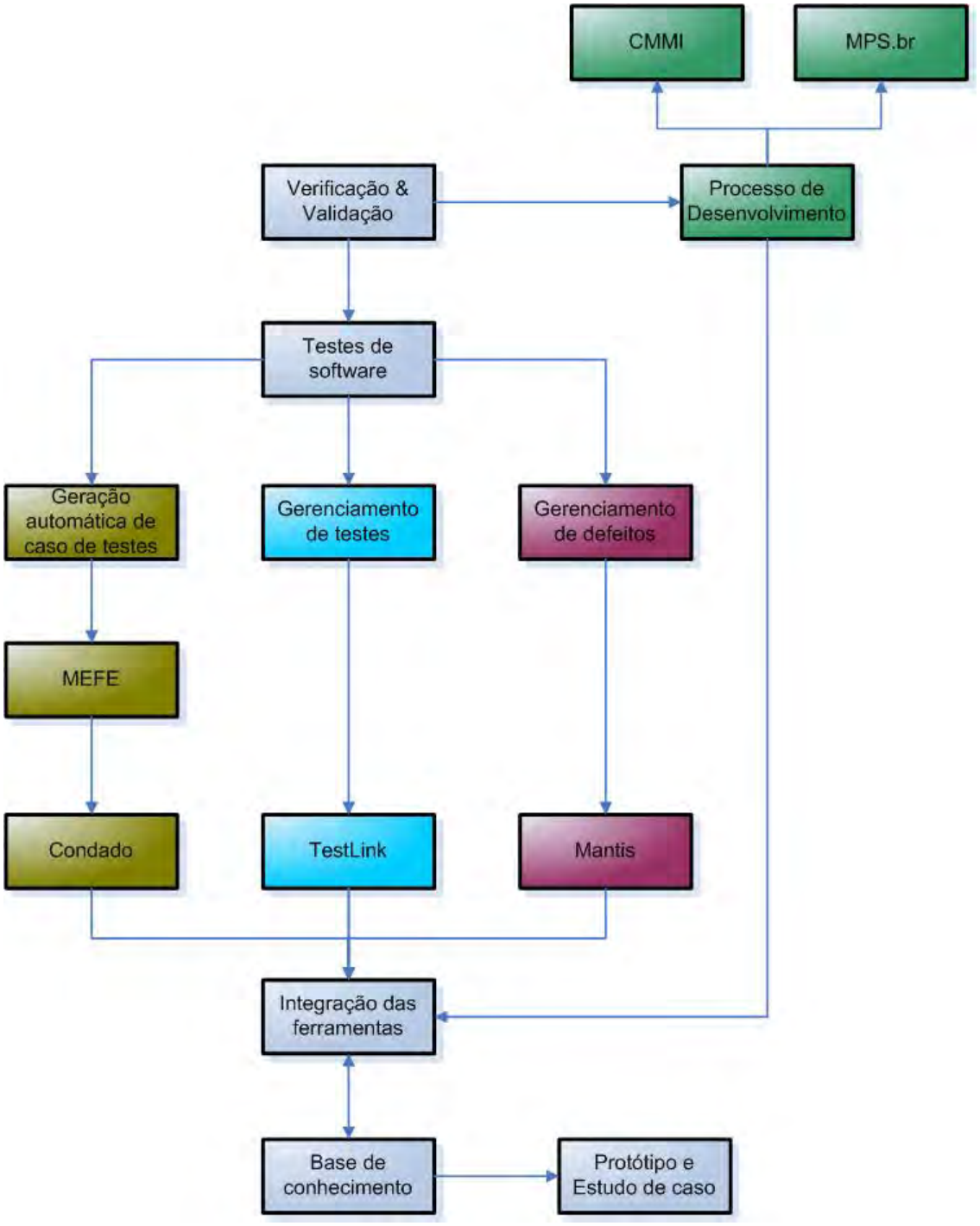

Figura 1.2 - Esquema de estudo para proposição do VVTeste.

Um estudo de como realizar a integração e como configurar as ferramentas para o armazenamento e recuperação das informações geradas durante os testes foi realizado.

Além do trabalho de integração das ferramentas, um estudo sobre todas as informações importantes para relatórios de lições aprendidas e estatísticas de projetos de testes foi realizado. Este estudo resultou em uma base de conhecimento única para todas as ferramentas integradas. Essa base de conhecimento auxilia no atendimento de objetivos dos processos de desenvolvimento de Verificação e Validação e também de outros processos dos modelos de referência.

Um protótipo de sistema foi criado para consumo das informações da base de conhecimento através de gráficos, demonstrando a capacidade de gerar novas informações através dessa base foi realizado.

Todo o estudo baseou-se em soluções que atendessem a requisitos como: Ferramentas livres, Integração com Interoperabilidade, Facilidade de uso e atendimento aos requisitos esperados.

1.3. Organização do trabalho

Este trabalho será organizado em oito capítulos sendo este primeiro capítulo a introdução ao trabalho.

O Capítulo 2 apresenta o estudo realizado sobre qualidade de software, aborda os modelos de referência de desenvolvimento CMMI e MPS, e descreve os conceitos de testes de softwares, gerenciamento de testes e defeitos.

No Capítulo 3 são apresentados alguns trabalhos correlatos que apresentam propostas semelhantes a este e também relatos sobre as utilizações das ferramentas usadas neste trabalho.

O Capítulo 4 descreve as ferramentas utilizadas e suas principais características.

No Capítulo 5 apresenta o ambiente VVTeste e como as ferramentas selecionadas são integradas e configuradas. Os módulos desenvolvidos e a base de conhecimento de testes de software são descritos.

No Capítulo 6 o ambiente proposto é avaliado em relação aos modelos de desenvolvimento de software.

No Capítulo 7 o ambiente VVTeste é utilizado em um estudo de caso e seus resultados são apresentados.

Por fim, no Capítulo 8 são apresentadas as conclusões alcançadas por este trabalho e as considerações finais.

Os Apêndices de A a E disponibilizam informações complementares sobre os módulos desenvolvidos pelo ambiente VVTeste.

Nos Apêndices F e G são apresentados detalhamentos sobre a documentação gerada durante o estudo de caso realizado neste trabalho.

2 QUALIDADE E TESTES DE SOFTWARE

Chegar a um conceito capaz de abordar de maneira clara e completa os objetivos a serem alcançados para a qualidade tem sido um desafio para diversas pessoas nos últimos anos. (JURAN, 1988) observou a qualidade da seguinte maneira:

“Segundo uma óptica de resultados a qualidade consiste nas características do produto que satisfazem as necessidades do cliente e geram lucros, entretanto, alta qualidade implica, geralmente, maiores custos. Segundo uma óptica de custos, a qualidade é a ausência de defeitos ou erros de fabricação, logo alta qualidade custa, em regra, menos dinheiro para as empresas.”

A definição de Juran aborda aspectos estratégicos que são considerados ao se planejar um sistema de qualidade em uma organização. (CROSBY, 1979) declarou sucintamente:

“Qualidade é conformidade com os requisitos”

A palavra “requisito” de acordo com o dicionário Michaelis (MICHAELIS, 2012), significa:

“Exigência imprescindível para a consecução de certo fim.”

Sendo assim, descobrir as reais necessidades do cliente e transformá-las em requisitos documentados, se torna uma atividade essencial para o sucesso da Qualidade. (FEIGENBAUM, 1994) criador do modelo “*Total Quality Control*” (TQC), expandiu o conceito da qualidade para todas as pessoas de uma organização, e chegou à seguinte conclusão:

“Qualidade é a correção dos problemas e de suas causas ao longo de toda série de fatores relacionados com marketing, projetos, engenharia, produção e manutenção, que exercem influência sobre a satisfação do usuário.”

As definições de Juran, Crosby e Feigenbaum se complementam ao se aplicar a qualidade em um software. O primeiro ao declarar a visão estratégica e de quebra de paradigma, o segundo ao afirmar a necessidade de transformar desejos de clientes em requisitos de produto e o terceiro por apresentar uma filosofia de qualidade em todos os setores de uma organização, visando à satisfação dos usuários.

Com tais contribuições conceituais e considerando os aspectos de desenvolvimento de software, (GUERRA, et al., 2009) afirma:

“Pode-se definir qualidade de produto de software como a conformidade a requisitos funcionais e de desempenho declarados explicitamente, padrões de desenvolvimento claramente documentados e as características implícitas que são esperadas de todo software desenvolvido profissionalmente”.

A construção de software não é uma tarefa simples. Pelo contrário, pode se tornar bastante complexa, dependendo das características e dimensões do sistema a ser criado (DELAMARO, et al., 2007). Por conta disso, a preocupação com a qualidade de software tem aumentado nos últimos anos, principalmente em sistema críticos, como por exemplo, os softwares embarcados.

Pesquisas em Engenharia de Software demonstram a importância de sistematização e processos para o desenvolvimento de software. Segundo (PRESSMAN, 2002), “a Engenharia de Software é uma disciplina que pode ser vista, de forma objetiva, com o estabelecimento e o uso dos princípios básicos

da engenharia, com a finalidade de desenvolver software de maneira sistemática e econômica, resultando em um produto confiável e eficiente".

As falhas em software geraram grandes prejuízos em projetos espaciais. Dois acontecimentos marcantes ilustram tais prejuízos: uma falha ocorrida durante o lançamento do Ariane 5, que explodiu por causa de um problema de software, gerou prejuízos da ordem de 300 milhões de dólares (HECHT, et al., 2005). A falha de transmissão de um dos canais de comunicação da Sonda Huygens da Nasa, prejudicou o envio das imagens, que passaram a ser transmitidas pela metade (DEUTSCH, 2002).

Para que tais erros sejam descobertos antes do software ser liberado para utilização, existe uma séria de atividades, coletivamente chamadas de "Validação, Verificação e Testes", ou "VV&T" (DELAMARO, et al., 2007).

A Associação para Promoção da Excelência do Software Brasileiro (SOFTEX) e o Programa Brasileiro da Qualidade e Produtividade em Software (PBQP Software) são exemplos do esforço brasileiro para melhoria da qualidade de software no Brasil. A SOFTEX, desde 2003, tem liderado o estabelecimento do programa para Melhoria do Processo de Software Brasileira (MPS.br)

No contexto mundial, o Instituto de Engenharia de Software, do inglês, *Software Engennering Institute* (SEI), mantido pelo Departamento de Defesa dos Estados Unidos, têm desenvolvido padrões mundialmente reconhecidos pela Organização Internacional para Padronização, do inglês, *International Organization for Standardization* (ISO) e pelo Modelo de Maturidade de Capacidade de Engenharia de Sistema, do inglês, *System Engineering Capability Maturity Model* (SE-CMM).

Em 2000, o SEI estabeleceu um novo modelo, a partir do já existente CMM, para o desenvolvimento de software, que até então, contava com o SW-CMM

(*Capability Maturity Model for Software*), o SE – CMM (*System Engineering Capability Maturity*) e o IPD-CMM (*Integrated Product Development*). A integração desses modelos criou o *Capability Maturity Model Integration* (CMMI).

Tanto o CMMI como a MPS.br definem processos e objetivos a serem alcançados durante as atividades de Verificação e a Validação do software.

2.1. MPS.br

O programa MPS.br visa orientar empresas para implantação de processos de qualidade de software. Ele se adequa ao perfil de empresas com diferentes tamanhos e características, públicas e privadas, embora, especial atenção seja dada às micro, pequenas e médias empresas. O modelo MPS é compatível com os padrões de qualidade aceitos internacionalmente e tem como pressuposto o aproveitamento de toda a competência existente nos padrões e modelos de melhoria de processo já disponíveis (SOFTEX, 2011).

O modelo de referência MPS.br é dividido em quatro partes que se completam, são elas: Guia Geral, Guia de Avaliação, Guia de Aquisição e um conjunto de onze Guias de Implementação.

Uma organização que deseja evoluir na sua capacidade de desenvolvimento de software será analisada pela MPS.br através de estágios. Em cada estágio o nível de maturidade da organização é avaliado de acordo com os objetivos definidos pelo modelo. Ao todo são 7 níveis, que se iniciam em G (Parcialmente gerenciado) e chegam até o nível de maturidade A (Em otimização).

Além da divisão por estágio de maturidade, a MPS.br também é organizada em 19 processos, que devem ser definidos e aplicados na organização. O nível A

não possui processos específicos e sim a melhoria de vários processos, a Tabela 2.1 apresenta o nível de maturidade de B a G e seus respectivos processos esperados.

Tabela 2.1 - Processos da MPS.br

Processo	Sigla	Nível
Gerência de Projetos (Evolução)	GPR	B
Gerência de Riscos	GRI	C
Desenvolvimento para Reutilização	DRU	C
Gerência de Decisões	GDE	C
Verificação	VER	D
Validação	VAL	D
Projeto e Construção do Produto	PCP	D
Integração do Produto	ITP	D
Desenvolvimento de Requisitos	DRE	D
Gerência de Projetos (Evolução)	GPR	E
Gerência de Reutilização	GRU	E
Gerência de Recursos Humanos	GRH	E
Definição do Processo Organizacional	DFP	E
Avaliação e Melhoria do Processo Organizacional	AMP	E
Medição	MED	F
Garantia da Qualidade	GCA	F
Gerencia de Portfólio de Projetos	GPP	F
Gerencia de Configuração	GCO	F
Aquisição	AQU	F
Gerencia de requisitos	GRE	G
Gerencia de projetos	GPR	G

Fonte: Guia Geral da MPS.br (2011)

Os processos de Validação e de Verificação são abordados no modelo no nível de maturidade D (Largamente Definido).

No contexto da MPS.br, a Validação é a confirmação, por exame e fornecimento de evidência objetiva, de que os requisitos específicos, para um determinado uso pretendido, são atendidos (ISO/IEC, 2008). O objetivo da

validação é assegurar que o software que está sendo desenvolvido é o software correto de acordo com os requisitos do usuário (ROCHA, et al., 2001).

Uma forma de realizar a validação é aplicar uma série de testes que demonstrem que o produto correto está sendo desenvolvido. O teste é um dos mais importantes métodos de garantia da qualidade do produto e, frequentemente, o mais usado (TIAN, 2005).

A MPS.br define sete resultados esperados para o processo de Validação, os quais são apresentados na Tabela 2.2.

Tabela 2.2 - Resultados esperados do Processo de Validação.

Resultado esperado	**Sigla**
Produtos de trabalho a serem validados são identificados	VAL1
Uma estratégia de validação é desenvolvida e implementada, estabelecendo cronograma, participantes envolvidos, métodos para validação e qualquer material a ser utilizado na validação.	VAL2
Critérios e procedimentos para validação dos produtos de trabalho a serem validados são identificados e um ambiente para validação é estabelecido	VAL3
Atividades de validação são executadas para garantir que o produto esteja pronto para uso no ambiente operacional pretendido	VAL4
Problemas são identificados e registrados	VAL5
Resultados de atividades de validação são analisados e disponibilizados para as partes interessadas	VAL6
Evidências de que os produtos de software desenvolvidos estão prontos para o uso pretendido são fornecidas	VAL7

Fonte: Guia de implementação do nível D da MPS.br (2011).

O processo de Verificação trata de como avaliar produtos de trabalho e serviços, garantindo que atendam a seus requisitos, por meio da identificação dos itens a serem verificados, do planejamento da verificação de cada um

destes itens e da execução da verificação conforme planejado ao longo do desenvolvimento do produto (SOFTEX, 2011).

A verificação de produtos de software deve ser realizada por meio da aplicação de métodos e técnicas específicos ao longo do desenvolvimento do produto. Dois dos principais métodos de verificação são as revisões por pares e testes (THELIN, 2002).

A MPS.br define 6 resultados esperados para o processo de verificação, os quais são apresentados na Tabela 2.3.

Tabela 2.3 - Resultados esperados do Processo de Verificação.

Resultado esperado	**Sigla**
Produtos de trabalho a serem verificados são identificados	VER1
Uma estratégia de verificação é desenvolvida e implementada, estabelecendo cronograma, revisores envolvidos, métodos para verificação e qualquer material a ser utilizado na verificação	VER2
Critérios e procedimentos para verificação dos produtos de trabalho a serem verificados são identificados e um ambiente para verificação é estabelecido	VER3
Atividades de verificação, incluindo testes e revisões por pares, são executadas	VER4
Defeitos são identificados e registrados	VER5
Resultados de atividades de verificação são analisados e disponibilizados para as partes interessadas	VER6

Fonte: Guia de implementação do nível D da MPS.br (2011).

Os processos de Validação e Verificação estão fortemente ligados e normalmente são executados paralelamente. Resumidamente podemos definir que a verificação se preocupa em avaliar se o produto está sendo desenvolvido corretamente, enquanto a validação visa assegurar que se está desenvolvendo o produto correto, isto é, o produto que o cliente deseja (BOEHM, 1981).

2.2. CMMI

O modelo de referência CMMI, que contém práticas para medir e orientar a maturidade de desenvolvimento de software surgiu com a finalidade de integrar os conteúdos já existentes e assegurar a relação com a ISO/IEC 15504 (*Software Process Improvement and Capability Determination*). Para isso contou com o apoio do governo americano, da SEI e das indústrias. A primeira versão deste modelo foi disponibilizada em 2000. Atualmente o CMMI é composto por três modelos: para desenvolvimento, o CMMI *for Development* (CMMI - DEV), para aquisição, o CMMI *for Acquisition* (CMMI - ACQ) e para serviços, o CMMI *for Services* (CMMI - SER).

O objetivo do CMMI (CMMI, 2010) é desenvolver a maturidade da organização e a melhoria de seu processo de desenvolvimento de software. A implantação do modelo pode ser realizada de duas maneiras: (i) Representação Continua que é baseada no modelo proposto pela ISO/IEC 15504 e (ii) Representação por Estágio, que é conforme o CMM da própria SEI.

2.2.1. Representação continua

A Representação Continua possui seis níveis de capacidade, iniciando em 0 (Incompleto) chegando até o 6 (Otimizando), que é adequada à norma ISO/IEC 15504. Cada melhoria da organização é chamada de área de processo, que nesta representação são agrupadas em quatro categorias: Gerenciamento de processo, Gerenciamento de projeto, Engenharia e Suporte.

Para que a área de processo seja melhorada são definidos objetivos e práticas específicas, que serão aplicados para atender exclusivamente a um processo e objetivos e práticas genéricas, que têm a finalidade de definir uma sequência a ser seguida, permitindo uma integração entre os processos.

Para cada área de processo, são definidos objetivos e práticas genéricas, que são utilizados em diversos momentos do processo. Também são declarados objetivos e práticas específicas, que serão aplicados em processos individuais. Um processo só atinge o nível 1 de capacidade, quando todas as práticas e objetivos específicos definidos ao processo são realizados.

Os processos de Verificação e Validação fazem parte dos 22 processos definidos por essa representação e são classificados na categoria Engenharia.

2.2.2. Representação por estágio

Como no SW-CMM e na MPS.br, as áreas de processo são organizadas em níveis de maturidade. O processo serve de parâmetro para avaliar a capacidade da organização, diferentemente da representação continua que avalia a capacidade de processos.

A representação por estágio sugere uma ordem de evolução da organização, através das áreas de processo de cada nível de maturidade. Outro aspecto importante desta representação, é que a cada avanço de nível pela organização, isso significa que além do nível por ela alcançada, ela satisfaz todos os processos dos níveis anteriores. Para avaliar a maturidade da organização são definidos 5 níveis que vão de 1 (Inicial) até 5 (Otimizado).

A versão 1.3 do CMMI – DEV disponibilizada em 2010 (CMMI, 2010), define 22 áreas de processos divididas entre os níveis 2, 3, 4 e 5. Para o modelo, o nível 1 serve para que a organização possa avaliar os seus processos mais críticos, definir os padrões iniciais e se preparar estruturalmente para iniciar a implantação das melhorias descritas pelo modelo.

Os processos de Verificação e Validação devem ser implementados a partir do nível 3 de maturidade.

2.2.3. Processos de Verificação e Validação

Os processos de Verificação e Validação são semelhantes, porém tratam de questões diferentes. A Verificação tem a finalidade de avaliar o produto de acordo com os requisitos especificados, já a Validação tem o propósito de demonstrar que um produto ou componente de produto atende ao seu uso pretendido quando colocado em seu ambiente (CMMI, 2010).

Para tanto o CMMI define metas e práticas para cada um destes processos. A Tabela 2.4 mostra as metas e suas respectivas práticas especificadas para o processo de Verificação (VER).

Tabela 2.4 - Verificação - Metas por práticas específicas.

Metas	**Práticas específicas**
SG1 – Preparar a Verificação	SP 1.1 – Selecionar os produtos de trabalho para Verificação
	SP 1.2 – Estabelecer o ambiente de Verificação
	SP 1.3 – Estabelecer procedimentos e critérios de Verificação
SG2 – Realizar revisão em pares	SP 2.1 – Preparar para revisão por pares
	SP 2.2 – Realizar revisão por pares
	SP 2.3 – Analisar dados de revisão por pares
SG3 – Verificar os produtos de trabalhos selecionados	SP 3.1 – Realizar verificação
	SP 3.2 – Analisar resultados de verificação e identificar ações corretivas

Fonte: Guia CMMI para desenvolvimento versão 1.3. (2010).

O processo de Validação dispõe de apenas duas metas, conforme apresentado na Tabela 2.5.

Tabela 2.5 - Validação - Metas por práticas especificadas.

Metas	Práticas específicas
SG1 – Preparar a Validação	SP 1.1 – Selecionar os produtos para Validação
	SP 1.2 – Estabelecer o ambiente de Validação
	SP 1.3 – Estabelecer procedimentos e critérios de Validação
SG2 – Validar o Produto ou os Componentes de Produto	SP 2.1 – Realizar Validação
	SP 2.2 – Analisar resultados de Validação

Fonte: Guia CMMI para desenvolvimento 1.3. (2010)

2.3. Testes de software

A definição de testes de software pode ser dada de diferentes formas, como:

> *"O processo de executar um programa com o objetivo de encontrar erro."*
>
> *(MYERS, 1979)*

Ou

> *"O processo de avaliar um sistema ou um componente de um sistema por meios manuais ou automáticos para verificar se ele satisfaz os requisitos especificados ou identificar as diferenças entre resultados esperados e obtidos." (IEEE, 1983)*

Ou ainda

> *"O teste de software é uma atividade dinâmica, ou seja, são aquelas que se baseiam na execução de um programa ou de um modelo, utilizando algumas entradas em particular e verificar se seu comportamento está de acordo com o esperado."*
>
> *(DELAMARO, et al., 2007).*

Uma definição interessante para um processo de testes de software seria a união dessas três abordagens. A primeira exalta a importância de avaliar as operações do sistema em relação a requisitos esperados, buscando ver se em

condições normais o software realiza as suas atividades de maneira correta. Já a definição de Myers aborda a situação que a atividade de testes deve tentar “sabotar” o software, no intuito de encontrar defeitos. A terceira definição completa dizendo que o software não deve fazer nada além previsto, evitando que funções desnecessárias gerem falhas.

Existem vários tipos de testes de software, como: Testes de Caixa Preta, Caixa Branca, Teste de regressão, Teste de carga, etc. Cada um desses tipos é utilizado em uma fase do desenvolvimento e com finalidades distintas.

Durante as atividades de testes são utilizados alguns termos, que podem ser confundidos e utilizados de maneiras distintas. Nesta dissertação os significados de alguns desses termos (DELAMARO, et al., 2007) é resumido a seguir:

- Engano: Ação humana que produz um defeito.
- Falha: Passo, processo ou definição de dados incorretos.
- Erro: Estado inconsistente ou inesperado gerado por uma falha durante a execução do programa.
- Defeito: Fato ocorrido por conta do erro, que gera um resultado diferente do resultado esperado.

As atividades de testes, tais como criação de casos de testes, planos de testes, registros de defeitos, consumem boa parte dos recursos de um projeto de software, por isso é importante que seja feito um gerenciamento dos testes, para aumentar a eficiência e a qualidade dos testes realizados.

2.3.1. Gerenciamento de testes

Peter Ducker (apud BASTOS, 2007) disse: “O planejamento não é uma tentativa de predizer o que vai acontecer. O planejamento é um instrumento

para raciocinar agora, sobre que trabalhos e ações serão necessários hoje, para merecermos um futuro. O produto final do planejamento não é a informação: é sempre trabalho".

Para o sucesso de uma bateria de testes de software, um testador precisa saber planejar as suas atividades. O fruto desse planejamento é o plano de testes, que descreve o que fazer e em função do que você precisa fazer.

Um plano de testes é um conjunto de estratégias que guiam ou representam o processo de testes. Muitas dessas estratégias são documentadas parcialmente, em múltiplos documentos, e estão sujeitas a mudanças conforme o projeto evolui (KANER, et al., 2001).

Um artefato importante no plano de testes são os casos de testes (SANTHANAM, et al., 2002). Os casos de teste servem para definir como o produto será testado e pode ser visto como um roteiro a ser seguido pelo testador. Um ou mais requisitos de testes podem ser avaliados por um caso de teste.

O planejamento dos testes define também quando os testes serão executados, qual a estrutura e os ambientes necessários, onde e como armazenar toda a informação gerada durante o ciclo de testes.

Existem diversas maneiras de realizar a criação de um conjunto de casos de testes. Uma das maneiras é utilizar um modelo que representa o comportamento do software, conforme será apresentado na seção 2.3.3.

2.3.2. Gerenciamento de defeitos

O gerenciamento dos defeitos é definido em (MOLINARI, 2008) como um conjunto de processos e procedimentos que visa armazenar e gerenciar a

informação a respeito dos defeitos encontrados ao longo do ciclo de vida de um software, desde o projeto até a sua retirada ou saída de produção.

Um processo básico (MOLINARI, 2008) para o gerenciamento dos defeitos consiste em:

1) Detectar o defeito. Um relatório do defeito com informações com Identificação do defeito, descrição, prioridade, provas e evidências da sua existência deve ser elaborado.
2) Validar o relato do defeito. Uma analise é feita e se for constatado que o defeito ocorre, o mesmo será aceito para correção.
3) Atribuir a um desenvolvedor para correção.
4) Realizar a correção no código do software.
5) Testar o software novamente para verificar se o defeito realmente foi corrigido.
6) Se o defeito foi corrigido, a ocorrência é fechada.

As principais informações de um Gerenciamento de Defeitos são: Identificação do defeito, Descrição, Severidade, Prioridade, Risco associado, Status, Provas e evidências da existência do defeito.

2.3.3. Teste baseados em modelo

O teste baseado em modelos, do inglês, *Model-based Test* (MBT), é uma técnica que gera testes de software a partir de descrições explícitas de comportamento de um aplicativo (ROBINSON, 1999). Um modelo é uma descrição do comportamento de um sistema. Como os modelos são mais simples do que os sistemas que os descrevem, eles podem nos ajudar a entender e prever o comportamento do sistema.

Existem várias formas de modelar um sistema, tais como Diagrama de caso de uso, *StateCharts* e Máquinas de estados finitas (MEFs), essa última é uma técnica bastante utilizada e serve para especificação do aspecto comportamental de sistemas reativos, particularmente, na área de protocolos de comunicação.

O sistema modelado em uma MEF é descrito por uma máquina, composto de estados e transições, estando em somente um de seus estados num dado momento (HOPKROFT, 1979). O estado armazena informações sobre o passado, já as transições indicam mudanças de estado.

Uma MEF (GILL, 1962) é definida como uma tupla <X,Z,S,so,fz,fs>, sendo que:

- X - um conjunto não vazio de símbolos de entrada;
- Z - um conjunto finito de símbolos de saída;
- S - um conjunto finito não-vazio de estados;
- so - o estado inicial;
- fz - (S x X) → z é a função se saída;
- fs - (S x X) → s é a função de próximo estado.

Existe também a Máquina de Estado Finitos Estendidos (MEFE), que inclui variáveis de contexto, predições e ações. Uma transição de uma MEFE não é caracterizada somente pelo estado de origem, estado de destino e interação (ou evento) de entrada, como em uma MEF, mas também por predicados e ações (HOPKROFT, 1979).

Uma MEFE é representada como uma 8-tupla <S, so, I, O, V, P, A, g>, onde:

- S é o conjunto não vazio de estados;
- so é o estado inicial;

- I é o conjunto finito de estados;
- O é o conjunto finito de saídas;
- V é o conjunto de variáveis;
- P é o conjunto de predicados que operam sobre as variáveis;
- A é o conjunto de ações relacionadas às variáveis;
- g é a função de transição de estado definida como g: S x I x P(V) → S x O x A(V).

A representação de sistemas através de modelos auxilia a preparação dos testes, permitindo que a equipe de testes já conheça o funcionamento do software e inicie a construção de casos de testes manualmente ou automaticamente através de uma ferramenta dedicada, antes mesmo do software ser construído.

3 TRABALHOS RELACIONADOS

A busca de soluções para auxiliar a realização de testes de software tem sido alvo de várias pesquisas e desenvolvimentos. A utilização de ferramentas de software livre, capazes de atender situações específicas e parte do processo de verificação e validação de software, tem sido buscada por organizações de todos os tipos.

Este capítulo apresenta alguns estudos e ferramentas criadas para apoiar as atividades de testes de software.

O site OpenSourceTesting.org (OPENSOURCETESTING.ORG, 2012) disponibiliza discussões, noticias e uma extensa lista de ferramentas livres para as atividades de testes. Existem ferramentas de gerenciamento de testes, de defeitos, de testes de desempenho, de testes funcionais e de unidade.

Existem algumas ferramentas comerciais robustas e de alto custo, que também gerenciam o processo de testes. Alguns exemplos dessas ferramentas dedicadas são: a Rational TestManager (INTERNATIONAL BUSINESS MACHINES, 2012), a HP Test Management (HEWLETT-PACKARD, 2012) e a Visual Studio Test Professional (MICROSOFT, 2012).

Dentre as opções de ferramentas de baixo custo e que atendam ao propósito do ambiente proposto neste trabalho, nenhum outro estudo ou ferramenta que abordam esses conceitos de maneira integrada foi encontrado.

Porém vários trabalhos tratam os conceitos de maneira isolada e se propõem a atender uma determinada necessidade específica. Em alguns trabalhos, as ferramentas são apresentadas como forma de atender às metas estabelecidas por modelos de referência de desenvolvimento, como por exemplo, a MPS.br.

No trabalho de (CARDIAS JUNIOR, et al., 2010) encontra-se uma análise avaliativa de ferramentas livres usadas na implantação do processo de gerências de requisitos da MPS.br. Entre essas ferramentas, a Mantis é testada e aprovada para atender alguns resultados esperados deste processo. O autor relata em sua conclusão que as ferramentas utilizadas foram utilizadas de maneira isolada e não integradas, o que pode inviabilizar a utilização em projetos grandes.

Outro trabalho que utiliza a ferramenta Mantis é o de (BÉZIVIN, et al.,2005). Nele é proposto um metamodelo de dados para explicar e extrair informações de várias ferramentas e situações. No trabalho ele utiliza as ferramentas BugZila, Mantis e planilhas do Excel. Através da modelagem das informações, são gerados arquivos XMLs no formato do metamodelo definido.

Essa abordagem tem como proposta a interoperabilidade das informações. Na conclusão do trabalho é relatado que as informações adquiridas em diferentes projetos, com infraestrutura distintas, poderão ser reutilizadas em projetos de contextos semelhantes, porém não explica como esses dados serão unificados e consumidos.

No trabalho de Braga et al. (BRAGA, et al., 2010) é feito um estudo de como gerenciar o conhecimento dos testadores em um projeto de testes exploratórios. O autor cita, no inicio do trabalho, que é difícil extrair o conhecimento das pessoas e passá-lo para a forma escrita. Em toda sua experimentação, utiliza a integração das ferramentas Mantis e do TestLink, para relacionar os defeitos encontrados aos casos de testes. Porém não inclui uma camada que integre ambas as ferramentas em um único banco de dados.

Em (BRAGA, et al., 2010) a conclusão relatada é que a atividade de externalizar o conhecimento do testador é realmente complicado. Algumas limitações das ferramentas utilizadas são também apresentadas. No caso da

TestLink, a ferramenta não permite que casos de testes criados em projetos distintos sejam consultados, o que na opinião dele, dificulta na disseminação da informação. Por fim, o trabalho ainda cita que o processo de registro dos testes não foi burocrático, pelo contrário foi feito com simplicidade e eficiência, permitindo que o testador focasse a sua atenção no teste e não no registro.

A atividade de geração automática de casos de testes é um problema conhecido, pelo esforço necessário para realização dessa atividade manual. (MANINI, 2009) propõe em seu trabalho uma ferramenta de apoio ao teste baseado em casos de uso chamado Héstia. O autor compara a sua ferramenta com a TestLink, onde ele ressalta que a TestLink não possui um mecanismo capaz de mapear e gerenciar automaticamente casos de uso como casos de testes. A Héstia não se estende ao gerenciamento de defeitos.

Existem também alguns trabalhos que procuram integrar o planejamento dos testes com a automação da execução dos testes, como por exemplo, o de (COLLINS, et al., 2010) que apresenta um relato de experiência da integração das ferramentas TestLink, Mantis e Selenium em um ambiente de testes utilizando a abordagem ágil chamada SCRUM. No relato é apresentado que a integração trouxe bons resultados ao projeto em que foi utilizado.

Como podem ser observadas, as características do software em testes são muito importantes para a definição das ferramentas a serem usadas. Além disso, existem várias ferramentas livres que podem auxiliar nas atividades do processo de testes.

A proposta deste trabalho considera as características de um laboratório de verificação e validação de software e propõe uma base de conhecimento de testes para armazenar todas as informações geradas durante os projetos e prevê re-utilização de casos de teste e de resultados de projetos passados.

4 FERRAMENTAS UTILIZADAS

O ambiente VVTeste integra três ferramentas de teste que apoiam diferentes atividades de verificação, validação e teste de software: geração de casos de teste, gerenciamento de teste e gerenciamento de defeitos. Este capítulo descreve as ferramentas integradas no ambiente VVTeste.

4.1. Ferramenta de geração automática de casos de testes

A criação de casos de testes é uma tarefa custosa e pode ser bastante complexa de acordo com o sistema em testes (SUT). Além disso, a cobertura dos testes é essencial na garantia da qualidade do software. Existem na comunidade vários esforços para melhorar e ajudar nessa atividade de testes. Para tanto, algumas ferramentas dedicadas são capazes de gerar os casos de testes através de modelos pré-definidos.

No ambiente VVTeste, que visa principalmente o apoio a software embarcado, a ferramenta CONDADO se torna uma solução “caseira” para o ambiente, pois foi desenvolvida em conjunto entre o INPE e a Universidade Estadual de Campinas (UNICAMP), e recentemente tem recebido colaborações da Universidade Federal de Lavras (UFLA).

A CONDADO foi desenvolvida para a geração automática de casos de teste para sistemas de comunicação de protocolo baseado em Máquina de estado finitas estendida (MEFE). A abordagem adotada pela ferramenta, com a combinação de três testes de caixa preta - teste de transição de estados, teste de sintaxe e teste de domínio - possibilita a geração de teste cobrindo a parte do controle e os dados dos parâmetros de interações do protocolo (MARTINS, et al., 1999).

Para possibilitar a especificação do modelo de teste (uma MEF, a partir da qual são gerados os casos de teste) foi definida uma linguagem chamada de Linguagem de Especificação de Protocolos (LEP) permitindo então que a CONDADO interprete a especificação do modelo, isto é, uma especificação de teste, e em seguida, gere os casos de teste.

A especificação em LEP é convertida em uma especificação de teste cujo formato é baseado em Cláusulas de Horn, as quais são interpretadas por um programa em linguagem Prolog. A conversão é obtida através de serviços de um analisador e na satisfação dos requisitos da especificação transformada em dados e transições. Em seguida, juntamente com restrições definidas pelo usuário, um gerador utiliza o conjunto de cláusulas implementadas em Prolog para gerar sequências de teste as quais englobam controle e dados (ATIFS, 2005).

A saída gerada pela ferramenta CONDADO é um script de teste contendo uma sequência de entradas e suas saídas respectivas. Um conjunto de entradas e saídas formado por um caminho de transições entre os estados existentes na MEFE constitui um caso de teste. O padrão empregado define os comandos de entrada como *input* e os de saída como *output*, como pode ser visto no Apêndice G.

4.2. Ferramenta de gerenciamento de testes

O gerenciamento dos testes gera uma grande quantidade de informações, que são aproveitadas de maneira mais eficiente quando manipuladas através de uma ferramenta dedicada.

Para que uma ferramenta possa ser utilizada no gerenciamento dos testes, ela deve atender basicamente os seguintes requisitos (BSTQB, 2005):

- Dar apoio ao gerenciamento de testes e a suas atividades.
- Fazer a interface entre as ferramentas de execução, ferramentas de gerenciamento de defeitos e ferramentas de gerenciamento de requisitos.
- Prover controle de versão independente ou uma interface com o gerenciador de configuração.
- Dar apoio a rastreabilidade do teste e seus respectivos resultados.
- Registrar os resultados e gerar o relatório de progresso do teste.
- Prover análise quantitativa (estatísticas), relacionadas aos testes (ex: testes executados e testes que passaram) e aos objetos de teste (ex: incidentes levantados), visando fornecer informações sobre o objeto de teste.

Uma ferramenta capaz de atender os requisitos para gerenciamento de teste é a TestLink. A TestLink foi escolhida porque permite a inclusão do gerenciamento de requisitos de testes, permitindo a associação dos requisitos ao caso de testes (TESTLINKCOMMUNITY, 2005). Outras características importantes da ferramenta são:

- A ferramenta é livre e está em constante evolução;
- Permite a criação de campos personalizados;
- É uma ferramenta Web, por isso é multi-plaforma (Windows, Mac, Linux, etc);
- Permite a definição de plano de testes, suíte de testes e caso de testes;
- Gerencia a execução dos testes;
- Possui uma interface amigável e autoexplicativa. A cada passo do projeto, o sistema explica como realizar aquela operação na própria página.
- Permite a importação de casos de testes.

Além de gerenciar as informações do projeto de teste, que podem ser personalizadas de acordo com a necessidade, o TestLink também disponibiliza vários relatórios e gráficos sobre o desempenho dos testes. Com uma interface de fácil utilização, se torna uma opção eficiente para um laboratório de V&V de software.

Esta ferramenta encontra-se atualmente na versão 1.9 e é mantida pela comunidade “TestLink Community”. A ferramenta está disponível para download no site http://www.teamst.org/ desde 2005 e tem sido melhorada ao longo dos anos.

4.3. Ferramenta de gerenciamento de defeito

Sendo a detecção de defeitos uma das finalidades dos testes de software, todos os resultados alcançados que sejam diferentes do esperado devem ser registrados. Este acontecimento também é chamado de incidente e precisa ser rastreável desde descoberta até a sua correção.

Durante a execução dos testes os defeitos encontrados precisam ser reportados a equipe desenvolvedora, descrevendo detalhes do problema encontrado, evidências e passos para reprodução.

Uma ferramenta de gerenciamento de defeitos deve funcionar como um repositório de tais informações e auxiliar na priorização, na atribuição de tarefas (ex: corrigir o defeito ou executar um teste de confirmação) e na atribuição de status (rejeitado, pronto para teste ou duplicado).

Estas ferramentas permitem que o progresso do defeito seja monitorado durante o projeto. Muitas vezes proveem o apoio para análises estatísticas e relatórios.

O ambiente VVTeste integra a ferramenta Mantis para apoio às atividades de gerenciamento de defeitos de software. A Mantis foi escolhida por:

- Permitir a criação de campos personalizados.
- Personalizar o processo de gerenciamento de defeitos.
- Ser multiplataforma (Funciona em Windows, Linux e outros sistemas operacionais).
- Trabalhar com banco de dados gratuito (MYSQL).
- Permitir a integração com ferramentas de gerenciamento de testes, como o TestLink.
- Permitir o controle de acesso e perfil de usuário.
- Exibir estatísticas dos defeitos encontrados.
- Ser uma ferramenta Web, desenvolvida em PHP.
- Estar em constante evolução.
- Dispor de interface amigável.

A Mantis é desenvolvida e mantida pela comunidade MantisBT. Esta ferramenta começou a ser criado em 2000 (MANTISBT, 2000), como solução para o gerenciamento de defeitos e tem sofrido melhorias e evoluções até os dias atuais.

Esta ferramenta é livre tanto para utilização como para evolução, permitindo que cada empresa personalize a ferramenta as suas necessidades. Atualmente ela se encontra na versão 1.2.8 e pode ser adquirida através do site: http://www.mantisbt.org/download.php.

5 VVTESTE

O ambiente VVTeste tem por finalidade auxiliar as atividades de testes existentes nos processos de Verificação e Validação de software.

Este capítulo mostra quais as atividades dos processos são atendidas pelas ferramentas do ambiente VVTeste, como elas são integradas e utilizadas, além de apresentar uma análise sobre o atendimento dos resultados esperados pelos modelos de desenvolvimento de software CMMI e MPS.br.

5.1. Visão geral

O processo de testes proposto em (MOREIRA FILHO, et al., 2003) basicamente é composto de cinco atividades, conforme Figura 5.1: Planejamento, Preparação, Especificação, Execução e Entrega.

Figura 5.1 - Processo básico de gerenciamento de testes.
Fonte: Adaptado de Moreira Filho, et al. (2003)

O planejamento dos testes consiste em definir as estratégias para os testes, criar o plano de testes, com as algumas informações dos dados que serão necessários, quando (em que fase) os testes serão realizados, como eles serão verificados e quais os riscos associados a este projeto.

Na etapa de preparação, as principais atividades são: (i) definir a arquitetura para os testes, (ii) verificar se haverá ferramentas de apoio, e caso tenha, configurá-las e (iii) preparar o pessoal, em relação ao conhecimento no sistema.

Durante a especificação são criados os casos de testes, que servirão como guia durante os testes de cada operação do sistema, essa etapa é uma das mais complexas do projeto, pois os casos gerados devem cobrir todo o sistema.

Na quarta etapa, durante a execução dos testes, o ambiente é preparado para que os casos de testes sejam executados contra o sistema em testes, e os resultados dos testes são registrados.

Por fim, a entrega do projeto de testes, onde os documentos e relatórios dos testes são arquivados e utilizados conforme definido no projeto do software.

Este processo de teste gera uma grande quantidade de informações e de documentos que precisam ser organizados e gerenciados de maneira segura e eficiente. Sendo assim, a utilização de uma ferramenta dedicada se torna indispensável. A alternativa encontrada e que atende as necessidades deste ambiente é a TestLink, que permite o gerenciamento das informações desde planejamento até a entrega do relatório de testes.

As funcionalidades existentes na TestLink permitem cobrir todas as cinco atividades definidas no processo de testes. A Figura 5.2 representa a cobertura da ferramenta durante a gerência dos testes.

Figura 5.2 – Representação do processo de gerenciamento de testes utilizando a TestLink.

Entretanto, a Testlink não orienta um testador a conceber ou especificar casos de teste. Por este motivo, a ferramenta CONDADO é incorporada na VVTeste para apoiar a geração automática de casos de testes.

A automatização da especificação dos casos de testes através da CONDADO não limita a criação de outros tipos de casos de testes diretamente no TestLink pelos analistas de testes. Inclusive, é possível que outras ferramentas também possam gerar casos de testes automáticos que podem ser inseridos dentro do mesmo projeto, permitindo interação e flexibilidade entre diferentes equipes de testes.

A utilização da CONDADO e a integração com a TestLink, auxiliam na realização da atividade de Especificação do processo de testes, conforme apresentado na Figura 5.3.

Figura 5.3 – Ferramenta CONDADO como apoio a atividade de Especificação do processo de testes.

Outra atividade importante em um processo de Verificação e Validação é o gerenciamento dos defeitos encontrados durante a execução dos testes no software. Um controle eficiente dos defeitos é essencial para a qualidade e a segurança do software.

O gerenciamento de defeitos possui atividades próprias: abertura, admissão, atribuição, resolução e fechamento (BASTOS, 2007). Estas atividades podem

ser vistas como etapas de um processo de gerenciamento de defeitos, conforme apresentado na Figura 5.4.

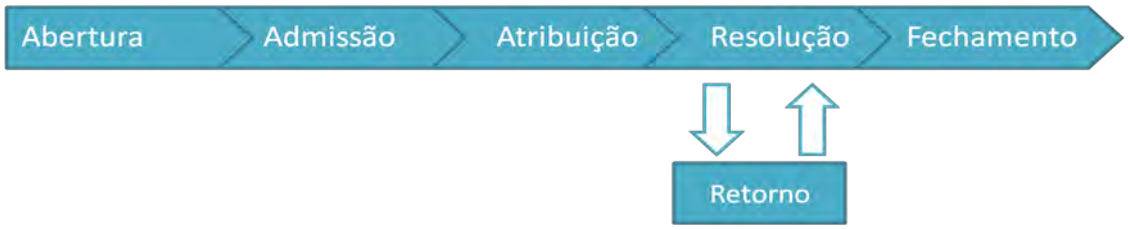

Figura 5.4 - Etapas do gerenciamento de defeitos.
Fonte: Adaptado de Bastos (2007)

O gerenciamento do defeito deve ser feito desde a primeira detecção até a sua resolução, sendo necessário registrar informações sobre a sua gravidade, sua intermitência, causas que ocasionaram o defeito, pessoas envolvidas e diversas outras.

A primeira etapa do gerenciamento de um defeito é a detecção do resultado não esperado. Durante a execução do teste, o responsável pelo teste avalia os resultados obtidos contra o esperado. Uma vez constatada a desigualdade, o testador define as informações do defeito, como, por exemplo: tipo, os passos para reprodução e a gravidade.

Um responsável pelo desenvolvimento ou pelo projeto do software avalia o defeito descrito e caso se confirme, o defeito passa a estar no estado “Admitido”.

Na terceira etapa, o defeito é atribuído a um determinado responsável, que deverá encontrar uma solução. Este usará as informações fornecidas pelo testador para localizar o defeito.

Após a correção, o responsável deve definir o status do defeito para Resolvido, descrevendo o que foi feito, as causas e outras informações que possam ser

úteis. O testador realiza um novo teste e se constatar que o problema não ocorre mais, muda o estado do defeito para “fechado”.

Caso o testador detecte que o problema não foi resolvido, o mesmo retorna o incidente ao responsável pela correção descrevendo os motivos da rejeição. Assim, retornando a etapa de correção do defeito e posteriormente o envio para o re-teste do testador.

A ferramenta Mantis cobre todas as etapas do processo de gerenciamento de defeitos mostradas na Figura 5.4. Porém, um ponto importante, tanto para o defeito como para o caso teste, é que eles precisam estar associados. O caso de testes precisa assumir o status de “Com falha” e ser associado aos defeitos encontrados. Este status (“Com falha”) indica que o caso de teste foi capaz de detectar um defeito no software em teste.

A TestLink permite (através de seus arquivos de configuração) a integração com ferramentas de gerenciamento de defeitos, permitindo registrar os códigos dos defeitos de forma a criar links para as interfaces da Mantis.

A Figura 5.5 mostra como o processo de gerência de defeitos é integrado ao processo de gerência de teste e em quais atividades as ferramentas escolhidas nessa dissertação dão apoio ao processo.

A definição das atividades do processo de testes associadas às ferramentas dedicadas são à base do ambiente VVTeste proposto. O ambiente VVTeste contempla também a configuração de cada ferramenta de acordo com as necessidades de informação encontrada, a integração de todas as ferramentas e a criação de uma base de conhecimento que irá consumir todas as informações geradas nas ferramentas, disponibilizando em um único local dados para lições aprendidas, estatísticas, comparações entre projetos e planejamento de projetos futuros.

Figura 5.5 – Integração dos processos de gerenciamentos de testes e defeitos, e as ferramentas de apoio.

A seguir será apresentada a arquitetura desenvolvida para o ambiente VVTeste.

5.2. Arquitetura de integração da VVTeste

As ferramentas TestLink e Mantis já possuem facilidades para integração nativa. Com algumas alterações em seus arquivos de configuração as duas ferramentas são integradas, permitindo que durante a execução dos testes, o testador faça a associação com um ou vários defeitos que estão registrados na Mantis. Já a integração da ferramenta CONDADO com as outras ferramentas exigiu algum esforço.

O ambiente VVTeste foi concebido em 4 módulos: (i) Módulo Integração Condado x TestLink (MICT), (ii) Módulo Inclusão de dados (MID), (iii) Módulo Aquisição de dados (MAD) e (iv) Módulo Consulta de dados (MCD). Além dos módulos, o ambiente tem uma base de conhecimento única para armazenar

todas as informações adquiridas durante os projetos de testes. A Figura 5.6 apresenta a arquitetura do ambiente VVTeste.

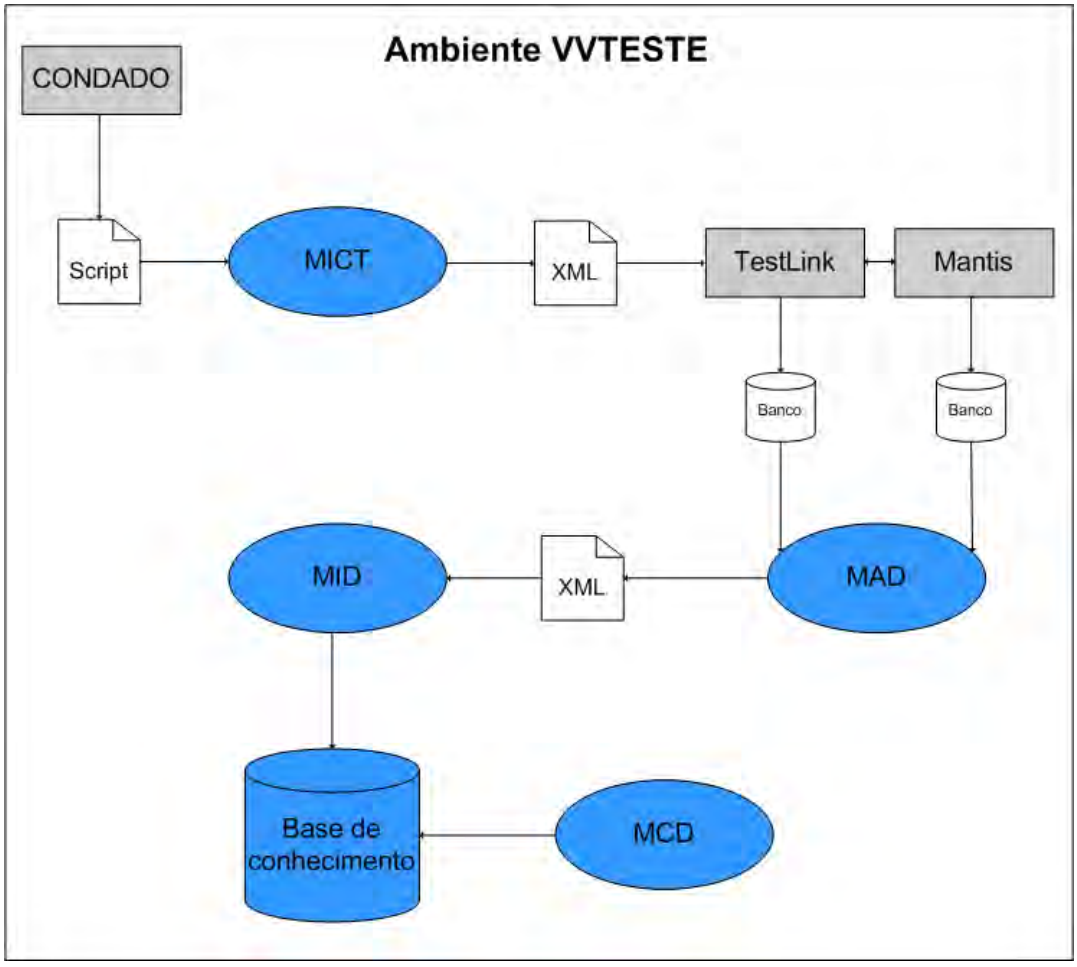

Figura 5.6 - Arquitetura de integração da VVTeste.

As caixas em cinzas representam as ferramentas existentes utilizadas pelo ambiente. As figuras em branco representam os arquivos ou as fontes de informações que são utilizadas para a integração das ferramentas e os módulos criados para a integração das ferramentas no ambiente VVTeste estão representados por círculos azuis.

O ambiente proposto foi modelado utilizando o principio básico de interoperabilidade, por isso a manipulação das informações é feita com

arquivos XML, facilitando a integração futura de outras ferramentas a este ambiente.

O módulo MICT é responsável pela transformação do script de teste gerado pela Condado em um XML, que posteriormente é importado na TestLink.

O MAD é o responsável por obter as informações nos bancos de dados das ferramentas Mantis e TestLink e gerar um conjunto de arquivos XMLs com todas essas informações. Já o MID faz o processamento dos XMLs gerados pelo MAD e inclui as informações obtidas na Base de conhecimento. O MAD e o MID foram desenvolvidos separadamente para permitir interoperabilidade ao ambiente, pois assim outras ferramentas também podem adicionar informações a base de conhecimento.

A base de conhecimento unifica as informações geradas durante a gerência dos testes e o controle dos defeitos, além de permitir que essas informações sejam cruzadas e consumidas de uma forma que não seria possível com as ferramentas isoladas. Já o MCD é uma ferramenta capaz de executar comandos SQL sobre a base de conhecimento e transformar o resultado em gráficos de colunas ou de pizza.

A seguir serão detalhadas todas as configurações realizadas nas ferramentas selecionadas e em seguida, é apresentado como cada módulo foi criado pelo VVTeste.

5.3. Configuração das ferramentas

As ferramentas TestLink e a Mantis podem ser ajustadas de acordo com as necessidades de um determinado ambiente de testes. Elas permitem a criação de campos personalizados, ampliando a cobertura das informações desejadas por uma equipe de testes.

Durante a análise das ferramentas e das necessidades do ambiente de V&V de software embarcado proposta pela VVTeste, chegou-se a um conjunto de personalizações a serem feitas em cada uma das ferramentas, com a finalidade de permitir o registro de todas as informações desejadas.

5.3.1. Personalização da TestLink

Todos os campos criados foram definidos a partir da consulta de normas e bibliografia especializada e de acordo com a importância da informação para o ambiente VVTeste.

Devido à ausência de um consenso no significado de alguns termos na área de testes de software, os termos como, tipo e critério de teste foram definidos para a VVTeste:

- Tipos de testes: Refere-se à visão do testador no momento do teste. Neste caso, os tipos são classificados como Caixa Branca (visão em nível de código), Caixa Preta (visão externa ou da funcionalidade) e Caixa Cinza (Uma visão intermediária entre caixa preta e caixa branca).
- Técnicas de testes: Também chamadas de critérios de testes, serão separadas de acordo com o seu tipo de testes. Existem inúmeras técnicas de testes e por isso, foram selecionadas as técnicas mais comuns, tais como Baseados em Modelos e Análise de valor limite.

É importante ressaltar que no ambiente VVTeste os valores dos campos podem ser alterados de acordo com a necessidade de uso. Um exemplo claro para esta prática é a definição das ferramentas de testes, que são as ferramentas que darão apoio durante a execução dos casos de testes. Como padrão inicial para a VVTeste, os valores Ferramenta 1, Ferramenta 2 e Ferramenta 3 são definidos e a equipe de testes tem a liberdade de alterar tais valores.

A VVTeste define a criação de 9 campos personalizados na TestLink, campos estes que serão utilizados na especificação de requisitos e de casos de testes e na execução dos testes. A Tabela 5.1 apresenta todos os campos personalizados.

Tabela 5.1 - Campos personalizados na TestLink.

Nome	Rótulo	Tipo	Valores	Utilização
ExecucaoTeste	Execução de teste	Radio	Automática e Manual	Especificação do caso de teste
FaseTeste	Fase de teste	Radio	Integração, Unidade, Aceitação e Sistema	Especificação do caso de teste
FerramentaTeste	Ferramenta de teste	Radio	Ferramenta 1, Ferramenta 2 e Ferramenta 3	Especificação do caso de teste
TecnicaTesteCB	Técnica de teste de caixa branca	Lista de multi seleção	Fluxo de controle, Fluxo de dados, Por complexidade, Potenciais - Usos Cobertura de comandos e Cobertura de decisão	Especificação do caso de teste
TecnicaTesteCP	Técnica de teste de caixa preta	Lista de multi seleção	Análise de valor limite, Funcional sistemático, Grafo de Causa - Efeito, Partição de equivalência, Baseado em caso de uso, Baseado em modelos, Tabela de decisão e Transição de estado	Especificação do caso de teste
TempoEstimado	Tempo estimado	Número	Qualquer valor numérico inteiro	Especificação do Caso de teste
TempoGasto	Tempo gasto	Número	Qualquer valor numérico inteiro	Resultado do teste
TipoRequisito	Tipo de requisito	Radio	Caso de uso, Interface, Funcional e Não funcional	Especificação do Requisito
TipoTeste	Tipo de teste	Radio	Caixa Branca, Caixa Cinza e Caixa Preta	Especificação do Caso de teste

As informações referentes ao Nome, tipo e o local onde o campo personalizado é ativado não podem ser diferentes dos apresentados na Tabela 5.1, pois é através delas que o módulo VVTeste – MAD (conforme seção 5.3.2) irá consumir essas informações.

Em relação aos tipos, um campo definido como “Radio” permite que apenas um dos valores possíveis seja selecionado, já um campo “Lista de multiseleção” permite a seleção de mais de uma opção e o campo “Número” aceita qualquer número inteiro.

5.3.2. Personalização da Mantis

A ferramenta dedicada ao gerenciamento de defeitos Mantis foi personalizada com a inclusão de algumas novas informações. Neste caso, a norma IEEE 1044 (IEEE 1044, 2009) que define padrões para classificação de defeitos de software foi utilizada como referência para definição dos valores dos campos.

Na sua versão padrão, a Mantis possui vários campos que são utilizados pelo VVTeste por atenderem a norma IEEE 1044 e pelo requisito que o ambiente pretende atender. Alguns valores inclusive vão além dos especificados pela norma, porém por se tratarem de recurso próprio da ferramenta, tais valores serão considerados no ambiente.

Os campos nativos e seus respectivos valores que são utilizados pelo VVTeste são:

- Gravidade, que define o impacto que o defeito gera no SUT e é classificado como: Recurso, Trivial, Texto, Mínimo, Pequeno, Grande, Travamento e Obstáculo.

- Resolução, que indica o resultado da ocorrência de defeito. A resolução pode ser: Aberto, Corrigido, Reaberto, Incapaz de reproduzir, Não corrigível, Duplicado, Não é um caso, Suspenso e Não será corrigido.
- Frequência, que se refere à incidência do defeito no SUT e pode ser definido como: Sempre, Às vezes, Aleatório, Não se tentou, Incapaz de reproduzir e N/D.
- Estado que define a situação atual do defeito. Seus valores são: Novo, Retorno, Admitido, Confirmado, Atribuído, Resolvido e Fechado.

A categoria do defeito também utiliza um recurso nativo da Mantis, porém ao contrário dos citados anteriormente, este campo não possui valores pré-definidos e por isso são classificados de acordo com a norma IEEE 1044 (IEEE 1044, 2009), que indica como categoria os seguintes valores: Funcionalidade, Manutenção, Desempenho, Segurança, Usabilidade e Outros. Estes valores são cabíveis de alteração e inclusão, caso seja necessário uma nova categoria para os defeitos detectados.

Em relação aos campos personalizados, a VVTeste define a criação de duas novas informações, que são apresentadas na Tabela 5.2.

Tabela 5.2 - Campos personalizados na Mantis

Nome	**Tipo**	**Valores**
Artefato	String	Caracteres alfanuméricos
Causa do defeito	Caixa de marcação	Descrição, Dados, Interface, Lógico, Normas, Sintaxe e Outros

O campo Artefato permite que durante o relato ou a solução do defeito, seja registrada qual a Função, a Classe, o Módulo ou qualquer outra denominação que aponte o local do defeito. Este campo é livre para o preenchimento de qualquer informação alfanumérica.

O campo Causa do Defeito deve representar o motivo que levou ao defeito. Os valores definidos pelo ambiente VVTeste também foram adquiridos através da norma IEEE 1044 (IEEE 1044, 2009). Assim como na ferramenta TestLink, nenhum dos campos personalizados são de preenchimento obrigatório, permitindo que os valores possam ser atualizados ao longo do ciclo de vida do defeito.

5.4. Implementação dos Módulos do Ambiente VVTeste

A seguir serão detalhados como cada módulo foi desenvolvido e como a integração pode ser feita através dessas ferramentas.

5.4.1. Módulo Integração Condado x TestLink (MICT)

Dada uma especificação do comportamento do software em MEFE, o usuário da Condado insere todos os estados e transições na ferramenta, a qual, posteriormente gera um *script* de teste com os casos de testes. Tal *script* possui um padrão bastante simples, com *tags* do tipo *input* para entrada e *output* para a saída esperada. Cada sequência *input/output* é chamada de *step*¸ ou passo do caso de teste.

Para integrar o script de teste gerado pela CONDADO com a ferramenta de gerenciamento de testes, foi desenvolvido o Módulo Integração Condado x TestLink (MICT). O MICT foi desenvolvido em Java e é capaz de interpretar o padrão do script gerado pela Condado e criar um XML com formato reconhecido pela TestLink, permitindo a importação desses casos de testes na ferramenta de gerenciamento de testes.

Este módulo foi desenvolvido com o paradigma da orientação a objetos, utilizando a linguagem de programação Java e a IDE NetBeans versão 6.8. Além disso, foi utilizada a biblioteca XSTREAM, versão 1.4.1, para a serialização dos objetos em arquivo XML.

A interface do MICT (Verificar Apêndice A) define quatros passos para a transformação do script CONDADO em XML:

- Leitura do arquivo de Script da Condado.
- Definição dos dados comuns dos casos de testes, que são: Prefixo, Número inicial da sequência de casos, resumo e pré-requisitos.
- Definição de até cinco palavras chaves.
- Definição do local onde o arquivo XML deve ser salvo.

O arquivo XML gerado pela MICT possui todos os casos de testes pertencentes em um script da CONDADO. Para exemplificar a transformação do script, a Figura 5.7 apresenta as informações referentes a um arquivo gerado pela CONDADO, contendo um caso de teste, que após processado pela MICT irá gerar o arquivo XML apresentado na Figura 5.8.

```
input(L,Liga) output(L,AcenderLuzONOFF)
input(L,Desliga) output(L,ApagarLuzONOFF)
```

Figura 5.7 - Exemplo de Script da Condado

A geração do arquivo XML só pode ser capaz após o mapeamento de todos os *nodes*, também conhecidos como atributos. Essa análise foi feita de acordo com a documentação gerada pelos mantenedores da ferramenta TestLink.

A integração da TestLink permite que diversos atributos sejam utilizados, porém apenas os que utilizados no VVTeste foram mapeados e estão apresentados na Tabela 5.3, sendo que na coluna TAG é apresentado o nome do atributo e na coluna Comentário um breve resumo de como a informação é obtida e/ou utilizada no ambiente.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<testcases>
<testcase name="CASOTESTE1">
  <summary>Caso de testes do equipamento</summary>
  <preconditions>O equipamento deve estar ligado na tomada</preconditions>
  <steps>
    <step>
      <actions>input(L,Liga)</actions>
      <expectedresults>output(L,AcenderLuzONOFF)</expectedresults>
      <execution_type>1</execution_type>
      <step_number>1</step_number>
    </step>
    <step>
      <actions>input(L,Desliga)</actions>
      <expectedresults>output(L,ApagarLuzONOFF)</expectedresults>
      <execution_type>1</execution_type>
      <step_number>2</step_number>
    </step>
  </steps>
  <keywords>
    <keyword name="Equipamento teste">
      <notes>Modelagem do equipamento teste</notes>
    </keyword>
  </keywords>
</testcase>
</testcases>
```

Figura 5.8 - Exemplo do XML gerado pelo VVTeste - Módulo MICT.

Tabela 5.3 - Mapeamento do *node* do XML gerado pela MICT.

TAG	Comentário
<?xml version="1.0" encoding="UTF-8"?>	Valor fixo no arquivo.
<testcases>	*Node* pai. Permitirá que 1 ou vários casos de testes estejam descritos dentro da sua estrutura.
<testcase name="CASOTESTE1">	Inicia um *Node* referente a um caso de teste. O atributo *name* indica o nome do caso de testes, que servirá como identificador na TestLink.
<summary>	Indica o resumo do caso de teste. Essa informação é a mesma em todos os casos de testes do Script e é definida na etapa 2 da MICT.
<preconditions>	Indica as pré-condições necessárias para a

Continua

Tabela 5.3 - Conclusão

	execução dos casos de testes. Essa informação é a mesma em todos os casos de testes do Script e é definida na etapa 2 da MICT.
<steps>	*Node* pai dos passos definidos para o caso de testes.
<step>	Define um passo do caso de teste, com entrada, resultado esperado, ordem e tipo de execução.
<actions>	Define a entrada ou estimulo do passo.
<expectedresults>	Define o resultado esperado para o estimulo dado pelo passo.
<execution_type>	Valor fixo. Se refere à forma de execução do passo no SUT. O valor 1 significa a Manual.
<step_number>	Número do passo. Refere-se à ordem de execução dos passos. Um mesmo caso de testes não pode ter dois ou mais número de passos iguais.
<keywords>	*Node* pai. Permitirá que 1 ou várias palavras chaves sejam associadas aos casos de testes.
<keyword name="Equipamento teste">	Inicia um *Node* referente a uma palavra chave. O atributo *name* indica o termo referente à palavra chave. Essa informação é a mesma em todos os casos de testes do Script e é definida na etapa 3 da MICT.
<notes>	Comentário da palavra chave.

Com o XML gerado pela MICT, é possível realizar a última etapa, que se refere à importação deste arquivo TestLink. A ferramenta TestLink já possui como recurso nativo de importação de casos de testes em formato XML e disponibiliza uma ação em sua interface para que os usuários façam esta importação. Para isso, o usuário deve acionar a ação Editar Casos de testes, escolher uma suíte de testes e Importar os Casos de Testes. A ferramenta irá abrir uma página para que o arquivo XML gerado pelo módulo MICT seja selecionado.

5.4.2. Módulo Aquisição de Dados (MAD)

Cada uma das ferramentas TestLink e Mantis possui um banco de dados independente, com padrões e características distintas. O módulo Aquisição de Dados (MAD) tem a finalidade de extrair todas as informações que serão inseridas na base de conhecimento VVTeste, de cada um desses bancos de dados.

O MAD faz a leitura do banco de dados, consome as informações desejadas e as exporta em arquivos XML, que posteriormente são consumidas pelo módulo Inclusão de Dados (MID) na inserção dos dados na base de conhecimento do VVTeste.

A aquisição das informações é feita em duas partes, sendo que cada uma delas gera dois arquivos XMLs. A primeira parte é a responsável pela aquisição das informações registradas no banco de dados da TestLink, ou seja, as informações referentes ao gerenciamento dos testes e a segunda faz a extração das informações dos defeitos no banco de dados da Mantis.

A TestLink possui um banco de dados relacional, apoiado pelo SGBD MySQL, onde todas as informações de requisitos, projetos, suítes, planos e casos de testes são armazenados. Na exportação são gerados 2 arquivos XML, um com as informações dos testes e outro com os dados dos usuários cadastrados na ferramenta.

Para a exportação dos campos personalizados da TestLink, é essencial que os campos tenham sido criados com os mesmos nomes apresentados na seção 5.2.1. As demais informações são adquiridas de forma padrão no banco de dados.

A Figura 5.9 apresenta o XML das informações do gerenciamento de testes, no nível mais alto, apresentando basicamente seis partes para cada projeto de teste:

- Informações gerais do projeto de teste;
- Plataformas de testes;
- Palavras chaves;
- Requisitos;
- Planos de testes;
- Suítes de testes.

As informações referentes ao projeto de testes são: ID, Descrição e Nome do projeto. Tais informações são adquiridas das tabelas *nodes_hierarchy* e *testprojects.* Por se tratarem de informações únicas em um projeto, foram incluídas no nível mais alto do XML.

```
<?xml version="1.0" encoding="UTF-8"?>
<Projetos>
<ProjetoTeste>
  <ID>1</ID>
  <Descricao>Este projeto visa criar o ambiente de testes para a VVTeste</Descricao>
  <Nome>Exemplo VVTeste</Nome>
  <PlataformasTestes>
  <PalavrasChave>
  <Requisitos>
  <PlanosTestes>
  <SuitesTestes>
</ProjetoTeste>
</Projetos>
```

Figura 5.9 - XML em alto nível gerado pela MAD.

Um projeto de teste utiliza uma ou várias plataformas durante a realização dos testes. Durante o planejamento dos testes é definido em qual plataforma os casos de testes serão executados, sendo que um mesmo caso de teste pode inclusive ser executado em mais de uma plataforma de teste. Como essa associação entre resultados de testes e plataforma ocorrem várias vezes, os

detalhes das plataformas são apresentadas nessa seção e posteriormente são apenas associadas através do ID. As informações detalhadas são consumidas da tabela *platforms* e possuem os seguintes campos: ID, Descrição e Nome, conforme apresentado na Figura 5.10.

```
<PlataformasTestes>
  <PlataformaTeste>
    <ID>3</ID>
    <Nome>EquipamentoTeste</Nome>
    <Descricao>Equipamento desenvolvido para realização de testes.
    Não reproduz totalmente o produto final</Descricao>
  </PlataformaTeste>
  <PlataformaTeste>
    <ID>4</ID>
    <Nome>ProdutoFinal</Nome>
    <Descricao>Este é o produto final</Descricao>
  </PlataformaTeste>
</PlataformasTestes>
```

Figura 5.10 - XML das plataformas de testes.

Assim como as plataformas, um projeto de teste também possui várias palavras chaves, que são utilizadas nos casos de testes, auxiliando na associação entre eles. Por isso, a seção do detalhamento das palavras chaves também está no nível mais alto do XML e são associados aos casos de testes apenas pelo ID da palavra. Além do campo identificador, os Nome e Descrição também são exportados e são adquiridos na tabela *keywords*. A Figura 5.11 exibe a parte do XML referente ao detalhamento das palavras chaves.

```
<PalavrasChave>
  <PalavraChave>
    <ID>12</ID>
    <Nome>Palavra 1</Nome>
    <Descricao>Descrição detalhada da Palavra 1</Descricao>
  </PalavraChave>
  <PalavraChave>
    <ID>9</ID>
    <Nome>Palavra 2</Nome>
    <Descricao>A palavra 2 possui uma descrição detalhada</Descricao>
  </PalavraChave>
</PalavrasChave>
```

Figura 5.11 - XML das palavras chaves.

A parte seguinte do XML descreve os requisitos associados ao projeto de testes exportado. Os requisitos podem ser associados a um ou vários casos de testes, por isso, são detalhados no inicio do arquivo e posteriormente são relacionados aos casos através de seu ID. Caso tenha algum requisito descrito que não esteja associado a nenhum caso de testes, ele não é exportado.

Do requisito, são exportadas as seguintes informações, conforme exibido na Figura 5.12: ID, Escopo e Tipo. A identificação é adquirida da tabela *requirements*, já o escopo é buscado na tabela *nodes_hierarchy* e o campo Tipo, por se tratar de um personalizado é armazenado na tabela *cfield_design_values,* através do identificador “TipoRequisito”.

```
<Requisitos>
  <Requisito>
    <ID>Req1.1</ID>
    <Escopo>Escopo do requisito 1.1</Escopo>
    <Tipo>Caso de Uso</Tipo>
  </Requisito>
  <Requisito>
    <ID>Req1.2</ID>
    <Escopo>Escopo do requisito 1.2</Escopo>
    <Tipo>Funcional</Tipo>
  </Requisito>
  <Requisito>
    <ID>Req1.3</ID>
    <Escopo>Escopo do requisito 1.3</Escopo>
    <Tipo>Caso de Uso</Tipo>
  </Requisito>
</Requisitos>
```

Figura 5.12 - XML dos requisitos do projeto de teste.

Em um projeto de testes são definidos diversos planos de testes, que serão executados em diferentes versões do software, que aqui são denominadas como *Baselines*.

As informações exportadas do plano de teste consistem em três campos: ID, Nome e Descrição. As informações são obtidas na tabela *testplans*. Já as

informações sobre as *Baselines* do plano são adquiridas na tabela *builds* e descritas no XML através dos campos ID, Titulo e Descrição.

Como apresentado na Figura 5.13, pode existir vários planos de testes em um projeto, sendo que cada plano possui uma lista de *Baselines*.

```
<PlanosTestes>
  <PlanoTeste>
    <ID>1357</ID>
    <Nome>Plano de teste 0001</Nome>
    <Descricao>Este é o plano de teste 0001</Descricao>
    <BaseLines>
      <BaseLine>
        <ID>8</ID>
        <Titulo>Baseline 1</Titulo>
        <Descricao>Descrição do Baseline 1</Descricao>
      </BaseLine>
      <BaseLine>
        <ID>9</ID>
        <Titulo>Baseline 2</Titulo>
        <Descricao>Descrição do Baseline 2</Descricao>
      </BaseLine>
    </BaseLines>
  </PlanoTeste>
</PlanosTestes>
```

Figura 5.13 - XML dos planos de testes.

A última parte das informações do projeto de teste se refere as suas suítes e seus respectivos casos de testes. Nesta seção são exportadas todas as informações das plataformas, palavras chaves, *baseline*, resultados de teste, código dos defeitos encontrados, requisitos relacionados, além da especificação dos passos, características e especificações dos casos de testes, conforme exibido na Figura 5.14.

A suíte de teste é descrita através dos campos ID, Descrição e Nome, que são obtidas através das tabelas *testsuites* e *nodes_hierarchy*. Cada suíte pode ser

composta por vários casos de testes, por isso o XML gerado possui uma lista de casos descritas dentro da *tag <CasosTestes>.*

O caso de teste é o conjunto que mais possui informações do XML, sendo que algumas se referem a campos fixos da TestLink, outros a personalizados, além dos resultados de cada execução dos testes, dos requisitos e a referência para cada defeito encontrado.

```
<SuitesTestes>
  <SuiteTeste>
  <SuiteTeste>
    <ID>1</ID>
    <Descricao>Normal</Descricao>
    <Nome>Caminho Normal 1</Nome>
    <CasosTestes>
      <CasoTeste>
        <ID>1</ID>
        <Nome>CT1 - Exceção 3</Nome>
        <PreCondicao>Deve estar no estado ligado</PreCondicao>
        <Procedimento>Deve gerar a exceção 3</Procedimento>
        <FaseTeste>
        <ExecucaoTeste>
        <TipoTeste>
        <TecnicasTesteCB>
        <TecnicasTesteCP>
        <ResultadosTestes>
        <PalavrasChave>
        <Requisitos>
        <TempoEstimado>90</TempoEstimado>
      </CasoTeste>
    </CasosTestes>
  </SuiteTeste>
</SuitesTestes>
```

Figura 5.14 - XML da suíte e seus respectivos casos de testes.

A estrutura dos campos ID, Nome, PreCondicao e Procedimento são simples, pois se tratam dos campos fixo da TestLink. Essas informações são adquiridas nas tabelas *tcversions* e *nodes_hierarchy*.

Os campos FaseTeste, ExecucaoTeste, TipoTeste, TecnicasTesteCB, TecnicasTesteCP e TempoEstimado são personalizados e adquiridos na tabela

cfield_design_values. Os campos personalizados de seleção única apresentam um atributo Descricao, com o valor definido ao caso de teste. Já o caso dos campos de seleção múltipla, como as técnicas de testes, é definido uma lista de valores, onde a informação armazenada está no atributo Nome, conforme apresentado na Figura 5.15.

O campo personalizado TempoEstimado, que é do tipo Número, é descrito da mesma forma que os atributos de tabelas fixa da TestLink.

```
<TecnicasTesteCP>
  <TecnicaTesteCP>
    <Nome>Grafo Causa-Efeito</Nome>
  </TecnicaTesteCP>
</TecnicasTesteCP>
```

Figura 5.15 - Exemplo de um campo de seleção múltipla.

Os requisitos que são verificados através do caso de testes são descritos pelo identificador definido no começo do XML. Como podem existir mais de um requisito associado ao caso de testes, a campo Requisitos é uma lista de valores, que é preenchida através do relacionamento existente na tabela *req_coverage*.

Os resultados de testes, exibido na Figura 5.16, é uma lista que registra todas as execuções de testes. Cada resultado contém um ID, a plataforma utilizada, a baseline do software, o status e a data de execução, o testador que realizou o teste, o tempo de duração e uma lista com os defeitos encontrados.

A plataforma e a *baseline* são identificados apenas pelo identificador, pois o suas informações detalhadas já foram descritas no inicio do XML. Os defeitos também são identificados apenas pelo código, mas nesse caso, o detalhamento do defeito está na ferramenta Mantis, que será descrito em outro arquivo XML.

As informações referentes à execução dos testes são adquiridas na tabela *executions* e os defeitos relacionados a essas execuções são consultadas na tabela *execution_bug*.

```
<ResultadosTestes>
  <ResultadoTeste>
    <ID>20</ID>
    <Plataforma>4</Plataforma>
    <BaseLine>10</BaseLine>
    <Nota>Funcionou corretamente</Nota>
    <Data>2011-11-29 23:23:19.0</Data>
    <Status>p</Status>
    <Testador>Marcos</Testador>
    <Defeitos>
      <Defeito>
        <BugID>14</BugID>
      </Defeito>
      <Defeito>
        <BugID>21</BugID>
      </Defeito>
    </Defeitos>
    <TempoGasto>30</TempoGasto>
  </ResultadoTeste>
</ResultadosTestes>
```

Figura 5.16 - XML com informações dos resultados de testes.

O segundo arquivo XML gerado a partir das informações contidas no banco de dados da TestLink se refere aos usuários cadastrados na ferramenta. Este arquivo possui a mesma estrutura de dados do arquivo de usuários da Mantis. Para relacionar o mesmo usuário do Mantis e da TestLink na Base de conhecimento, é sugerido que o atributo Login seja o mesmo em ambas as ferramentas.

As informações exportadas dos usuários, conforme apresentado na Figura 5.17 são: ID, Login, Nome, Sobrenome e Email. A tabela responsável por armazenar essas informações na TestLink é a *users*.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Usuarios>
  <Usuario>
    <ID>1</ID>
    <Login>Marcos</Login>
    <Nome>Marcos</Nome>
    <Sobrenome>Reis</Sobrenome>
    <Email>marcosfsreis@gmail.com</Email>
  </Usuario>
</Usuarios>
```

Figura 5.17 - XML dos usuários da TestLink.

A segunda parte da exportação se refere aos dados armazenados na ferramenta Mantis. O banco dessa ferramenta também utiliza o SGBD MySQL. A exportação é feita em dois arquivos, um contendo as informações dos defeitos e outro com os usuários do sistema. A Figura 5.18 apresenta a estrutura do arquivo XML criada a partir das informações da Mantis.

O arquivo gerado possui dois níveis dentro de um projeto de teste. A primeira define as informações gerais do projeto, como ID, Descricao e Nome e a segunda é uma lista com todos os defeitos reportados por aquele projeto.

Os campos referentes ao projeto de testes são fixos da Mantis e adquiridos na tabela *mantis_project_table*. Já as informações sobre o defeitos são armazenadas em diversas tabelas, sendo que alguns campos possuem algumas regras especificas.

As informações referentes ao ID e ao Resumo são adquiridas na tabela *mantis_bug_table*, os campos Descricao, Passos e Informacao são consumidos através da tabela *mantis_bug_text_table*.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Projetos>
<ProjetoTeste>
  <ID>5</ID>
  <Descricao>Descrição do Projeto Teste</Descricao>
  <Nome>Projeto Teste</Nome>
  <Defeitos>
    <Defeito>
      <ID>20</ID>
      <Resumo>Problema na exceção 3</Resumo>
      <Descricao>Ao passar do estado y para o x,
      o sistema apresentou uma exceção</Descricao>
      <Passos></Passos>
      <Informacao></Informacao>
      <Estado>Fechado</Estado>
      <Frequencia>Sempre</Frequencia>
      <Gravidade>Pequeno</Gravidade>
      <Resolucao>Corrigido</Resolucao>
      <Causas>
        <Causa>
          <Descricao>Lógico</Descricao>
        </Causa>
      </Causas>
      <Classificacao>Desempenho</Classificacao>
      <Artefato>Exceção 1</Artefato>
      <Relator>Marcos</Relator>
      <Desenvolvedor>Ana</Desenvolvedor>
    </Defeito>
  </Defeitos>
</ProjetoTeste>
</Projetos>
```

Figura 5.18 - XML de defeitos gerados a partir da Mantis.

Os campos, Estado, Frequência, Gravidade e Resolução, precisam ser tratados. Como a Mantis permite que o código da ferramenta seja alterado, ela define os valores desses campos com algumas faixas, o que permite ao seu usuário criar novos valores a estes campos e através da faixa, definir a ordem de relevância de cada um deles.

No ambiente proposto foram considerados somente os valores padrões da ferramenta. Para tanto, a Tabela 5.4 irá exibir a relação do código interno com a descrição exibida para o campo Estado, a Tabela 5.5 a relação existente para

o campo Frequencia, a Gravidade é descrita através da Tabela 5.6 e a Resolução na Tabela 5.7.

Os códigos dos campos citados no parágrafo anterior são armazenados na tabela *mantis_bug_table,* sendo que cada defeito sempre possui apenas um valor para cada um dos quatro campos.

Tabela 5.4 - Valores do campo Estado do defeito.

Código	**Estado**
10	Novo
20	Retorno
30	Admitido
40	Confirmado
50	Atribuído
80	Resolvido
90	Fechado

Tabela 5.5 - Valores do campo Frequência do defeito.

Código	**Frequência**
10	Sempre
30	Ás vezes
50	Aleatório
70	Não se tentou
90	Incapaz de reproduzir
100	N/A

Tabela 5.6 - Valores do campo Gravidade do defeito.

Código	Gravidade
10	Recurso
20	Trivial
30	Texto
40	Mínimo
50	Pequeno
60	Grande
70	Travamento
80	Obstáculo

Tabela 5.7 - Valores do campo Resolução do defeito.

Código	Resolução
10	Aberto
20	Corrigido
30	Reaberto
40	Incapaz de reproduzir
50	Não corrigível
60	Duplicado
70	Não é um caso
80	Suspenso
90	Não será corrigido

O campo Causa é personalizado e permite a definição de mais de uma causa por defeito. Por isso ela é descrita no XML através de uma lista, sendo a *tag* <*Causas*> de nível pai e cada valor é exibido dentro da *tag* filha <*Causa*>*.* Para preencher essa informação, o módulo faz a consulta na tabela *mantis_custom_field_string_table* utilizando o nome “Causa do defeito”.

O campo Classificacao é adquirido na tabela *mantis_category_table*. Este é um campo fixo da Mantis, que permite através das configurações da ferramenta, criar vários valores para o defeito, sendo que sempre um defeito possui apenas uma classificação.

O artefato é um campo personalizado do tipo *String*, onde o usuário pode escrever qualquer texto. Suas informações são obtidas na tabela *mantis_custom_field_string_table* utilizando o nome de referência “Artefato”.

Os campos Relator e Desenvolvedor se referem a usuários do sistema. O primeiro define quem registrou o incidente, e é consultado na tabela *mantis_bug_table* através do atributo *reporter_id.*

O desenvolvedor é adquirido utilizando uma regra mais robusta. Ele só é preenchido quando o defeito se encontra nos estados Resolvido e Fechado, sendo que o campo é preenchido com o último usuário responsável pela mudança do defeito para o estado Resolvido. Para isso, o módulo consulta a tabela *mantis_bug_history_table.*

A segunda parte da exportação dos dados da Mantis, referente aos usuários do sistema, possui a mesma estrutura do XML da TestLink, conforme exibido na Figura 5.17.

5.4.3. Módulo Inclusão de Dados (MID)

O Módulo de Inclusão de Dados (MID) tem a finalidade de inserir ou atualizar as informações sobre os projetos de testes na base de conhecimento.

Com o propósito de desenvolver um ambiente com maior interoperabilidade, este módulo faz a atualização das informações através de arquivos XML. Isso

permite que qualquer fonte de dados seja utilizada, desde que os dados sejam organizados na estrutura definida dos arquivos.

O MAD apresentado na seção anterior, disponibiliza as informações armazenadas na TestLink e na Mantis em 4 arquivos XML, conforme padrão estipulado por este módulo.

Uma regra essencial no processamento das informações é a garantia da integridade dos dados. Isso significa que a cada tabela processado, o módulo irá verificar se essa informação já está cadastrada na base de conhecimento, através de campo “chaves”, e caso não exista a informação é inserida, caso contrário ela é apenas atualizada.

O processamento das informações pelo módulo pode ser dividido em três partes: Dos usuários (colaboradores), dos defeitos e do caso de testes.

A primeira recebe os arquivos de usuários da TestLink e da Mantis (que possuem a mesma estrutura) e faz o processamento das informações, utilizando a *tag* <*login*> como atribulo chave. Os usuários são armazenados na tabela Funcionario da base de conhecimento.

A segunda etapa faz o processamento das informações referentes aos defeitos. Por conta disso, dez tabelas são populadas: Artefato, Gravidade, Frequencia, Estado, Resolucao, ClassificacaoDefeito, CausaDefeito, Defeito_CausaDefeito, Defeito e ProjetoTeste.

Para a validação das sete primeiras tabelas citadas, o MID faz a comparação do valor enviado no XML como o campo Descricao da tabela. Na inserção dos dados, o módulo também gera um valor único inteiro para o atributo ID. Em relação à tabela Defeito, a validação é feita através da *tag* <*ID*> e do atributo ID da tabela da base de conhecimento.

Como um defeito pode estar associado a várias causas, o XML pode apresentar na *tag* <*Causas*> uma lista de causas associadas. Neste caso, para cada item dessa listagem será criado um registro na tabela Defeito_CausaDefeito utilizando os identificadores das tabelas Defeito e CausaDefeito.

Os defeitos registrados são organizados em um projeto de teste, informação essa que pode ser adquirida no XML dos defeitos. Na parte inicial da estrutura do arquivo, são apresentadas informações sobre o nome e a descrição do projeto, que são inseridos na tabela ProjetoTeste. Para validar a já existência desse projeto na base de conhecimento, é utilizado o nome do projeto como comparação.

O terceiro processamento prepara as informações referentes aos casos de testes, seus resultados, requisitos associados, palavras chaves, entre outras informações. Por conta da grande quantidade de informações, este arquivo XML irá preencher até 20 tabelas da base de conhecimento.

A tabela principal é a CasoTeste que utiliza a *tag* <*ID*> para validar se a informação já existe na base de conhecimento. As tabelas FerramentaTeste, TipoTeste, ExecucaoTeste e FaseTeste são inseridos através dos valores existentes no caso de teste. Através da descrição de seus valores, o módulo verifica se ele já está cadastrado e associa o ID definido na base de conhecimento aos seus respectivos atributos da tabela CasoTeste.

A tabela TecnicaTeste recebe informações de suas *tags* do tipo lista: TecnicasTesteCB e TecnicasTesteCP, que significam técnicas de testes de caixa brancas e técnicas de teste de caixa preta, respectivamente. O MID processa os valores desses dois campos e inseri em uma única tabela, facilitando a manipulação dos dados posteriormente.

Por permitir que um caso de teste seja definido por mais de uma técnica de teste, é necessário criar um relacionamento entre elas através de outra tabela, que neste caso é CasoTeste_TecnicaTeste.

Os requisitos também são processados e identificados através do ID. Os dados são armazenados nas tabelas TipoItemRequisito, Requisito e Requisito_CasoTeste.

As informações sobre Plataformas de testes, Palavras chaves, Planos e Suítes de testes são organizadas no XML em uma lista no começo do arquivo. Por isso, cada *node* da lista equivale a um registro na tabela. Em todos esses casos, o campo que é validado no momento da inserção é o nome.

O arquivo de casos de testes também preenche a tabela ProjetoTeste, assim como o de defeitos. Para que exista uma melhor associação entre as gerencias de testes e de defeitos, é altamente recomendado que o nome de ambos os projetos sejam o mesmo, pois nesse caso, o módulo irá identificar de forma única o projeto.

A MID foi desenvolvida em Java, utilizando a tecnologia Hibernate para manipulação dos dados no banco de dados MySQL. A IDE utilizada no desenvolvimento foi a NetBeans versão 6.8, utilizando a biblioteca Mysql Connector Java-5.1.18 para acesso ao banco de dados.

5.4.4. Módulo Consulta de Dados (MCD)

O Módulo de Consulta de Dados (MCD) consta de um aplicativo protótipo para ilustrar como as informações podem ser consumidas da Base de conhecimento. Utilizando comandos SQL, este aplicativo gera gráficos do tipo Pizza e Barras verticais.

O MCD foi desenvolvido em linguagem Java e utiliza a biblioteca JFreeChart, versão 1.0.13, para a apresentação dos gráficos. A IDE utilizada no desenvolvimento da aplicação foi a NetBeans versão 6.8. O módulo também considera que a base de conhecimento foi criada no SGBD MySQL com o nome de VVTeste. No Apêndice E são apresentadas as interfaces deste módulo.

As consultas do aplicativo podem ser feitas de duas formas: pré-definidas ou personalizadas. Nas consultas pré-definidas o protótipo disponibiliza 26 analises, conforme Tabela 5.8 sem que seja necessário o conhecimento da base de conhecimento e nem de comandos SQL. Já nas consultas personalizadas, o protótipo permite que o usuário defina sua própria consulta SQL e o resultado desse comando é disposto automaticamente nos gráficos.

A forma de apresentação dos gráficos varia de acordo com a análise feita. Um gráfico que possui apenas duas análises, como por exemplo, “Relator x Defeito” (que irá exibir a quantidade de defeitos encontrados por relator) irá organizar as informações em um gráfico de pizza. Esta forma de apresentação será a utilizada sempre que a análise dos dados for feitas sobre 2 variáveis, independente se foi uma consultada pré definida ou uma personalizadas.

Quando a analise for feita sobre três parâmetros, como por exemplo, “Defeito x Relator x Estado” (Quantidade de defeitos agrupados por estado e por relator), a forma de apresentação será a de barras verticais.

A cada geração de gráfico, o protótipo cria uma nova tela, o que permite que várias analises sejam feitas e comparadas simultaneamente pelo usuário da MCD.

Tabela 5.8 - Consultas pré-definidas do MCD.

Análise	Comentário
Defeito x Relator	Quantidade de defeitos agrupados por relator
Defeito x Estado	Quantidade de defeitos agrupados pelo seu estado atual.
Defeito x Estado x Relator	Quantidade de defeitos agrupados pelo seu estado atual e pelo relator.
Defeito x Técnica de teste	Quantidade de defeitos por técnica utilizada no caso de teste
Defeito x Causa de defeito	Quantidade de defeitos por sua causa
Defeito x Técnica de teste x Causa de defeito	Quantidade de defeitos por técnica de teste e pela sua causa
Defeito x Execução de teste	Quantidade de defeitos por forma de execução do caso de teste
Defeito x Fase de teste	Quantidade de defeitos por fase de testes
Defeito x Ferramenta de teste	Quantidade de defeitos por ferramenta utilizada na execução dos casos de testes
Defeito x Ferramenta de testes x Causa de defeito	Quantidade de defeitos por ferramenta utilizada na execução e pelas causas dos defeitos
Defeito x Tipo de teste	Quantidade de defeitos por tipo de teste
Defeito x Gravidade	Quantidade de defeitos por gravidade do defeito
Defeito x Classificação do defeito	Quantidade de defeitos por classificação do defeito
Defeito x Resolução	Quantidade de defeitos por resolução
Defeito x Resolução x Causa de defeito	Quantidade de defeitos por resolução e pela causa do defeito
Defeito x Tipo de requisito	Quantidade de defeitos por tipo de requisito
Defeito x Plataforma	Quantidade de defeitos por plataforma de testes
Defeito x Baseline	Quantidade de defeitos por baseline de testes
Defeito x Palavra chave	Quantidade de defeitos por palavra chave do caso de teste
Caso de teste x Palavra chave	Quantidade de casos de testes por palavra chave

Continua

Tabela 5.8 - Conclusão

Caso de teste x Técnica de teste	Quantidade de casos de testes por técnica de teste
Caso de teste x Fase de teste	Quantidade de testes por fase de teste
Caso de teste x Tipo de teste	Quantidade de testes por tipo de teste
Caso de teste x Tipo de teste x Técnica de teste	Quantidade de casos de testes por tipo e por técnica de teste
Caso de teste x Defeito x Tipo de requisito	Quantidade de casos de testes por defeito e por tipo de requisito
Caso de teste x Fase de teste x Tipo de requisito	Quantidade de casos de testes por fase de teste e por tipo de requisito

5.5. Base de conhecimento

A base de conhecimento de testes de software proposta no VVTeste visa armazenar as principais informações geradas durante um projeto de teste de software, unificando em um único repositório de dados informações de várias ferramentas, permitindo que esses dados sejam cruzados e novas informações sejam geradas.

As entidades e seus respectivos atributos foram definidos a partir de uma análise das informações importantes para uma base de conhecimento de testes de software. A definição desses dados auxiliou inclusive na detecção de campos personalizados que foram criados nas ferramentas TestLink e Mantis, como já foi apresentado nas seções 5.2.1 e 5.2.2.

Basicamente, a base pode ser dividida em: (i) Requisitos, (ii) Casos de testes e (iii) Defeitos. Cada parte relaciona um conjunto de entidades que ajudam a caracterizá-las. Além disso, elas também estão associadas entre si, o que permite que os dados sejam cruzados entre si.

Para facilitar a manipulação do ambiente VVTeste, que já conta com duas ferramentas que utilizam o SGBD MySQL, a base de conhecimento também foi desenvolvida para este banco de dados.

O Modelo de Entidade e Relacionamento (MER), vide Figura 5.19, ilustra o modelo conceitual do projeto de banco de dados.

A parte que trata dos Requisitos pode também ser utilizada como item de teste, dependendo do escopo do projeto de testes do SUT. Para tanto, a base de conhecimento define duas entidades: TipoItemTeste e Requisito.

No VVTeste, a entidade *TipoItemTeste* possui o campo *TipoRequisito* criado na TestLink. O *TipoItemTeste* classifica um determinado *Requisito*, que se refere aos requisitos cadastrados na TestLink. O gerenciamento de requisitos é opcional durante o gerenciamento de testes, por isso, na base de conhecimento esta área pode não estar definida para alguns projetos de testes.

Na parte de Gerenciamento dos testes, a principal entidade é a *CasoTeste*. Essa entidade é a responsável pelo relacionamento com os requisitos de testes.

Um *CasoTeste* define uma forma de *ExecucaoTeste*, que pode ser manual ou automático, que nesse caso conta com o apoio de uma *FerramentaTeste*. Além disso, um caso de teste também é executado em uma determinada *FaseTeste* e pode ser de um *TipoTeste,* que por sua vez pode utilizar várias técnicas de teste (*TecnicaTeste)*. Essas informações são geradas durante o gerenciamento dos testes na TestLink, conforme apresentado na seção 5.2. A *TecnicaTeste* agrupa os valores dos campos *TecnicaTesteCB* e *TecnicaTesteCP* em uma única entidade.

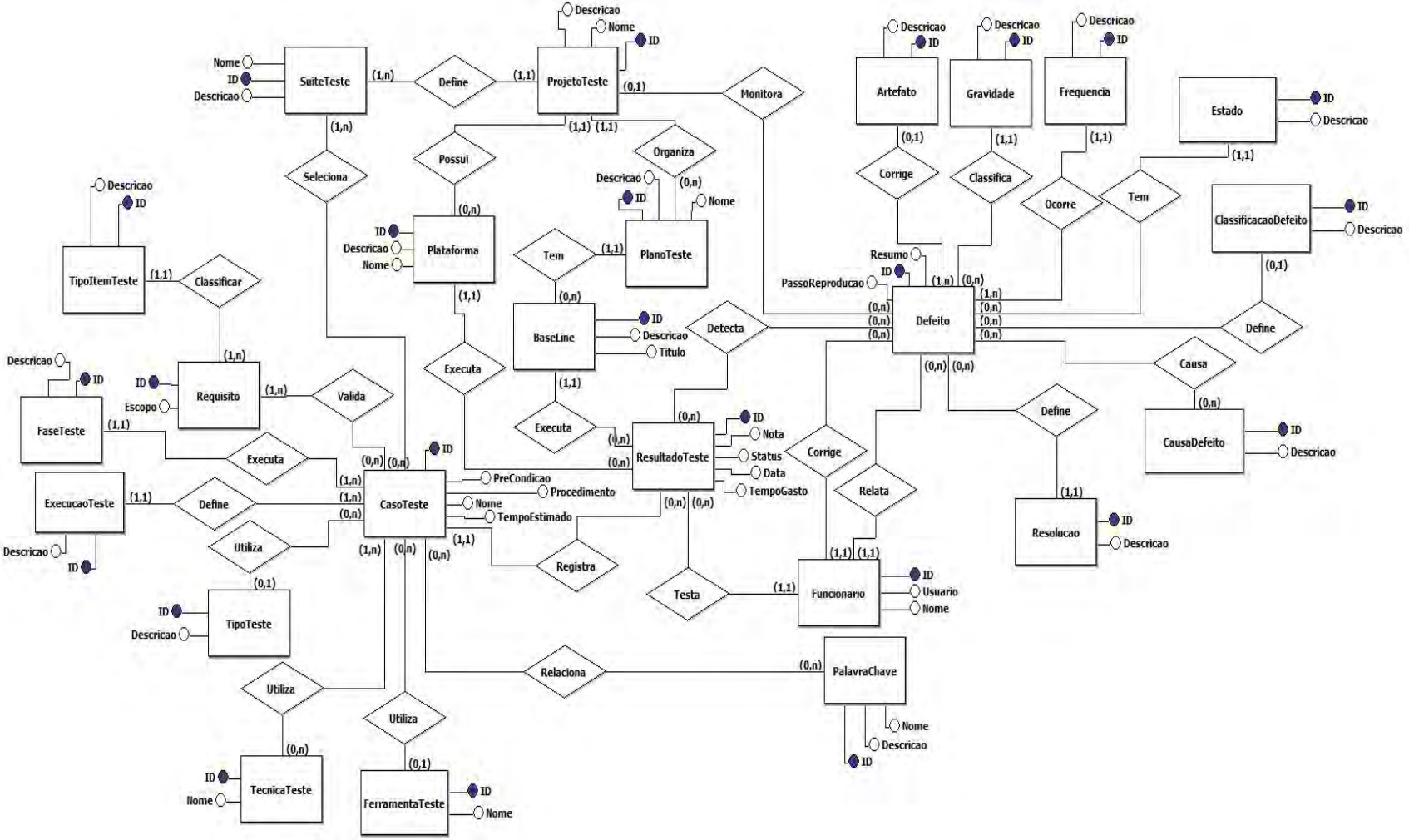

Figura 5.19 - Diagrama de Entidade e Relacionamento da Base de conhecimento.

Os casos de testes também podem estar associados a várias palavras chaves, que serão armazenadas na entidade *PalavraChave*.

Um caso de teste poderá estar associado em várias suítes de testes, através da entidade *SuiteTeste*, que for sua vez faz parte de um projeto de teste (*ProjetoTeste)* com suas *Plataforma, PlanoTeste* e *BaseLine.*

A plataforma consiste no ambiente em que o SUT está sendo exposto no momento do teste. Já o *baseline*, que também pode ser denominado como *release* ou *build*, consiste na versão do SUT utilizado.

Um plano de teste consiste em um conjunto de casos de testes que são executados em determinadas plataforma com uma versão especifica do software em teste. Essa execução dos testes irá gerar os resultados de testes, que são armazenadas na entidade *ResultadoTeste*, com o status do resultado, tempo gasto no teste, a data e os comentários sobre a execução.

A entidade de *ResultadoTeste* também é a responsável por realizar o relacionamento com a terceira área da base de conhecimento, responsável pelas informações do gerenciamento de defeitos. Um *ResultadoTeste* pode apresentar vários defeitos e que devem estar associados ao resultado de teste responsável pela sua detecção.

A entidade *Defeito* é a responsável por armazenar todas as informações dos defeitos registrados no projeto de testes. Grande parte dos dados é classificada através de outras entidades, que representam os campos criados na ferramenta Mantis (seção 5.2).

Um defeito é classificado de acordo com sua *Gravidade*, ocorre com uma determina *Frequencia* e se encontra em um determinado *Estado*. Além disso, um defeito também é definido em uma determinada *ClassificacaoDefeito*.

Durante o ciclo de vida do defeito, é detectada a *CausaDefeito* e uma *Resolucao* é adotada a ela. Durante a resolução do defeito, o mesmo pode ser relacionado a um *Artefato*, que pode ser uma classe ou função do software, um módulo ou até mesmo uma funcionalidade do programa.

Os *Funcionario* são associados tanto na área do gerenciamento de testes, quanto na de gerenciamento de defeitos. Essa entidade irá definir o responsável pela execução dos testes, o relator do defeito e também o desenvolvedor que realizou a correção no software. O relacionamento feito é através do *username* do usuário, por isso, é aconselhado que o *username* do funcionário seja o mesmo na TestLink e no Mantis.

Durante a elaboração do modelo conceitual da base de conhecimento, foi buscado que os nomes das entidades representassem o mais próximo possível o significado das informações que elas armazenam. Isso auxiliará durante a preparação de comando SQL no Módulo de Consulta de Dados.

Em relação ao projeto de banco de dados, foram aplicados todos os sete passos definidos por Elmasri & Navathe (ELMASRI, et al., 1994) no mapeamento do modelo conceitual em modelo lógico e posteriormente validado com as três formas normais da técnica de normalização.

Com isso, foram criadas várias tabelas de relacionamentos, principalmente nos relacionamentos do tipo muitos para muitos. O modelo físico gerado está disponível no Apêndice D.

Com a definição do modelo lógico, foi possível a criação do script capaz de criar o modelo físico do banco de dados no SGBD MySQL. Este script também está disponível no Apêndice D.

6 AVALIAÇÃO DO AMBIENTE VVTESTE EM RELAÇÃO AOS MODELOS DE REFERÊNCIA

Os processos de Verificação e Validação de software são parte de todo um processo de desenvolvimento de um software. Os modelos CMMI e MPS.br definem vários aspectos e áreas que precisam ser abordados durante a construção e a vida de um software.

A seguir é apresentado como o ambiente proposto VVTeste auxilia os processos de V&V descritos pelos modelos de referência, descrevendo qual é o papel do ambiente no atendimento do resultado esperado. Para apresentar a avaliação feita, serão descritos o que o modelo solicita que seja feito e como a VVTeste auxilia no atendimento as diretrizes esperadas.

6.1. O modelo CMMI

O CMMI (CMMI, 2010) descreve que o propósito da Verificação (VER) é assegurar que os produtos de trabalho selecionados atendem aos seus requisitos especificados e para isso define 3 metas: Preparar para a verificação, realizar revisões por pares e verificar os produtos de trabalhos selecionados.

A Tabela 6.1 apresenta como são atendidas as três práticas previstas pela meta SG1 - Preparar para a verificação, que tem como principal objetivo preparar uma estrutura para a realização da verificação e definir os produtos de trabalho e os métodos que serão utilizados durante o processo.

As informações descritas no campo “*O que?”* da Tabela 6.1 foram retiradas do guia de desenvolvedores do CMMI, versão 1.3 (CMMI, 2010).

Tabela 6.1 - Avaliação da meta SG1 - Preparar a verificação.

SP 1.1 – Selecionar os Produtos de Trabalho para Verificação	
O que?	Selecionar os produtos de trabalho a serem verificados e os métodos de verificação que serão usados para cada um. Os produtos de trabalho são selecionados com base em suas contribuições ao atendimento dos objetivos e requisitos do projeto e ao tratamento dos seus riscos. **Produtos de Trabalho Típicos** 1. Listas dos produtos de trabalho selecionados para verificação 2. Métodos de verificação para cada produto de trabalho selecionado
Como?	Através do VVTeste, via a TestLink, é possível cadastrar os requisitos que serão avaliados na Verificação. Além disso, caso o método de verificação seja o teste e os produtos de trabalho possam ser modelados em máquinas de estados finitas, o usuário pode usar a Condado para a geração dos casos de testes. Para auxiliar na tomada de decisão dos métodos adotados nos testes, a equipe de testes pode consumir as informações da base de conhecimento, o que aumenta a eficiência dos testes através das lições aprendidas em projetos anteriores.
SP 1.2 – Estabelecer o Ambiente de Verificação	
O que?	Estabelecer e manter o ambiente necessário para dar apoio à verificação. Um ambiente deve ser estabelecido para que a verificação ocorra. O ambiente de verificação deve ser adquirido, desenvolvido, reutilizado, modificado ou qualquer combinação dessas ações, dependendo das necessidades do projeto. **Produtos de Trabalho Típicos** 1. Verificação de ambiente
Como?	O VVTeste tem a finalidade de atender principalmente a está prática. No que diz respeito a testes, o ambiente é formado pela Condado, para geração automática de casos de testes, que está integrada a TestLink para gerenciamento dos testes e a Mantis para registro dos defeitos detectados. O ambiente pode ser reutilizado em diversos projetos, sendo que as informações obtidas em projetos anteriores podem ser consultadas tanto nas ferramentas como na base de conhecimento. A VVTeste tem o foco principal no apoio aos testes, porém a

Continua

Tabela 6.1 - Conclusão

	ferramenta TestLink pode ser utilizada para armazenar informações de outros métodos adotados na Verificação, atendendo as expectativas desta prática.
SP 1.3 – Estabelecer Procedimentos e Critérios de Verificação	
O que?	Estabelecer e manter procedimentos e critérios de verificação para os produtos de trabalho selecionados. **Produtos de Trabalho Típicos** 1. Procedimentos de verificação 2. Critérios de verificação
Como?	O procedimento pode ser definido através de caso de testes. A ferramenta Condado fornece parte dos casos de testes que podem ser executados no software. Estes procedimentos são armazenados na TestLink, que permite que outros caso de testes sejam adicionados, aumentando a cobertura dos testes.

A segunda meta desse processo é a SG2 - Realizar Revisão por Pares, cuja finalidade é que duas pessoas juntas procurem defeitos nos produtos de trabalho e também detectem modificações necessárias. Para tanto, essa meta define três práticas a serem realizadas. Na Tabela 6.2 são apresentados os objetivos dessas práticas e como o VVTeste apoia essa atividade.

Tabela 6.2 - Avaliação da meta SG2 - Realizar revisões em pares.

SP 2.1 – Preparar para Revisão por Pares	
O que?	As atividades de preparação para a revisão por pares incluem tipicamente a identificação da equipe que será convidada para participar na revisão de cada produto de trabalho; a identificação dos revisores-chave que devem participar da revisão por pares; a preparação e atualização de todos os documentos que serão usados durante a revisão por pares, tais como listas de verificação, critérios de revisão e cronograma das revisões. **Produtos de Trabalho Típicos** 1. Cronograma das revisões por pares 2. Listas de verificação das revisões por pares 3. Critérios de entrada e saída para os produtos de trabalho 4. Critérios para solicitação de outra revisão por pares 5. Material de treinamento de revisão por pares 6. Produtos

Continua

Tabela 6.2 - Conclusão

Como?	O VVTeste, via a TestLink, pode ser utilizada para gerenciar a execução das listas de verificação, além de armazenar os critérios de entrada e saídas para cada produto verificado. A utilização do VVTeste permite que os defeitos encontrados, devidamente armazenados na Mantis, sejam associados aos itens de verificação. O VVTeste, via a TestLink, também possui recursos (em Relatórios e Telas) que permite aos gerentes acompanhar o andamento das atividades de verificação.
SP 2.2 – Realizar Revisão por Pares	
O que?	Conduzir a revisão por pares nos produtos de trabalho selecionados e identificar os problemas resultantes da revisão por pares. **Produtos de Trabalho Típicos** 1. Resultados de revisão por pares 2. Problemas encontrados na revisão por pares 3. Dados de revisão por pares
Como?	Problemas encontrados na revisão por pares podem ser registrados no VVTeste, via Mantis, permitindo o acompanhando deste incidente até a sua resolução.
SP 2.3 – Analisar Dados de Revisão por Pares	
O que?	Analisar dados sobre a preparação, condução e resultados de revisão por pares. Produtos de Trabalho Típicos 1. Dados de revisão por pares 2. Itens de ação de revisão por pares
Como?	Os dados dos defeitos encontrados podem ser consultados no VVteste de maneira analítica, através de cada registro ou de forma sintética, com os gráficos e estatísticas geradas pela ferramenta. Além disso, a base de conhecimento armazena as informações de projetos anteriores, o que pode auxiliar na definição dos itens de ação. O projeto atual também pode ser consultado na base de conhecimento através do MCD, onde é possível visualizar outros tipos de gráficos.

Após a verificação através de um método específico, como é o caso da revisão por pares, o CMMI define que a terceira e última meta que é a SG3 - Verificar os Produtos de Trabalhos Selecionados. Nesta atividade os métodos, procedimentos e critérios são utilizados para verificar o produto e qualquer

serviço associado. Para isso, são definidas duas práticas para essa meta, que são apoiadas pela VVTeste, conforme apresentado na Tabela 6.3.

Tabela 6.3 - Avaliação da meta SG3 - Verificar os Produtos de Trabalhos Selecionados.

SP 3.1 – Realizar Verificação	
O que?	Os produtos de trabalho são verificados em relação aos seus requisitos especificados. As verificações dos produtos e dos produtos de trabalho propiciam a detecção precoce de problemas e pode resultar na remoção antecipada dos mesmos. Os resultados da verificação economizam gastos consideráveis de isolamento de defeitos e retrabalho associado aos problemas de localização de defeitos. **Produtos de Trabalho Típicos** 1. Resultados de verificação 2. Relatórios de verificação 3. Demonstrações 4. Log de procedimentos “as run”
Como?	O VVTeste armazena os requisitos especificados, além dos procedimentos de testes, permitindo que os resultados da verificação sejam registrados. Já os defeitos encontrados são registrados na Mantis. O VVTeste permite a impressão de relatórios contendo as informações que estão registradas.
SP 3.2 – Analisar Resultados de Verificação e Identificar Ações Corretivas	
O que?	Analisar os resultados de todas as atividades de verificação e identificar ação corretiva. Os resultados reais devem ser comparados com o critério de verificação estabelecido para determinar a aceitação. Os resultados das análises são registrados como evidência de que a verificação ocorreu. Para cada produto de trabalho, todos os resultados disponíveis da verificação são analisados de forma incremental e ações corretivas são iniciadas para assegurar que os requisitos sejam atendidos. **Produtos de Trabalho Típicos** 1. Relatórios de análise (ex: estatísticas de desempenho, análise de causas de não conformidades, tendências e comparação do

Continua

Tabela 6.3 - Conclusão

	comportamento real do produto com modelos) 2. Relatórios de problemas 3. Solicitações de mudanças nos métodos de verificação, critérios e ambiente 4. Ações corretivas nos métodos, critérios, e/ou ambiente de verificação
Como?	Por se tratar de ferramentas Web, o acesso às informações e consequentemente os resultados da verificação são simplificados na VVTeste. Os relatórios de problemas podem ser gerados. A base de conhecimento pode ser utilizada para auxiliar na definição de ações corretivas, através de lições aprendidas de projetos anteriores, e também na avaliação dos resultados do projeto atual.

O ambiente VVTeste fornece um apoio bastante robusto a todo o processo de Verificação definido pelo CMMI, pode disponibiliza uma ferramenta para geração de casos de testes automática através de modelo de máquina de estados. Isso auxilia na definição do escopo dos produtos de trabalhos e nos métodos de verificação.

Além disso, a TestLink e a Mantis não se limitam a dar apoio apenas no método de testes da verificação. Todos os problemas encontrados podem ser registrados na Mantis e a TestLink auxilia em grande parte do gerenciamento da verificação.

A base de conhecimento e o módulo MCD auxiliam na avaliação dos resultados alcançados durante os testes e na tomada de decisão das ações corretivas.

O CMMI define três metas para a realização do processo de Validação (VAL), que são semelhantes ao de Verificação, utilizando inclusive métodos similares, como por exemplo, o teste, a inspeção e a simulação. O que muda entre esses processo é a visão sobre o produto, a validação tem o objetivo de demonstrar

que o produto ou parte do produto atende ao seu uso pretendido quando utilizado em seu ambiente alvo (CMMI, 2010).

O processo de Validação não requer a prática de revisão de pares, por isso possui apenas a SG1 – Preparar a validação e a SG2 - Validar o Produto ou os Componentes de Produto. Essas práticas solicitam o mesmo ambiente e as mesmas atividades do processo de Verificação, mudando apenas a característica da execução.

Por isso, o ambiente proposto auxilia o processo de Validação da mesma forma que o processo de Verificação. O ambiente integrado aos dois processos inclusive aumenta a eficiência dos registros, da preparação das equipes e ainda proporciona uma base de conhecimento mais robusta.

6.2. O modelo MPS.br

A Melhoria do Processo de Software Brasileira tem uma proposta bastante semelhante à abordada pelo CMMI aos processos de Verificação (VER) e Validação (VAL). Os resultados esperados pela MPS.br para estes dois processos são parecidos, apenas sendo adaptados ao foco da atividade.

Para realizar essa avaliação, a Verificação define seis resultados esperados, já a Validação é composta por sete, sendo que o *VAL7 - Evidências de que os produtos de software desenvolvidos estão prontos para o uso pretendido são fornecidas* é a única atividade que não possui um resultado equivalente no processo de Verificação.

Utilizando o fator de semelhança entre os dois processos, a Tabela 6.4 irá apresentar como o ambiente VVTeste dá apoio a estes processos da MPS.br apresentando de maneira unificada os resultados de 1 a 6 dos processos de

VER e VAL. As informações do "*O que?"* foi retirada do Guia de implementação – Parte 4 da MPS.br (SOFTEX, 2011).

Tabela 6.4 - Avaliação dos processos de VER e VAL da MPS.br.

VER1 - Produtos de trabalho a serem verificados são identificados e VAL1 - Produtos de trabalho a serem validados são identificados	
O que?	Espera-se que os produtos que serão avaliados sejam localizados e escolhidos. No caso da Verificação, essa informação pode ser adquirida do plano do projeto ou dos requisitos descritos. Na Validação, a importância para o cliente e a complexidade das funcionalidades é considerada durante essa identificação
Como?	Os requisitos podem ser consultados ou alterados na ferramenta TestLink. Outro fator importante, é que informações de projetos anteriores que podem ser adquiridas tanto na TestLink como na Base de conhecimento podem ajudar a avaliar a complexidade dos produtos de trabalho.
VER2 - Uma estratégia de verificação é desenvolvida e implementada, estabelecendo cronograma, revisores envolvidos, métodos para verificação e qualquer material a ser utilizado na verificação e VAL2 - Uma estratégia de validação é desenvolvida e implementada, estabelecendo cronograma, participantes envolvidos, métodos para validação e qualquer material a ser utilizado na validação	
O que?	Como o próprio nome já diz, este é o momento de realizar o planejamento. Tanto a Validação como na Verificação, a técnica de teste é citada como forma de realizar essas avaliações. Para que o teste possa ser executado, os casos de testes precisam ser criados, registrados e estarem disponíveis aos envolvidos. A divisão dos trabalhos entre os envolvidos também é uma atividade importante, pois interfere no cronograma.
Como?	Para a geração dos casos de testes o ambiente disponibiliza a ferramenta Condado para os produtos de trabalho que possam ser modelados através de MEFEs. Neste caso, eles são criados automaticamente e integrados à ferramenta de gerenciamento dos testes, a TestLink. O VVTeste, via a TestLink tem um papel muito importante nessa atividade, porque além de consumir os casos de testes gerados pela Condado, ele permite que outros casos sejam descritos. Além disso,

Continua

Tabela 6.4 - Continuação

	ele também permite a divisão do trabalho entre os envolvidos na avaliação. Todas as informações referentes aos métodos adotados e informações da estratégia adotada que esteja diretamente ligada à execução da avaliação podem ser descritas na Testlink, a qual pode ser facilmente consultada por todos os membros envolvidos.
VER3 - Critérios e procedimentos para verificação dos produtos de trabalho a serem verificados são identificados e um ambiente para verificação é estabelecido e **VAL3 - Critérios e procedimentos para validação dos produtos de trabalho a serem validados são identificados e um ambiente para validação é estabelecido**	
O que?	Para essas atividades espera-se que todas as informações e estruturas necessárias para a avaliação dos produtos de trabalhos sejam definidas, isso inclui os critérios e procedimentos de testes, ferramentas de apoio para planejamento, gerenciamento e execução dos testes e a infraestrutura e hardwares necessários.
Como?	Os critérios e procedimentos podem ser armazenados no VVTeste, via a TestLink, juntamente com cada um dos casos de testes gerados. Conforme descrito pelo modelo, os aspectos importantes para o ambiente estabelecido é que ele seja capaz de gerenciar o planejamento e a execução dos testes. Isso também pode ser feito através da TestLink. Este ambiente não contempla nenhuma forma automatizada de execução dos testes, pois normalmente este tipo de ferramenta depende muito da arquitetura utilizada no desenvolvimento do software, sendo assim, caso a organização deseje automatizar a execução, deverá avaliar uma ferramenta que atenda as características do produto.
VER4 - Atividades de verificação, incluindo testes e revisões por pares, são executadas e **VAL4 - Atividades de validação são executadas para garantir que o produto esteja pronto para uso no ambiente operacional pretendido**	
O que?	Espera-se que todo o planejamento realizado seja executado, utilizando diversas técnicas, que segundo o modelo, inclui obrigatoriamente os testes e as revisões por pares para a Verificação.
Como?	Durante a execução das atividades as informações são consultadas e dirigidas no VVTeste, através da TestLink. Nela os executores

Continua

Tabela 6.4 - Continuação

	registram quais os casos de testes foram executados com sucesso, quais falharam e os que foram impedidos de serem avaliados. O gerente pode acompanhar, através dos gráficos e das interfaces da TestLink, o andamento das atividades e comparar o andamento cronograma estipulado.
VER5 - Defeitos são identificados e registrados e VAL5 - Problemas são identificados e registrados	
O que?	Este resultado espera que todos os problemas encontrados sejam documentos e que sejam definidos quais serão tratados. Ressalta a importação do acompanhamento do problema até a sua conclusão, apesar de estabelecer que a organização tenha liberdade para não corrigir todos os problemas, de acordo com os seus critérios de avaliação.
Como?	O VVTeste, via a Mantis, permite registrar esses defeitos. A integração entre TestLink e Mantis, permite relacionar os casos de testes aos defeitos encontrados. A Mantis permite o acompanhamento do defeito/problema até a sua resolução, aumentando a segurança sobre os relatos registrados.
VER6 - Resultados de atividades de verificação são analisados e disponibilizados para as partes interessadas e VAL6 - Resultados de atividades de validação são analisados e disponibilizados para as partes interessadas	
O que?	Tem a finalidade de avaliar os resultados obtidos durante a verificação/validação. Os envolvidos devem responder perguntas do tipo: - Os critérios definidos foram satisfeitos? - As ações corretivas planejadas foram concluídas? - A verificação/validação foi executada conforme planejado? - Os resultados obtidos permitem a aprovação do artefato?
Como?	As ferramentas Mantis e TestLink disponibilizam diversos relatórios e gráficos que auxiliam na análise dos resultados. Além disso, através da Base de conhecimento e do módulo MCD, as resultados podem ser analisados de outras formas, inclusive comparando com projeto ou baterias de testes anteriores.
VAL7 - Evidências de que os produtos de software desenvolvidos estão prontos para o uso pretendido são fornecidas	
O que?	Se as atividades de validação apresentam evidências que os requisitos e as expectativas dos clientes foram atendidos, o passo final deve ser a apresentação desses relatórios as partes interessadas.

Continua

Tabela 6.4 - Conclusão

	Uma reunião com os clientes e/ou usuários finais pode ser feita para a apresentação dos resultados da validação, a correção dos problemas detectados, para que se obtenha o aceite de que o produto está pronto para o uso.
Como?	Uma das formas de evidenciar os produtos é apresentar um relatório dos problemas encontrados e solucionados. Este relatório pode ser adquirido através da Mantis, que permite a exportação de alguns dados para outros formatos, como por exemplo, o Word, e disponibiliza alguns dos gráficos produzidos.

Assim como no CMMI, o ambiente VVTeste também fornece um apoio eficiente as atividades de Verificação e Validação da MPS.br, principalmente quando a técnica utilizada nas atividades é a de testes. Além disso, a integração de todas as ferramentas organiza as informações geradas e consumidas pelos membros participantes, disponibilizando as informações de diversas formas, que podem ser trabalhadas e apresentadas às partes interessadas.

7 ESTUDO DE CASO

Este Capítulo apresenta uma prova de conceito do ambiente VVTeste. Com a finalidade de validar se o ambiente VVTeste alcança os resultados esperados, foi realizado um estudo de caso acadêmico, o qual cobriu desde a modelagem de um software até a análise dos dados na base de conhecimento.

O estudo de caso foi baseado no relatório apresentado em (MORAIS, ET al., 2009). Esse relatório descreve a validação de um software embarcado de uma máquina de café, para a qual a metodologia CoFi (*Conformance Test and Fault Injection*) (AMBROSIO, 2005) e a ferramenta Condado para a geração de casos de testes foram aplicadas. As máquinas de estados criadas, os casos de testes gerados, os requisitos especificados e os defeitos detectados são apresentados nesse relatório.

Utilizando parte dos dados proporcionados por este relatório, a prova de conceito do ambiente VVTeste mostrou atender as atividades de geração e gerenciamento de testes, e a utilização da base de conhecimento.

7.1. Informações gerais do estudo de caso usado

7.1.1. Sistema alvo de teste

A máquina de café, sistema alvo de teste, permite que sejam escolhidos três tipos de produtos, que para efeito desse projeto de testes são chamados de serviços: Café puro (S1), Cappuccino (S2) e Café com leite (S3).

Além do tipo de produto, a máquina também permite a variação em relação à forma de adoçar, permitindo as opções: Pouco açúcar e Muito açúcar.

A máquina também possui dispositivos de comandos e de monitoração. Para comandar os eventos do sistema, são disponibilizados seis botões e uma

entrada para fichas, e para indicar o estado ao usuário são disponibilizadas nove lâmpadas.

Além disso, existem dois dispositivos de atuação na máquina, um para controlar a liberação dos copos e outro para controlar a liberação de cada um dos tipos de produtos. Dois outros dispositivos avaliam o nível de cada produto e a presença de copos no estoque.

Para a exemplificação deste estudo de caso, será considerado apenas o Serviço 1 (S1), denominado como “Preparar copo de café puro”.

7.1.2. Requisitos

A especificação do sistema alvo define 14 requisitos a serem atendidos pelo sistema. Eles são identificados com o prefixo “R” e um número sequencial único, como por exemplo: R1 e R2.

Os requisitos definidos são:

- R1 - O sistema deverá permanecer desligado até que o botão de liga/desliga seja passado para o estado *On*.
- R2 - Sempre que o sistema for ligado, ele deverá verificar se há copo no estoque e se há café, leite e chocolate nos reservatórios. Em caso positivo para todos eles, o sistema poderá aceitar a inserção de fichas na máquina. Caso não haja copo ou qualquer um dos produtos na quantidade suficiente para a produção do pedido, a máquina não poderá aceitar a inserção de fichas até que seja reposto o que está em falta.
- R3 - Após a inserção da ficha, a máquina só deverá aceitar os seguintes comandos das seguintes escolhas nesta ordem: (i) Escolha do tipo de produto (café puro, café com leite ou cappuccino); (ii) Escolha da forma de adoçar (pouco ou muito açúcar).

- R4 - Caso o usuário forneça um comando não esperado (fora da ordem especificada no requisito R3), o sistema deverá permanecer no estado corrente, ou seja, não deverá responder a nenhum evento não esperado.
- R5 - O processamento de determinado pedido não poderá ser abortado. Uma vez que uma ficha é inserida na máquina, um ciclo de processamento de produto deverá ser concluído para se voltar ao estado inicial.
- R6 - Após a inserção de uma ficha, a máquina não poderá receber a inserção de uma nova ficha até a finalização do processamento do pedido em andamento.
- R7 - À medida que as escolhas são feitas pelo usuário, as lâmpadas correspondentes à opção desejada deverão ser acesas no momento da escolha. Ou seja, quando o usuário escolher o tipo de produto (café puro, café com leite ou cappuccino), a lâmpada que indica sua escolha deverá acender no momento em que ele aperta o botão e assim por diante.
- R8 - Quando o processamento de um pedido for finalizado, deverá ser acendida uma lâmpada para indicação de tal situação e que só deverá ser apagada quando o copo for retirado do suporte.
- R9 - Ao ser retirado o copo do suporte após a finalização de um pedido, todas as lâmpadas da máquina (exceto a lâmpada que indica que a máquina está ligada) deverão ser apagadas.
- R10 - Após todas as lâmpadas serem apagadas devido à retirada de um copo do suporte após a finalização do processamento de um pedido, a presença ou não dos recursos (produtos e copos) deverá ser verificada. Caso pelo menos um dos produtos esteja em falta, a máquina não poderá aceitar a inserção de uma nova ficha até que este produto em falta seja reposto.
- R11 - Caso a máquina seja desligada antes do início do processamento efetivo do produto, ao ser ligada novamente, ela deve ir para seu estado

inicial. Ou seja, o usuário perderá a ficha e o pedido que estava em andamento será descartado. Um novo pedido deverá ser feito com uma nova ficha.

- R12 - A máquina sempre demorará dez segundos para processar o pedido feito (este é o tempo entre o acionamento do botão da última escolha e o pedido estar dentro do copo pronto para ser consumido).
- R13 - Caso a máquina seja desligada durante os dez segundos de processamento do produto, ela deverá continuar funcionando até a finalização do processamento. Apenas após sua finalização é que a máquina será efetivamente desligada.
- R14 - Quando a máquina for desligada, a lâmpada que indica seu ligamento (*On*) deverá ser apagada. Se houver quaisquer outras lâmpadas acesas, todas elas também deverão ser apagadas.

7.1.3. Os casos de teste

De acordo com o conceito da CoFI, devem ser criados modelos parciais para definir o comportamento do sistema. Para tanto, esses comportamentos são divididos entre: Normal, Exceções especificadas, Eventos normais ocorridos em momentos inoportunos (caminhos furtivos) e Frente às falhas de hardware (tolerância a falhas).

A criação dos modelos formais apresentada no relatório de (MORAIS, et al., 2009), definiu sete modelos, sendo que:

- Um para o comportamento normal;
- Três para as exceções especificadas, denominadas de Ex5, Ex6 e Ex134;
- Dois para os caminhos furtivos, denominadas de CF1 e CF2;
- Um para tolerância as falhas.

Os setes modelos foram descritos através do Modelador de Máquinas de Estados, conhecida como MME. As imagens dos modelos estão disponíveis no Apêndice F. Em seguida, os modelos gerados foram processados na Condado para a geração automática dos casos de testes.

A Condado gerou um total de 76 casos de testes para os setes modelos descritos. Para cada modelo, a Condado gera um arquivo de extensão *.seq* com os casos de testes criados. Sendo assim, foram gerados setes arquivos.

7.2. Passos para uso do VVTeste

7.2.1. Preparação do projeto de teste

O projeto de teste consiste em criar um ambiente para que os dados relevantes dos testes, tais como requisitos e casos de testes, sejam registrados. A ferramenta para o gerenciamento dos testes é a TestLink.

O primeiro passo é a criação do projeto de teste. O projeto de teste descrito neste capítulo tem o nome de “Estudo de caso”, seus casos de testes recebem o prefixo “EC” e a funcionalidade de controle de requisitos é ativada.

O segundo passo é a criação de todos os campos personalizados, utilizando os alias definidos na seção 5.2.1, e em seguida a associação desses campos ao projeto de testes criados, usando-se do recurso de personalização de campos disponibilizado pela TestLink.

O último passo é a criação dos usuários que irão utilizar a ferramenta, definido o nível de acesso/papel no projeto, a sua identificação e senha de acesso ao TestLink. Foram criados três usuários: Ana, Mauricio e Thiago. Assim como os campos personalizados, é preciso definir que os usuários possuem acesso ao projeto de teste “Estudo de caso”.

7.2.2. Cadastro dos Requisitos

Para o cadastro dos requisitos foi criada uma especificação de requisitos de sistema com o título “Máquina de café”. Dentro dessa especificação foram cadastrados os quatorzes requisitos definidos para a máquina de café.

Os requisitos foram identificados como R1, R2, R3, R4, R5, R6, R7, R8, R9, R10, R11, R12, R13 e R14 e foram divididos entre três tipos de requisitos: Interface, Funcional e Não funcional.

7.3. Casos de testes Gerados

Neste estudo de caso são considerados apenas 30 dos 76 casos de teste gerados automaticamente pela ferramenta Condado.

O próximo passo consta da criação das suítes de testes na TestLink. Foi definida a regra que cada modelo seria considerado como uma suíte, por isso foi criada setes suítes com os seguintes nomes: Serviço 1 - Normal, Serviço 1 - Exceção 5, Serviço 1 - Exceção 6, Serviço 1 - Exceção 134, Serviço 1 - Caminho furtivo 1, Serviço 1 - Caminho furtivo 2 e Serviço 1 - Falha. Em cada suíte, foi anexada uma imagem com a modelagem da máquina de estados para auxiliar os testadores no momento da execução dos testes.

Para que os casos de testes sejam importados na TestLink, eles precisam estar em arquivos XML em um formato definido pela ferramenta. Para que isso seja possível, é preciso abrir o Módulo VVTeste – MICT para transformar os arquivos *.seq* da Condado em XMLs da TestLink.

Além de preparar o arquivo no formato XML, o MICT permite que a definição de palavras chaves para cada conjunto de casos de testes, além de dados gerais dos casos de testes (CTs). A Tabela 7.1 apresenta informações sobre o

processamento de cada arquivo de casos de testes pela MICT no estudo de caso.

Tabela 7.1 - Informações dos casos de testes importados.

Serviço	Qtde CTs	Prefixo	Palavras chaves
Normal	2	S1-N	Copo de café, Modo operação Normal e Serviço 1
Exceção 5	6	S1-Ex5	Copo de café, Modo operação Exceção 5 e Serviço 1
Exceção 6	2	S1-Ex6	Copo de café, Modo operação Exceção 6 e Serviço 1
Exceção 134	3	S1-Ex134	Copo de café, Modo operação Exceção 134 e Serviço 1
C. Furtivo 1	5	S1-CF1	Copo de café, Modo operação Caminhos Furtivos 1 e Serviço 1
C. Furtivo 2	4	S1-CF2	Copo de café, Modo operação Caminhos Furtivos 2 e Serviço 1
T. Falha	8	S1-Fa	Copo de café, Modo operação Falha e Serviço 1

Com os arquivos XMLs preparados, cada um deles foi importado em sua respectiva suíte de testes. Em seguida, cada caso de testes foi aberto e as informações referentes a tempo estimado, técnica de teste, ferramenta de teste, tipo de teste, fase de teste e execução de testes foram definidos.

A última etapa do cadastro dos casos de testes foi à associação aos requisitos cadastrados. Através a ação “Selecionar requisitos” da TestLink, é possível definir a cobertura dos casos de testes em relação aos requisitos.

Todas as informações da especificação dos testes foram exportadas no formato Word através da TestLink e está disponível no Apêndice G.

7.4. Preparação do plano de teste

O plano de teste serve como um roteiro para a execução dos testes, definindo quais testes serão executados, por quem, em qual plataforma e em qual versão do produto.

Para a preparação do plano de testes, foram feitas as seguintes configurações na TestLink:

1. Criação de um plano de testes com nome de "Plano de teste 0001".
2. Criação de uma plataforma de testes com o nome de “Ambiente de homologação”
3. Criação de uma *baseline* denominada de “Realese 1”.
4. Através da ação “Adicionar / Remover Plataformas” do Plano de teste, o mesmo foi adicionado ao plano.
5. Através da ação “Adicionar / Remover Casos de testes” do Plano de teste, todos os 30 casos de testes do Serviço 1 foram adicionados.
6. Através da ação "Atribuir casos de testes", os casos de testes foram divididos, por tipo de comportamento do software representado pelas MEFs, da seguinte forma entre os usuários:
   - Ana: Serviço 1 - Normal e Serviço 1 – Falha;
   - Thiago: Serviço 1 - Exceção 5, Serviço 1 - Exceção 6 e Serviço 1 - Exceção 134;
   - Mauricio: Serviço 1 - Caminho furtivo 1 e Serviço 1 - Caminho furtivo 2.

7.5. Preparação do ambiente para o Gerenciamento dos defeitos

Para que os defeitos encontrados durante a execução dos testes possam ser registrados, a ferramenta Mantis foi configurada para receber essas informações.

Foi criado um projeto com nome de “Estudo de caso”. O nome entre os projetos da TestLink e da Mantis devem ser iguais, para que seja feita a unificação de ambos os projetos na base de conhecimento, conforme apresentado em capítulos anteriores.

As categorias de defeitos definidas pela VVTeste foram definidas no projeto criado. Além disso, os dois campos personalizados foram criados e associados a este projeto.

Para a utilização da ferramenta, foram criados 4 usuários. Ana, Mauricio e Thiago foram definidos com o papel de Relatores e Rodrigo com o papel de Desenvolvedor. Os usuários relatores foram criados com o mesmo *login* da TestLink, de acordo com a indicação feita pelo ambiente VVTeste.

7.6. Resultados da execução dos testes

Com os ambientes devidamente configurados e os testes planejados, a etapa seguinte é a execução dos testes. Os casos de testes foram divididos entre os testadores: Ana, Mauricio e Thiago.

Os testes foram iniciados por Ana, partindo dos casos de testes da suíte Serviço 1 – Normal. Conforme apresentado no relatório, os 2 casos de testes foram capazes de encontrar quatro defeitos, sendo que dois deles era o mesmo problema.

Os quatros defeitos foram registros na ferramenta Mantis da seguinte forma:

- Código 22: Falha ao acionar o botão de pouco açúcar;
- Código 23: Falha ao acionar o botão de muito açúcar;
- Código 24: Falha na luz "liga" ao preparar copo de café com pouco açúcar;

- Código 25: Falha na luz "liga" ao preparar copo de café com muito açúcar;

Os dois casos de testes foram registrados como "falha" e os defeitos foram associadas à execução dos casos de teste. O caso S1-N-1 ficou associado aos defeitos 22 e 24 e o S1-N-2 aos 23 e 25.

Em seguida, foi feita a execução dos testes gerados para o tipo de comportamento Tolerância a Falha pela testadora Ana. Esta suíte de testes foi capaz de encontrar um novo defeito, onde a situação de desligar a máquina, sem que o copo seja retirado era impossível de ser executada. Esse passo era descrito no caso de testes como *input(L,fACU)*. O defeito foi registrado na Mantis com o código 26.

Os casos de testes S1-Fa-1 e S1-Fa-2 foram considerados como bloqueados, pois tentavam realizar o passo *input(L,fACU)*, impossibilitando que os testes desses casos fossem concluídos. Os demais casos de testes dessa suíte foram executados com sucesso.

Os casos de testes associados ao testador Thiago foram executados. Conforme o relatório, esses casos de testes são capazes de encontrar os defeitos do “botão Pouco açúcar”, do “botão Muito açúcar” e a da Luz do “ligar” que pisca após o processamento do café, já encontrados por Ana. Porém, um defeito foi encontrado e registrado, referente ao não desligamento da máquina após os 10 segundos do preparo do café, quando o copo não é retirado do compartimento de preparo.

Para simular os testes paralelos, os defeitos já encontrados pela Ana serão novamente cadastrados por Thiago na Mantis, gerando assim os códigos 28, 29 e 30. O defeito relacionado ao não desligamento da máquina foi cadastrado com o código 27.

Com isso, os resultados dos testes de Thiago foram:

- O caso S1-Ex5-1 foi definido com “falha” e associado aos defeitos 27, 28 e 30;
- O caso S1-Ex5-2 foi definido com “falha” e associado aos defeitos 27, 29 e 30;
- O caso S1-Ex6-1 foi definido com “falha” e associado aos defeitos 27 e 28.
- O caso S1-Ex6-2 foi definido com “falha” e associado aos defeitos 27 e 29.
- O caso S1-Ex134-1 foi definido com “falha” e associado aos defeitos 28 e 30.
- O caso S1-Ex134-2 foi definido com “falha” e associado aos defeitos 29 e 30.
- Os casos S1-Ex5-3, S1-Ex5-4, S1-Ex5-5, S1-Ex5-6 e S1-Ex134-3 foram definidos com “sucesso”;

Para efeito de demonstração do estudo de caso, os testes designados ao Mauricio não são executados.

Após a conclusão dos testes de Thiago e Ana, foram visualizados alguns gráficos e tabelas nas ferramentas TestLink e Mantis. A Figura 7.1 demonstra um gráfico da relação do resultado dos testes em relação a as suítes de testes. Já a Figura 7.2 apresenta uma comparação entre os resultados de testes e o testador.

Na ferramenta Mantis, foi feita uma consulta na página principal sobre todos os defeitos cadastrados. O resultado dessa consulta é apresentado na Figura 7.3.

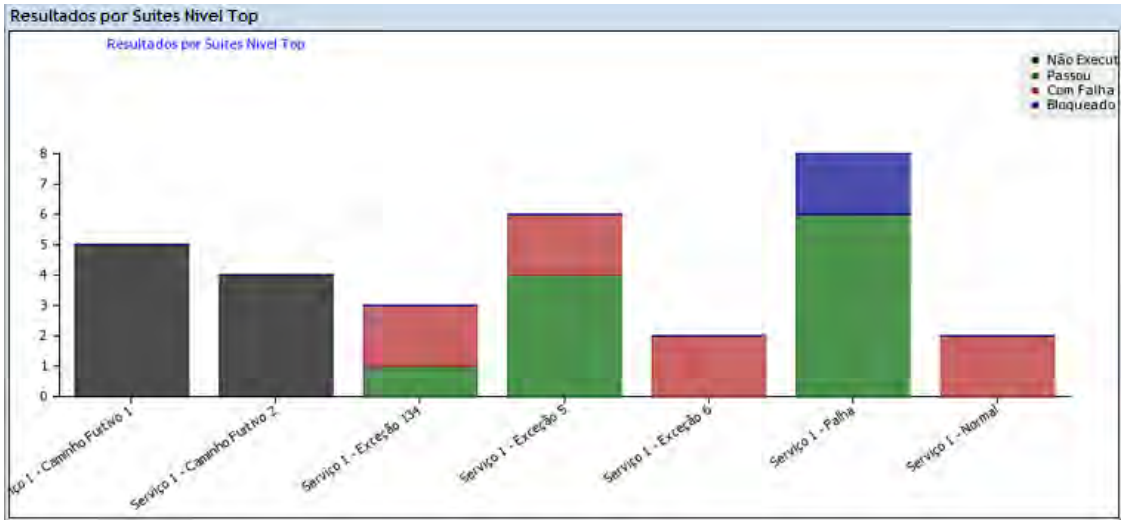

Figura 7.1 - Resultados por suítes de teste.

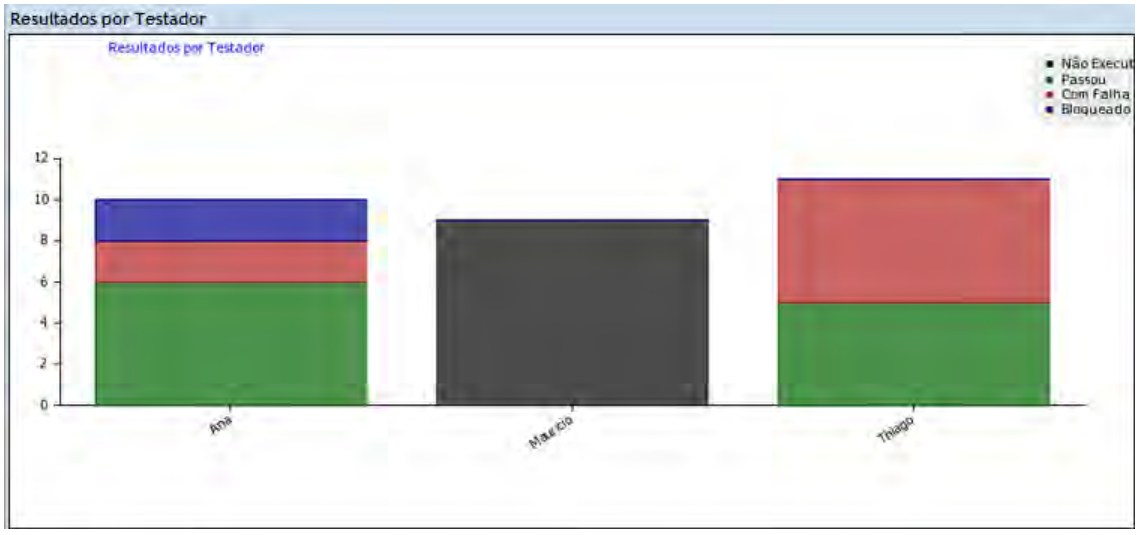

Figura 7.2 - Resultados por testador.

Visualizando Casos (1 - 9 / 9) [Imprimir Relatórios] [Exportar CSV] [Exportação para Excel] [Gráfico] [Exportação XML]

P	Núm	#	Categoria	Gravidade	Estado	Atualizado	Resumo
_	0000030		Outros	pequeno	novo	**2012-01-26**	Luz "liga" pisca após o processamento do café
_	0000029		Funcionalidade	**grande**	novo	**2012-01-26**	Botão muito açúcar não corresponde a luz muito-açúcar
_	0000028		Funcionalidade	**grande**	novo	**2012-01-26**	Botão pouco açúcar não corresponde a luz pouco-açucar
_	0000027		Funcionalidade	**obstáculo**	novo	**2012-01-26**	Não se consegue desligar a máquina sem que o copo seja retirado.
_	0000026		Segurança	pequeno	novo	**2012-01-26**	Não foi possível testar a falha no sensor de açúcar
_	0000025		Funcionalidade	pequeno	novo	**2012-01-26**	Falha na luz "liga" ao preparar copo de café com muito açucar
⌄	0000024		Funcionalidade	pequeno	novo	**2012-01-26**	Falha na luz "liga" ao preparar copo de café com pouco açucar
_	0000023		Funcionalidade	**grande**	novo	**2012-01-26**	Falha ao acionar o botão de muito açucar
_	0000022		Funcionalidade	**grande**	novo	**2012-01-26**	Falha ao acionar o botão de pouco açucar

novo | retorno | admitido | confirmado | atribuído | resolvido | fechado

Figura 7.3 - Defeitos registrados na Mantis.

7.7. Correção de alguns defeitos

O usuário definido para a correção dos defeitos foi denominado como Rodrigo. Por conta disso, os incidentes 22, 23, 24, 25, 28, 29 e 30 foram atribuídos a ele.

Para efeito de demonstração, foi simulado que o desenvolvedor Rodrigo fez a correção dos problemas das luzes do pouco açúcar e de muito açúcar. Na analise dos defeitos, ele detecta que existem defeitos duplicados.

Por isso, os defeitos 22, 23, 28 e 29 tiveram a sua situação alterada para “Resolvido”, sendo que os defeitos 22 e 23 tiveram como resolução a opção “Duplicada”, pois relatavam os defeitos registrados nos defeitos 28 e 29, que por sua vez, tiveram a sua resolução como “Corrigida”.

Por conta da integração entre TestLink e Mantis, é possível também acompanhar a correção dos defeitos pela TestLink. No relatório de matriz de consulta, a ferramenta exibe uma listagem com a situação da execução do caso de teste, quem testou e os defeitos associados, que mudam de cor a cada alteração de situação da Mantis.

7.8. Exportação dos dados para base de conhecimento

O processo de testes normalmente é executado até que todos os defeitos sejam corrigidos e os testes de reavaliação sejam feitos. Para efeito do estudo de caso, os dados já serão exportados antes do término dos testes, para que se tenha uma maior quantidade de situações entre os registros gerados.

Outro fator importante para permite e exportação antes do término dos testes, é que o módulo MID permite que no momento da inclusão dos dados na base de conhecimento, seja definido se os dados devem ser atualizados.

A exportação dos dados é feita através do módulo VVTeste – MAD, onde são definidos os nomes dos bancos onde estão os dados da TestLink e da Mantis. Em seguida, a ferramenta conecta aos bancos de dados e carrega uma lista com todos os projetos cadastrados em cada uma das ferramentas.

Como foi feita a padronização dos nomes dos projetos, deve ser selecionado a opção “Estudo de caso” em ambas as listas da TestLink e da Mantis. Em seguida, basta selecionar o local e o nome dos quatros arquivos XMLs que serão gerados pelo módulo.

7.9. Importação dos dados na base de conhecimento

A importação dos dados é feita através do módulo VVTeste – MID, onde os arquivos XMLs gerados pela MAD são processados e armazenados na base de conhecimento.

Devem-se selecionar os quatro arquivos gerados pela MAD em seus respectivos campos, e definir o que fazer quando o registro já existir na base de conhecimento (Substituir pelo mais novo ou ignorar o mais novo).

Os dados serão inseridos na base de conhecimento e já poderão ser consultados através do módulo MCD ou de outra ferramenta externa.

7.10. Consulta aos dados na base de conhecimento

A consulta à base de conhecimento VVTeste é feita através do módulo MCD. Neste módulo é possível utilizar as 26 consultas pré-definidas ou até mesmo criar novos gráficos através do recurso de personalização de comando SQL.

Apesar do estudo de caso não gerar um grande volume de informações, já é possível realizar algumas análises comparativas. Algumas consultas que foram feitas são capazes de demonstrar, por exemplo, a relação entre as técnicas de

testes e os defeitos encontrados e os tipos de requisitos em relação às causas dos defeitos, conforme Figura 7.4. Esse tipo de comparação não poderia ser feita diretamente na Mantis ou na TestLink.

7.4 – Gráfico gerado pela MCD a partir do estudo de caso.

Assim que outros projetos forem importados a base de conhecimento, os resultados alcançados serão uma fonte mais precisa de lições aprendidas e de estatísticas de projetos de testes.

7.11. Avaliação do estudo de caso

O estudo de caso abordando os testes em uma máquina de café foi capaz de passar por todas as ferramentas propostas pelo ambiente VVTeste, demonstrando a cobertura feita nas atividades de testes.

Através da modelagem das MEFEs, a Condado gerou um conjunto de casos de testes que foi capaz de cobrir as validações necessárias no software em testes. Além disso, o módulo MICT aumentou a eficiência dos cadastros desses casos de testes gerados na ferramenta de gerenciamento de testes.

A TestLink se mostrou uma ótima ferramenta de gerenciamento de testes. Através dela, os requisitos puderam ser cadastrados, os casos de testes puderam ser organizados através de suítes de testes, a imagem dos modelos puderam ser anexados e disponibilizados a todos os usuários, os testes puderam ser divididos, os resultados registrados e todo o acompanhamento dos testes puderam ser feito em tempo real.

O fato de ser uma ferramenta Web e com controle de acesso aos dados, a ferramenta comprovou a sua qualidade para realização da atividade de gerenciamento de testes.

A ferramenta Mantis comprovou a sua eficiência no gerenciamento de defeitos. O controle das fases de resolução do defeito pôde ser feita, incluindo papéis de relatores e desenvolvedores. No estudo de caso não foi utilizado, porém mais papéis podem ser adicionados no fluxo, como o analista que avalia se o problema relatado realmente é um defeito que precisa ser corrigido.

A integração da TestLink e da Mantis permitiu que os defeitos pudessem ser associados aos casos de testes, facilitando o acompanhamento da correção dos defeitos.

Os módulos MAD e MID foram capazes de processar as informações adquiridas dos bancos de dados da TestLink e Mantis, atualizando a base de conhecimento. Além disso, o propósito de interoperabilidade proposto pelo VVTeste, permitiria ainda que outros defeitos pudessem ser importados na base de conhecimento através de arquivos XMLs.

O estudo de caso não gerou grande quantidade de informações capazes de permitir análises mais elaboradas, porém foi suficiente para mostrar que os dados armazenados em um local único permite a obtenção de informações cruzadas disponibilizadas por cada uma das ferramentas.

As atividades, a organização e as evidências exigidas pelos modelos de referência MPS.br e CMMI para os processos de Verificação e Validação, principalmente quando a técnica utilizada é a de testes são atendidas pelo ambiente VVTeste, se tornando uma alternativa viável para este fim.

8 CONCLUSÃO

Nessa dissertação foram apresentados conceitos sobre as atividades de testes de software existentes nos processos de verificação e validação. A partir de alguns estudos e dirigido por alguns modelos de referência de desenvolvimento de software, este trabalho criou um ambiente capaz de auxiliar nas atividades de um laboratório de verificação e validação de software.

Este capítulo apresenta as conclusões alcançados por este trabalho durante os seus estudos e implementações. Para tanto, está dividida em: Trabalho realizado, Contribuições, Limitações e Trabalhos futuros.

8.1. Trabalho realizado

Este trabalho apresentou um ambiente, denominado de VVTeste, para dar apoio aos processos de verificação e validação de software. Estudos realizados, principalmente em modelos como o CMMI e a MPS.br, apresentaram as atividades, objetivos e os resultados esperados durante a realização desses processos. Diante disso, o ambiente proposto integrou as atividades de geração automática de casos de testes, de gerenciamento de testes e defeitos, e uma base de conhecimento como forma de obter lições aprendidas e estatísticas de projetos já realizados.

Apesar desses conceitos já serem estudados e utilizados pela comunidade de software, a transformação do "*O que*" descrito pelos modelos, para o "*Como*", ainda não foi amplamente explorada, permitindo que novas formas possam ser buscadas e implementadas.

O presente trabalho utilizou três ferramentas livres, que foram integradas através de módulos criados, em Java, por este trabalho. A ferramenta Condado foi utilizada para a geração automática de casos de testes através da modelagem de MEFEs. Os casos criados por essa ferramenta são

transformados em arquivos XMLs e importados na TestLink, ferramenta para o gerenciamento dos testes. Para o gerenciamento dos defeitos, foi utilizada a ferramenta Mantis, que também foi integrada a TestLink.

A configuração e personalização das informações da Mantis e da TestLink foram feitas tomando como referências algumas normas. Porém isso não impossibilita que em alguns projetos específicos, os valores de alguns campos sejam alterados e adaptados a nova necessidade.

Além disso, o ambiente VVTeste integra uma base de conhecimento de testes de software única, que armazena todas as informações geradas durantes os projetos de testes, e que pode ser consultada para compor relatório de lições aprendidas. As informações são atualizadas nessa base de conhecimento através dos dados armazenados nas ferramentas TestLink e Mantis, o que as tornas seguras e de fácil controle.

Um protótipo para demonstrar como a base de conhecimento pode ser utilizada foi desenvolvimento por este trabalho. Neste protótipo, são disponibilizados vários gráficos, cruzando as mais diversas informações existentes na base de conhecimento, inclusive entre diversos projetos de testes.

O estudo de caso apresentado neste trabalho apresenta o potencial do ambiente proposto. A integração, a facilidade de operação e a segurança na geração e manipulação das informações proporcionam a um laboratório de testes, do INPE ou de qualquer outra organização/empresa, um ambiente eficiente para a realização dessas atividades e capacidade de gerenciar dados históricos.

Além disso, outro ponto relevante do ambiente proposto foi à busca por interoperabilidade nas suas integrações. Isso permite que o ambiente seja

evoluído e adaptado a novas necessidades. Como exemplo dessa característica pode-se ressaltar dois pontos.

A primeira é a importação dos casos de testes na TestLink, o que permite que os casos de testes possam ser gerados em uma ferramenta diferente da Condado e também importadas no planejamento dos testes. A segunda e a inclusão dos dados na base de conhecimento através de arquivos XMLs, isso permite que caso existam projetos que não tenham utilizado a TestLink ou Mantis, sejam importadas na base de conhecimento, permitindo a unificação de várias fontes de dados.

A utilização de ferramentas livres no ambiente VVTeste torna o ambiente viável a qualquer laboratório de testes, pois não requer investimentos financeiros na aquisição de ferramentas e atendem as necessidades práticas das atividades realizadas.

Se comparado aos trabalhos correlatos citados anteriormente, a VVTeste apresenta uma proposta diferente, pois propõem a integração de várias ferramentas, consumindo as principais vantagens de cada uma dela, alinhado com os modelos. Além disso, propõe uma forma de buscar as lições aprendidas em projetos anteriores, utilizando-as como estatísticas.

Este trabalho já constatou o interesse da comunidade em propostas de trabalhos deste gênero:

- Foi desenvolvida uma produção acadêmica, de publicação internacional, na área de confiabilidade de software (REIS, ET AL, 2011), onde a arquitetura da VVTeste foi apresentada.

- No Encontro Brasileiro de Testes de Software de 2012, um relato de experiência da utilização da Mantis foi apresentado (REIS, ET AL, 2012).

- Um artigo foi aceito no SpaceOps 2012, congresso mundial sobre tecnologias espaciais, e será apresentado em Junho de 2012 na Suécia apresentando a arquitetura do VVTeste e como ele atende o modelo de referência CMMI.

8.2. Contribuições

Este trabalho faz contribuições à área de qualidade de software, principalmente as atividades de testes software. As principais são:

1. Adaptabilidade do ambiente a diversos tipos de laboratórios de testes;
2. Atendimento aos resultados esperados dos modelos, representando uma forma de realizar as atividades proposta por elas;
3. A integração dos conceitos de Geração automática de casos de testes, Gerenciamento de testes e Gerenciamento de defeitos, demonstrando a aplicação e as vantagens de trabalhar com essas informações;
4. Assim como os conceitos, a integração das ferramentas utilizadas, fazendo que os dados não precisem ser recadastrados a cada etapa do processo, aumentando a qualidade, a segurança e a eficiência dos processos.
5. Criação de uma base de conhecimento de testes, modelada considerando as principais informações geradas durante uma bateria de testes e integrada às ferramentas utilizadas durante o processo de testes.
6. Criação de uma ferramenta para a consulta das informações armazenadas na base de conhecimento

8.3. Limitações

Este trabalho não levou em considerações alguns pontos, que por conta da delimitação do escopo proposto não foram explorados. A principal é a não integração a ferramenta de automação de testes. Esse assunto não foi

abordado, pois exige que a ferramenta utilizada esteja muito bem alinhada com o software que está em testes.

Além disso, o trabalho se limitou a geração automática de casos de testes através de MEFEs. Na comunidade existem outras formas e conceitos que poderiam ser tratados em conjunto.

O escopo do trabalho também foi limitado aos processos de verificação e validação, mas especificamente aos testes de software, porém é sabido que existem outros fatores que são essenciais para que o software alcance a qualidade esperada. O próprio ambiente criado poderia ser utilizado para o auxilio de outras atividades do desenvolvimento de um software.

8.4. Trabalhos futuros

Por se tratar de parte de um processo de desenvolvimento de software, o ambiente pode ser evoluído de diversas formas.

A geração automática de casos de testes não precisa estar limitada a Condado, permitindo que outras ferramentas que utilizem outras técnicas sejam integradas ao ambiente VVTeste.
A utilização de ferramentas de automação de testes também é um ponto que pode ser melhorado ao ambiente. A criação de alguma ferramenta que seja capaz de automatizar a execução de casos de testes criados a partir de MEFEs aumentaria ainda mais a eficiência dos testes.

Por fim, outros conceitos como a gerência de requisitos e de configuração, podem ser abordados e adaptados ao ambiente, permitindo que o controle sobre as alterações no software sejam feitas de maneira mais segura.

REFERÊNCIAS BIBLIOGRÁFICAS

AMBROSIO, A. M. **COFI**: uma abordagem combinando teste de conformidade e injeção de falhas para validação de software em aplicações espaciais. 2005. 209 p. (INPE-13264-TDI/1031). Tese (Doutorado em Computação Aplicada) - Instituto Nacional de Pesquisas Espaciais, São José dos Campos, 2005. Disponível em: <http://urlib.net/sid.inpe.br/MTC-m13@80/2005/09.06.13.34>. Acesso em: 09 abr. 2012.

ATIFS. **Ferramenta condado.** 2005. Disponível em: http://www.inpe.br/atifs/html/ferramenta_condado.html . Acesso em: 17 de Janeiro de 2012.

BASTOS, A.; CRISTALLI, R.; MOREIRA, T.; RIOS, E. **Base de conhecimento em teste de software**. São Paulo : Martins, 2007.

BERGERON, B. **Essentials of knowledge management**. Hoboken : John Wiley & Sons, 2003.

BÉZIVIN, JEAN, BRUNELIÈRE, HUGO e JOUAULT, FRÉDÉRIC. Model Engineering Support for Tool Interoperability. In: WISME,4., 2005, Jamaica. **Proceedings...** Disponível em: http://planet-mde.org/wisme-2005/ModelEngineeringSupportForToolInteroperability.pdf. Acesso em: 29 de janeiro de 2012.

BOEHM, B. **Software engineering economics**. Englewood Cliffs N.J.: Prentice-Hall, 1981.

BRAGA, KAREN e PRETZ, EDUARDO. **Conhecimento e teste exploratório:** um modelo de captação e execução. Trabalho de Conclusão de Curso – Feevale, Nova Hamburgo, 2010.

BRAZILIAN SOFTWARE TESTING QUALIFICATION BOOARD (BSTQB). **Base de conhecimento para certificação de testes** [S.l]: Comissão Internacional para Qualificação de Teste de Software. 2005.

CARDIAS JUNIOR; BRITO, A. Uma análise avaliativa de ferramentas de software livre no contexto da implementação do processo de gerência de requisitos do MPS.BR. In: WER, 10., 2010, Cuenca, Equador. **Anais...** Disponível em: http://wer.inf.puc-rio.br/WERpapers/pdf_counter.lua?wer=WER10&file_name=brito.pdf . Acesso em: 29 de janeiro de 2012.

CMMI, Product Team. **CMMI® for development**. Version 1.3. Software Enginnering Institute, 2010.

COLLINS, E.F.; LOBÃO, L.M. A. Experiência em automação do processo de testes em ambiente ágil com scrum e ferramentas OpenSource. In: SIMPÓSIO BRASILEIRO DE QUALIDADE DE SOFTWARE,9., 2010, Belem. **Anais...** Disponível em: http://www.lbd.dcc.ufmg.br/colecoes/sbqs/2010/RL4_eliane_collins.pdf9. Acesso em: 28 de janeiro de 2012.

CROSBY, P.B.A. **Quality is free:** the art of making quality certain. New York : McGraw-Hill, 1979.

DELAMARO, M. E.; MALDONADO, J. C.; JINO, M. **Introdução ao teste de software**. Rio de Janeiro: Elsevier, 2007.

DEUTSCH, LESLIE J. Resolving the Cassini/Huygens relay radio anomaly. Jet Propulsion Laboratory. In: AEROSPACE CONFERENCE. 2002, Pasadena. **Proceedings...** Disponível em: http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=1035262. Acesso em: 29 de janeiro de 2012.

ELMASRI, R.; NAVATHE, S.B. **Fundamentals of database systems** 2. ed. Menlo Park : Addison Wesley, 1994.

FEIGENBAUM, A. V. **Controle da qualidade total.** São Paulo : Makron Books, 1994.

GILL, A. **Introduction to the theory of finite state machines.** New York : McGraw-Hill, 1962.

GUERRA, A.C..; COLOMBO, R. M. T. **Tecnologia da informação -** qualidade de produto de software. Brasília: MCT/SEPIN, 2009.

HECHT, M.; BUETTNER, D. Software testing in space programs. **CrossLink Fall.** v.6, n.3, 2005. Disponível em: http://www.aero.org/publications/crosslink/fall2005/06.html. Acesso em: 29 de janeiro de 2012.

HOPKROFT, J.E, **Introduction to automata theory, language and computation**. Reading, MA: Addison-Wesleym, 1979.

HEWLETT-PACKARD. **HP test management**. 2012. Disponível em: http://www8.hp.com/us/en/software/software-solution.html?compURI=tcm:245-936908&pageTitle=test-management. Acesso em: 29 de janeiro de 2012.

INTERNATIONAL BUSINESS MACHINES. **IBM software** - rational test manager. 2012. Disponível em: http://www-

01.ibm.com/software/awdtools/test/manager/. Acessado em: 29 de janeiro de 2012.

IEEE, Computer Society. **IEEE standard glossary of software engineering terminology**. IEEE-SA Standards, 1983.

IEEE 1044, Computer Society. **IEEE 1044 standart classification for software anomalies**. IEEE-SA Standards, 2009.

INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS (INPE). **Notícias CBERS**. São José dos Campos, 2010. Disponível em: http://www.cbers.inpe.br/noticias/index.php?cod=not183. Acessado em: 29 de janeiro de 2012.

THE INTERNATIONAL ORGANIZATION FOR STANDARDIZATION AND THE INTERNACIONAL ELECTROTECHNICAL COMISSION (ISO/IEC).**ISO/IEC 12207:2008 Systems and software engineering** — software life cycle processes. Geneve, 2008.

JURAN, J. M. **Quality control handbook**. New York : MC Graw Hill, 1988.

KANER, C.; BACH, J.; PETTICHORD, B. **Lesson learned in software testing.** New York: : Wiley, 2001.

MANINI, E. P. **Héstia** – ferramenta de apoio a teste de software com base em casos de uso. Trabalho de Conclusão de Curso - Centro Universitário Ritter dos Reis, Porto Alegre, 2009.

MANTISBT. **Mantis.** 2000. Disponível em: http://www.mantisbt.org. Acesso em: 29 de janeiro de 2012.

MARTINS, E.; SABIÃO, S. B.; AMBROSIO, A. M. **ConData:** a tool for automating specification-based test case generation for communication systems. **Software Quality Journal**. v.8, 1999.

MICHAELIS. **Dicionário online Michaelis**. 2012. Disponível em: http://michaelis.uol.com.br/moderno/portugues/index.php?lingua=portugues-portugues&palavra=requisito&CP=146197&typeToSearchRadio=exactly&pagRadio=50. Acesso em: 17 de Janeiro de 2012.

MICROSOFT. **Overview of visual studio test professional**. 2012. Disponível em: http://www.microsoft.com/visualstudio/en-us/products/2010-editions/test-professional/overview. Acesso em: 29 de janeiro de 2012.

MOLINARI, L. **Testes funcionais de software**. Florianopólis: Visual Books, 2008.

MORAIS, M. H. E.; AMBRÓSIO, A. M. **Metodologia CoFI (conformance and fault injection) aplicada a um exemplo didático**. São José dos Campos : INPE, 2009. (INPE-16635-RPQ/840).

MOREIRA FILHO, TRAYAHÚ e RIOS, E. **Projeto & engenharia de software - teste de software**. Rio de Janeiro: Alta Books, 2003.

MYERS, GLENFORD. **The art of software testing**. Nova York: Wiley, 1979.

NONAKA, I.; TAKEUCHI, H. **Criação de conhecimento na empresa**. Rio de Janeiro : Campus, 1997.

OPENSOURCETESTING.org. **OpenSourceTesting.org**. 2012. Disponível em: http://www.opensourcetesting.org. Acesso em: 29 de Janeiro de 2012.

POL, M, TEUNISSEN, R.; VEENENDAAL, E. **Software testing:** a guide to the TMap approach. Boston : Addison-Wesley, 2002.

PRESSMAN, R. S. **Engenharia de software**. Rio de Janeiro: McGraw Hill, 2002.

REIS, M. F.; AMBROSIO, A. M.; FERREIRA, M. Generation Environment, Execution and Test Management Software Embedded. In: LATIN-AMERICAN SYMPOSIUM ON DEPENDABLE COMPUTING, 5. (LADC), 2011, São José dos Campos**. Proceedings...** São José dos Campos: INPE, 2011. v. Suplemental. CD, On-line. Disponível em: <http://urlib.net/8JMKD3MGP8W/39G23K2>. Acesso em: 09 abr. 2012.

REIS, M. F.; AMBROSIO, A. M.; FERREIRA, M. Utilização da ferramenta Mantis para gerenciamento de defeitos de testes de software. In: Encontro Brasileiro de Testes de Software, 6. (EBTS), 2012, Recife**. Anais...** CD.

ROBINSON, H. Finite state model-based testing on a shoestring. In: STAR West, 1999. **Proceedings...** Disponível em: http://www.geocities.com/model_based_testing/shoestring.htm. Acesso em: 29 de Janeiro de 2012.

ROCHA, A.R.C., MALDONADO, J.C.; WEBER, K.C. **Qualidade de software – teoria e prática**. São Paulo: Prentice Hall, 2001.

SAKAR. **Inovação:** quebrando paradigmas para vencer. São Paulo: Saraiva, 2007.

SANTHANAM, P.; HAILPERN, B. Software debugging, testing and verification. **IBM System Journal**. p. 4 – 12, 2002.

SOFTEX. **Guia de implementação** – Parte 4: fundamentação para implementação do nível D do MR-MPS, [S.l]; Softex, 2011.

SOFTEX. **MPS.BR** - melhoria de processo do software brasileiro - guia geral. [S.l]: Softex, 2011.
TESTLINKCOMMUNITY. **TestLink**. 2005. Disponível em: http://www.teamst.org/. Acesso em: 29 de Março de 2011.

THELIN, T. **Empirical evaluations of usage-based reading and fault content estimation for software inspections**. Tes de doutorado. Lund University, Sweden, 2002.

TIAN, J. Software quality engineering **- testing, quality assurance, and quantifiable improvement**. Hoboken : John Willey & Sons, 2005.

TOMINAGA, J. **Simulador de satélites para verificação de planos de operações em voo.** 2010. 174 p. (sid.inpe.br/mtc-m19@80/2010/05.24.18.55-TDI). Dissertação (Mestrado em Computação Aplicada) - Instituto Nacional de Pesquisas Espaciais, São José dos Campos, 2010. Disponível em: <http://urlib.net/8JMKD3MGP7W/37HL3J8>. Acesso em: 09 abr. 2012.

APÊNDICE A – Interfaces do módulo MICT

A primeira aba do módulo MICT é responsável pela aquisição do script de teste da Condado. Assim que o script é processado, o módulo apresenta a quantidade de casos de testes encontrados no arquivo. A Figura A.1 apresenta a interface desta aba.

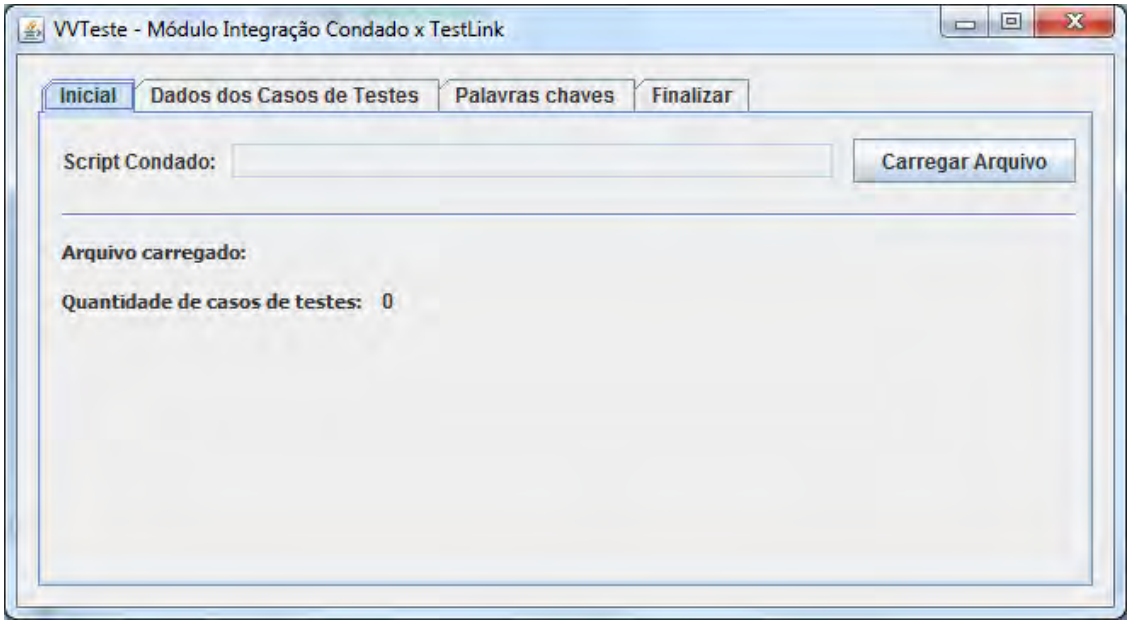

Figura A.1 - MICT - Aba Inicial

Na aba Dados dos Casos de Testes, o projetista de teste poderá definir algumas informações que serão definidas aos casos de testes. O campo “Prefixo do caso de teste” juntamente com o campo “Número iniciais”. Eles formam o identificador do caso de testes. Em um exemplo com vários casos de testes, o primeiro caso da lista seria identificado como Prefixo + Numero Inicial e os demais casos teriam o campo número inicial incrementado a cada caso de testes processado.

Os campos “Resumo” e “Pré condições” são definidos uma única vez e suas informações são replicadas a todos os casos de testes processados no módulo MICT. A Figura A.2 apresenta a interface na aba “Dados dos Casos de Testes”.

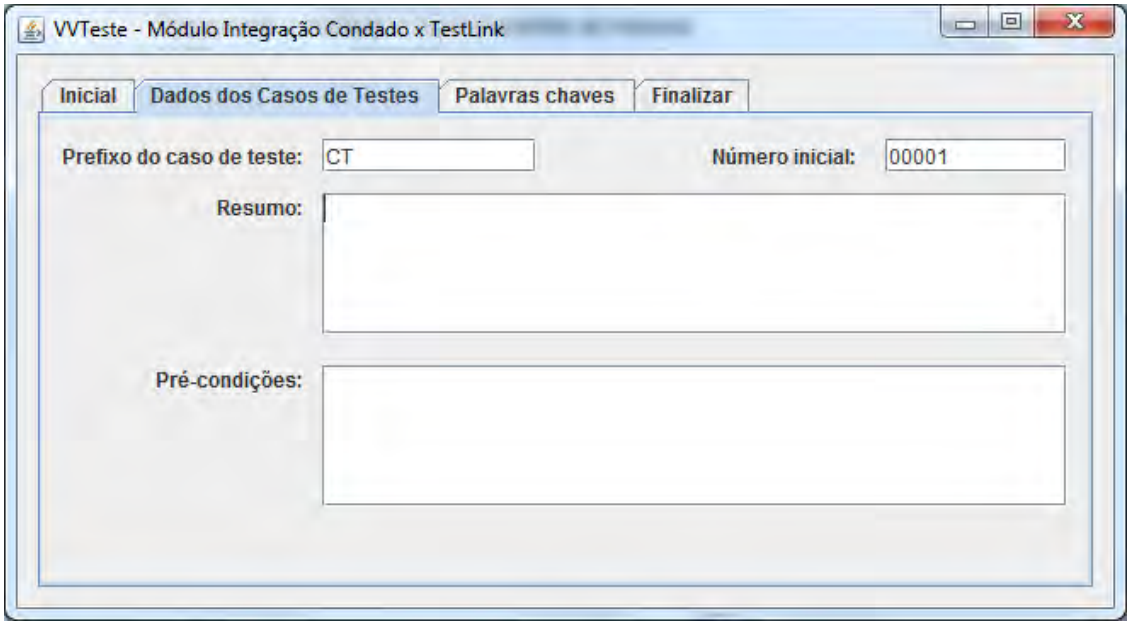

Figura A.2 - MICT - Aba Dados dos Casos de Testes.

A terceira aba permite a definição de cinco palavras chaves e seus respectivos comentários. As palavras definidas serão associadas a todos os casos de testes exportados. A Figura A.3 apresenta a aba onde essas informações são preenchidas.

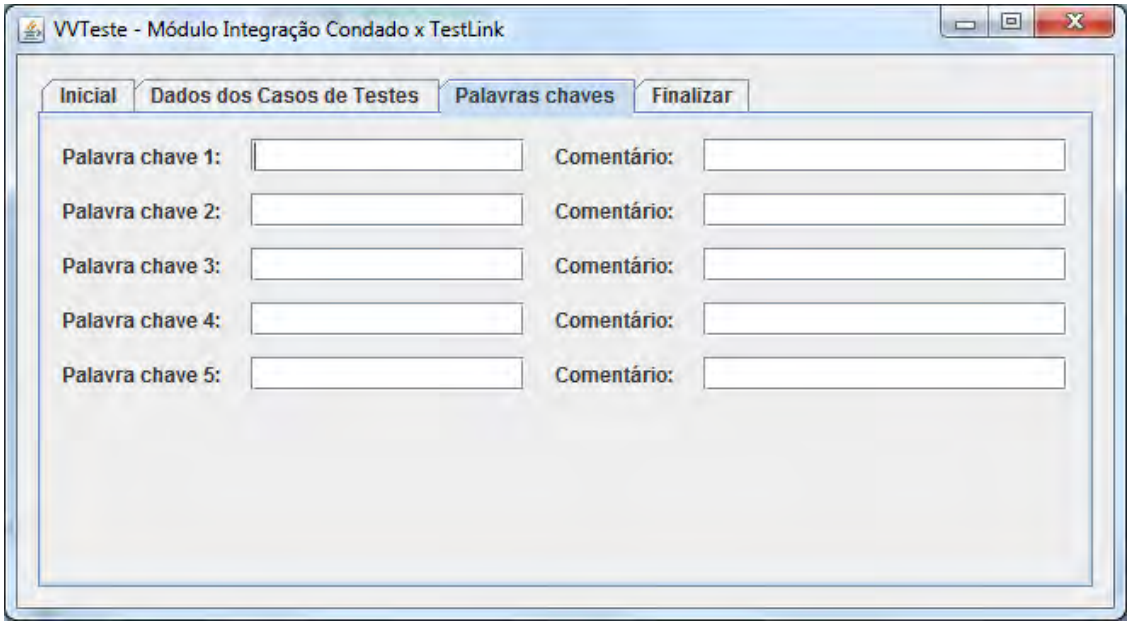

Figura A.3 - MICT - Aba Palavras chaves.

O último passo para a conversão do script Condado em arquivo XML é a definição do local onde o mesmo deve ser salvo, conforme apresentado na Figura A.4.

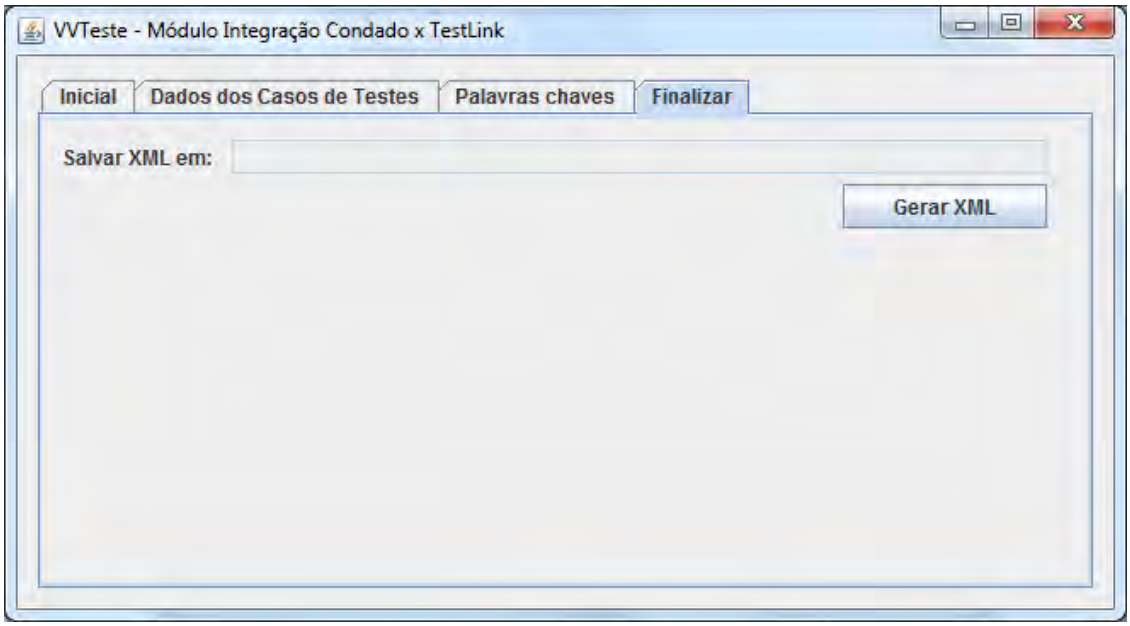

Figura A.4 - MICT - Aba Finalizar.

APÊNDICE B – Interfaces do módulo MAD

O módulo MAD tem a finalidade de extrair as informações dos bancos de dados da TestLink e da Mantis. Sua a interface é dividida em duas partes, de funcionamento semelhante, porém cada parte acessa um tipo de informação.

A parte superior da interface, disponível na Figura B.1, permite a conexão com o banco de dados da ferramenta TestLink. O campo “Nome do banco de dados TestLink” deve ser preenchido com o nome dado ao banco dessa ferramenta no momento de sua instalação. A ferramenta sugere o nome padrão que é “testlink”, porém se a ferramenta foi instalada com um nome diferente, poderá ser alterado.

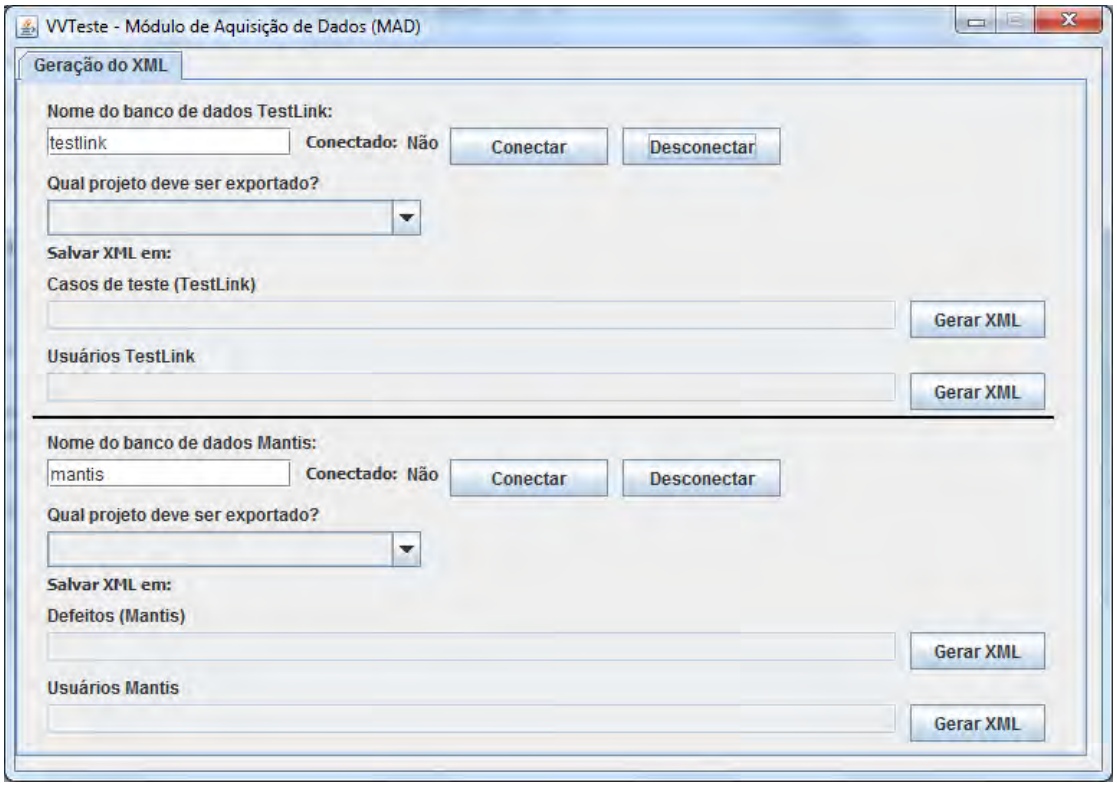

Figura B.1 - Interface da MAD.

Os botões “Conectar” e “Desconectar” visam estabelecer a conexão com o banco de dados. Assim que a ferramenta consegue acessar os dados do banco, ela preenche o campo “Qual projeto deve ser exportado?” com todos os

projetos de testes cadastrados na TestLink. O usuário deverá escolher um projeto para realizar a exportação.

Por fim, o usuário deve definir onde os arquivos XMLs com as informações do projeto de testes e dos usuários cadastrados devem ser salvos.

A parte inferior, referente à ferramenta Mantis, tem o mesmo funcionamento citado anteriormente, porém com a finalidade de exportar os defeitos cadastrados. O nome padrão desse banco de dados é “mantis”.

APÊNDICE C – Interfaces do módulo MID

O módulo MID tem a finalidade de consumir os arquivos XMLs gerados pelo módulo MAD e atualizá-los na base de conhecimento. O banco de dados deve obrigatoriamente ter o nome “VVTeste”.

Para isso, o módulo disponibiliza uma interface, conforme apresentado da Figura C.1, quatro campos para que o usuário possa selecionar os arquivos. Os arquivos devem ser selecionados em seus respectivos campos.

Além disso, não é de preenchimento obrigatório a seleção de todos os campos, o usuário pode optar por processar apenas os defeitos ou o planejamento dos testes.

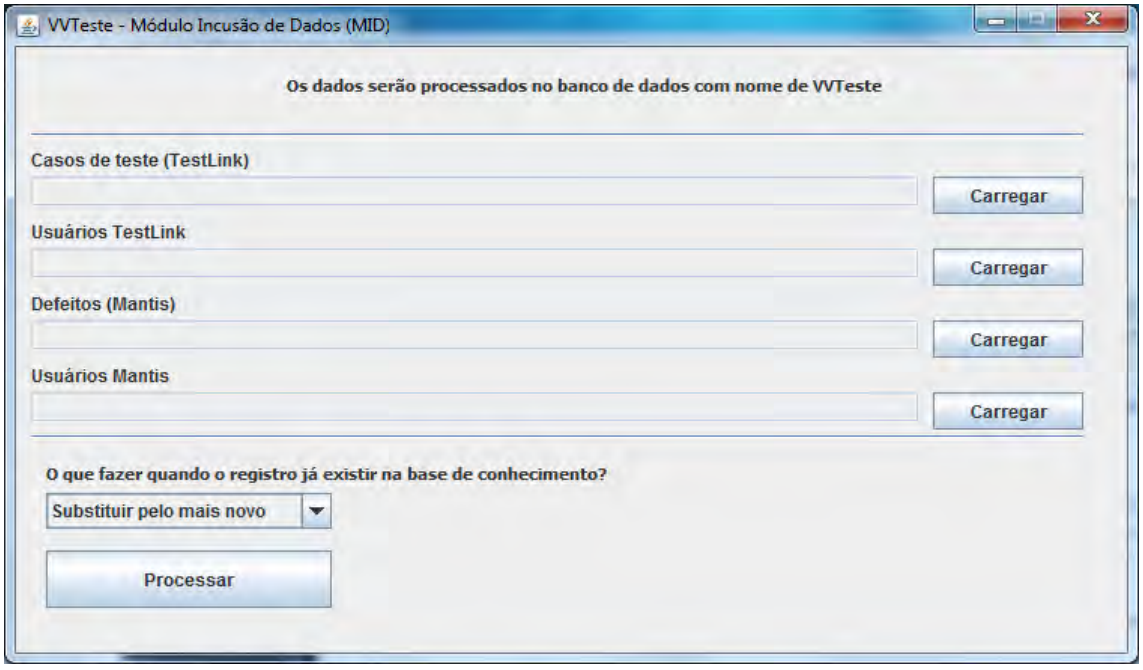

Figura C.1 - Interface da MID.

O campo “O que fazer quando o registro já existir na base de conhecimento” disponibiliza duas opções ao usuário.

A primeira é “Substituir pelo mais novo”, onde a ferramenta ao detectar que um defeito, caso de testes, usuário, ou qualquer outra informação já exista na base de conhecimento, irá atualizar as informações e não inserir, o que evita a duplicidade das informações.

A segunda é “Manter o já existente”, onde o dado já existe na base de conhecimento terá prioridade sobre o dado que está sendo inserido, sendo assim, não haverá alteração no banco neste caso.

Após o processamento das informações, o módulo exibe uma mensagem informando que os dados foram atualizados com sucesso. Caso ocorra algum problema, o sistema irá exibir a mensagem de erro.

APÊNDICE D – Modelo lógico e físico da base de conhecimento

Na seção 5.3.5 é apresentado um diagrama de entidade e relacionamento da base de conhecimento de testes da VVTeste. Neste apêndice são apresentados os modelos lógicos e físicos.

Assim como o modelo conceitual apresentado na seção 5.3.5, o modelo lógico, Figura D.1, também foi desenvolvido utilizado a ferramenta br.Modelo.

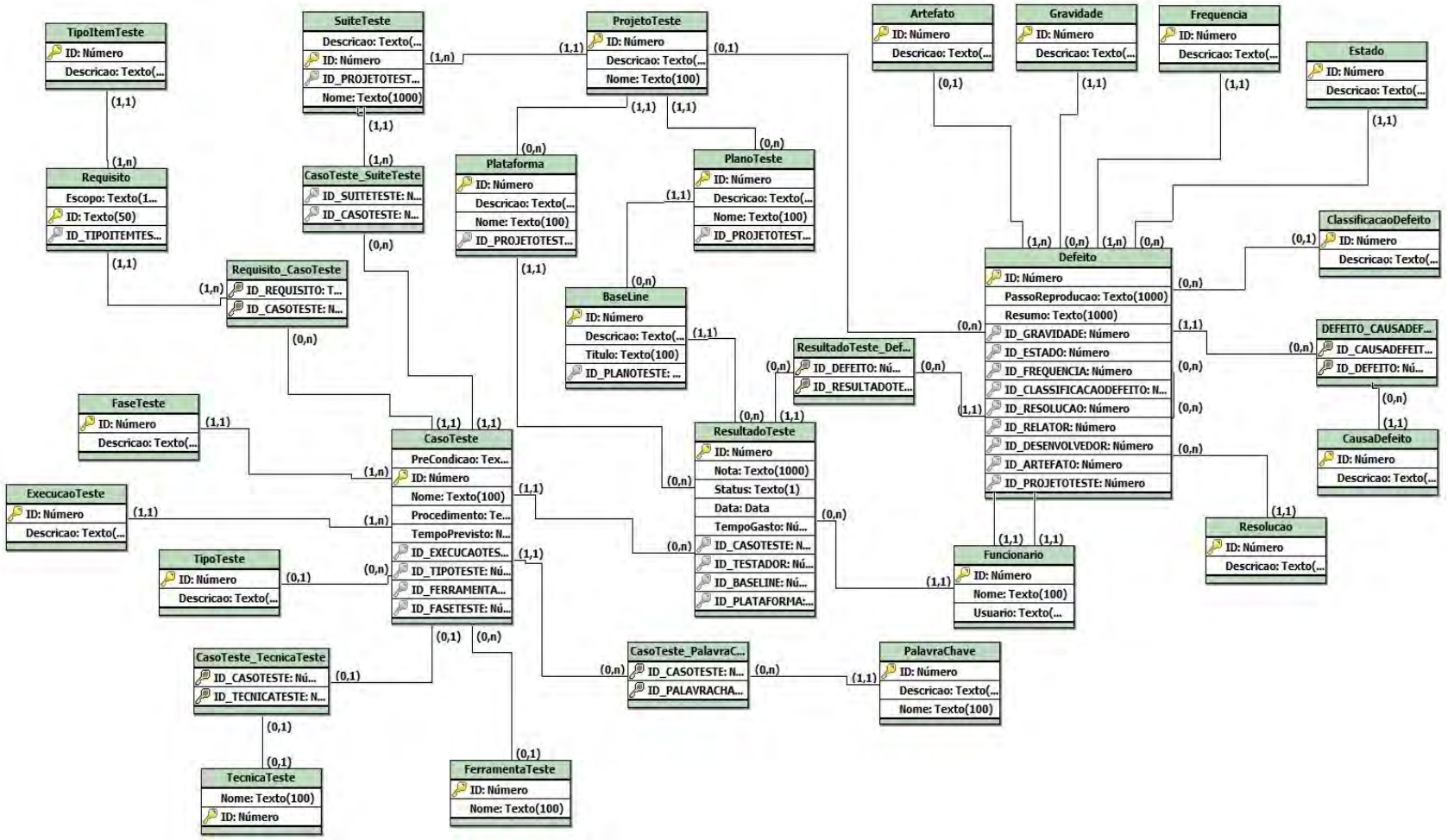

Figura D.1 - Modelo Lógico da Base de conhecimento.

O script gerado no modelo físico atende o padrão ANSI e foi executado no Sistema Gerenciador de Banco de Dados MySQL. O nome do banco de dados deve ser “VVTeste”. A seguir é disponibilizado todo o script desse banco de dados.

```
ALTER DATABASE VVTeste DEFAULT CHARACTER SET utf8 COLLATE utf8_general_ci;

CREATE TABLE Estado (
ID Int(10) AUTO_INCREMENT PRIMARY KEY,
Descricao VARCHAR(100)
) ENGINE= InnoDB;

CREATE TABLE SuiteTeste (
Descricao VARCHAR(1000),
Nome VARCHAR (1000),
ID Int(10) AUTO_INCREMENT PRIMARY KEY,
ID_PROJETOTESTE Int(10)
) ENGINE= InnoDB;

CREATE TABLE CasoTeste (
PreCondicao VARCHAR(1000),
ID Int(10) PRIMARY KEY,
Nome VARCHAR(100),
Procedimento VARCHAR(1000),
TempoPrevisto Int(10),
ID_EXECUCAOTESTE Int(10),
ID_TIPOTESTE Int(10),
ID_FERRAMENTATESTE Int(10),
ID_FASETESTE Int(10)
) ENGINE= InnoDB;
```

```
CREATE TABLE ProjetoTeste (
ID Int(10) AUTO_INCREMENT PRIMARY KEY,
Descricao VARCHAR(1000),
Nome VARCHAR(100)
) ENGINE= InnoDB;

CREATE TABLE TipoItemTeste (
ID Int(10) AUTO_INCREMENT PRIMARY KEY,
Descricao VARCHAR(100)
) ENGINE= InnoDB;

CREATE TABLE Requisito (
Escopo VARCHAR(1000),
ID VARCHAR(100) PRIMARY KEY,
ID_TIPOITEMTESTE Int(10),
FOREIGN KEY(ID_TIPOITEMTESTE) REFERENCES TipoItemTeste (ID)
) ENGINE= InnoDB;

CREATE TABLE ExecucaoTeste (
ID Int(10) AUTO_INCREMENT PRIMARY KEY,
Descricao VARCHAR(100)
) ENGINE= InnoDB;

CREATE TABLE FaseTeste (
ID Int(10) AUTO_INCREMENT PRIMARY KEY,
Descricao VARCHAR(100)
) ENGINE= InnoDB;

CREATE TABLE Requisito_CasoTeste (
ID_REQUISITO VARCHAR(100),
ID_CASOTESTE Int(10),
```

```
PRIMARY KEY(ID_REQUISITO,ID_CASOTESTE),
FOREIGN KEY(ID_REQUISITO) REFERENCES Requisito (ID),
FOREIGN KEY(ID_CASOTESTE) REFERENCES CasoTeste (ID)
) ENGINE= InnoDB;

CREATE TABLE CasoTeste_SuiteTeste (
ID_SUITETESTE Int(10),
ID_CASOTESTE Int(10),
PRIMARY KEY(ID_CASOTESTE,ID_SUITETESTE),
FOREIGN KEY(ID_SUITETESTE) REFERENCES SuiteTeste (ID),
FOREIGN KEY(ID_CASOTESTE) REFERENCES CasoTeste (ID)
) ENGINE= InnoDB;

CREATE TABLE FerramentaTeste (
ID Int(10) AUTO_INCREMENT PRIMARY KEY,
Nome VARCHAR(100)
) ENGINE= InnoDB;

CREATE TABLE TecnicaTeste (
Nome VARCHAR(100),
ID Int(10) AUTO_INCREMENT PRIMARY KEY
) ENGINE= InnoDB;

CREATE TABLE CasoTeste_TecnicaTeste (
ID_TECNICATESTE Int(10),
ID_CASOTESTE Int(10),
PRIMARY KEY(ID_CASOTESTE,ID_TECNICATESTE),
FOREIGN KEY(ID_TECNICATESTE) REFERENCES TecnicaTeste (ID),
FOREIGN KEY(ID_CASOTESTE) REFERENCES CasoTeste (ID)
) ENGINE= InnoDB;
```

```
CREATE TABLE Plataforma (
ID Int(10) AUTO_INCREMENT PRIMARY KEY,
Descricao VARCHAR(100),
Nome VARCHAR(100),
ID_PROJETOTESTE Int(10),
FOREIGN KEY(ID_PROJETOTESTE) REFERENCES ProjetoTeste (ID)
) ENGINE= InnoDB;

CREATE TABLE PlanoTeste (
ID Int(10) AUTO_INCREMENT PRIMARY KEY,
Descricao VARCHAR(1000),
Nome VARCHAR(100),
ID_PROJETOTESTE Int(10),
FOREIGN KEY(ID_PROJETOTESTE) REFERENCES ProjetoTeste (ID)
) ENGINE= InnoDB;

CREATE TABLE BaseLine (
ID Int(10) AUTO_INCREMENT PRIMARY KEY,
Descricao VARCHAR(100),
Titulo VARCHAR(100),
ID_PLANOTESTE Int(10),
FOREIGN KEY(ID_PLANOTESTE) REFERENCES PlanoTeste (ID)
) ENGINE= InnoDB;

CREATE TABLE CasoTeste_PalavraChave (
ID_CASOTESTE Int(10),
ID_PALAVRACHAVE Int(10),
PRIMARY KEY(ID_CASOTESTE,ID_PALAVRACHAVE),
FOREIGN KEY(ID_CASOTESTE) REFERENCES CasoTeste (ID)
) ENGINE= InnoDB;
```

```
CREATE TABLE PalavraChave (
ID Int(10) AUTO_INCREMENT PRIMARY KEY,
Descricao VARCHAR(100),
Nome VARCHAR(100)
) ENGINE= InnoDB;

CREATE TABLE Resolucao (
ID Int(10) AUTO_INCREMENT PRIMARY KEY,
Descricao VARCHAR(100)
) ENGINE= InnoDB;

CREATE TABLE Gravidade (
ID Int(10) AUTO_INCREMENT PRIMARY KEY,
Descricao VARCHAR(100)
) ENGINE= InnoDB;

CREATE TABLE Frequencia (
ID Int(10) AUTO_INCREMENT PRIMARY KEY,
Descricao VARCHAR(100)
) ENGINE= InnoDB;

CREATE TABLE ClassificacaoDefeito (
ID Int(10) AUTO_INCREMENT PRIMARY KEY,
Descricao VARCHAR(100)
) ENGINE= InnoDB;

CREATE TABLE DEFEITO_CAUSADEFEITO (
ID_CAUSADEFEITO Int(10),
ID_DEFEITO Int(10),
PRIMARY KEY(ID_CAUSADEFEITO,ID_DEFEITO)
) ENGINE= InnoDB;
```

```
CREATE TABLE CausaDefeito (
ID Int(10) AUTO_INCREMENT PRIMARY KEY,
Descricao VARCHAR(100)
) ENGINE= InnoDB;

CREATE TABLE Defeito (
ID Int(10) PRIMARY KEY,
PassoReproducao VARCHAR(1000),
Resumo VARCHAR(1000),
ID_GRAVIDADE Int(10),
ID_ESTADO Int(10),
ID_FREQUENCIA Int(10),
ID_CLASSIFICACAODEFEITO Int(10),
ID_RESOLUCAO Int(10),
ID_RELATOR Int(10),
ID_DESENVOLVEDOR Int(10),
ID_ARTEFATO Int(10),
ID_PROJETOTESTE Int(10),
FOREIGN KEY(ID_GRAVIDADE) REFERENCES Gravidade (ID),
FOREIGN KEY(ID_ESTADO) REFERENCES Estado (ID),
FOREIGN KEY(ID_FREQUENCIA) REFERENCES Frequencia (ID),
FOREIGN KEY(ID_CLASSIFICACAODEFEITO) REFERENCES
ClassificacaoDefeito (ID),
FOREIGN KEY(ID_RESOLUCAO) REFERENCES Resolucao (ID),
FOREIGN KEY(ID_PROJETOTESTE) REFERENCES ProjetoTeste (ID)
) ENGINE= InnoDB;

CREATE TABLE TipoTeste (
ID Int(10) AUTO_INCREMENT PRIMARY KEY,
Descricao VARCHAR(100)
) ENGINE= InnoDB;
```

```
CREATE TABLE Funcionario (
ID Int(10) AUTO_INCREMENT PRIMARY KEY,
Nome VARCHAR(100),
Usuario VARCHAR(100)
) ENGINE= InnoDB;

CREATE TABLE Artefato (
ID Int(10) AUTO_INCREMENT PRIMARY KEY,
Descricao VARCHAR(100)
) ENGINE= InnoDB;

CREATE TABLE ResultadoTeste (
ID Int(10) PRIMARY KEY,
Nota VARCHAR(1000),
Status CHAR(1),
Data Datetime,
TempoGasto Int(10),
ID_CASOTESTE Int(10),
ID_TESTADOR Int(10),
ID_BASELINE Int(10),
ID_PLATAFORMA Int(10),
FOREIGN KEY(ID_CASOTESTE) REFERENCES CasoTeste (ID),
FOREIGN KEY(ID_TESTADOR) REFERENCES Funcionario (ID),
FOREIGN KEY(ID_BASELINE) REFERENCES BaseLine (ID),
FOREIGN KEY(ID_PLATAFORMA) REFERENCES Plataforma (ID)
) ENGINE= InnoDB;

CREATE TABLE ResultadoTeste_Defeito (
ID_DEFEITO Int(10),
ID_RESULTADOTESTE Int(10),
PRIMARY KEY(ID_DEFEITO,ID_RESULTADOTESTE),
```

```
FOREIGN KEY(ID_DEFEITO) REFERENCES Defeito (ID),
FOREIGN KEY(ID_RESULTADOTESTE) REFERENCES ResultadoTeste (ID)
) ENGINE= InnoDB;

ALTER TABLE SuiteTeste ADD FOREIGN KEY(ID_PROJETOTESTE)
REFERENCES ProjetoTeste (ID);
ALTER TABLE CasoTeste ADD FOREIGN KEY(ID_EXECUCAOTESTE)
REFERENCES ExecucaoTeste (ID);
ALTER TABLE CasoTeste ADD FOREIGN KEY(ID_TIPOTESTE)
REFERENCES TipoTeste (ID);
ALTER TABLE CasoTeste ADD FOREIGN KEY(ID_FERRAMENTATESTE)
REFERENCES FerramentaTeste (ID);
ALTER TABLE CasoTeste ADD FOREIGN KEY(ID_FASETESTE)
REFERENCES FaseTeste (ID);
ALTER TABLE CasoTeste_PalavraChave ADD FOREIGN
KEY(ID_PALAVRACHAVE) REFERENCES PalavraChave (ID);
ALTER TABLE DEFEITO_CAUSADEFEITO ADD FOREIGN
KEY(ID_CAUSADEFEITO) REFERENCES CausaDefeito (ID);
ALTER TABLE DEFEITO_CAUSADEFEITO ADD FOREIGN
KEY(ID_DEFEITO) REFERENCES Defeito (ID);
ALTER TABLE Defeito ADD FOREIGN KEY(ID_RELATOR) REFERENCES
Funcionario (ID);
ALTER TABLE Defeito ADD FOREIGN KEY(ID_DESENVOLVEDOR)
REFERENCES Funcionario (ID);
ALTER TABLE Defeito ADD FOREIGN KEY(ID_ARTEFATO) REFERENCES
Artefato (ID);
```

APÊNDICE E – Interfaces do módulo MCD

O módulo MCD tem a finalidade de consultar as informações armazenadas na base de conhecimento VVTeste e apresentá-los em formato de gráficos. Existem duas formas de gerar os gráficos: Através de consultar pré definidas ou, para usuário mais avançados, a consulta personalizada.

Na aba Pré definidos, conforme Figura E.1, a ferramenta disponibiliza 26 consultas, separadas pelo tipo de gráficos, que pode ser por Pizza ou por Barra.

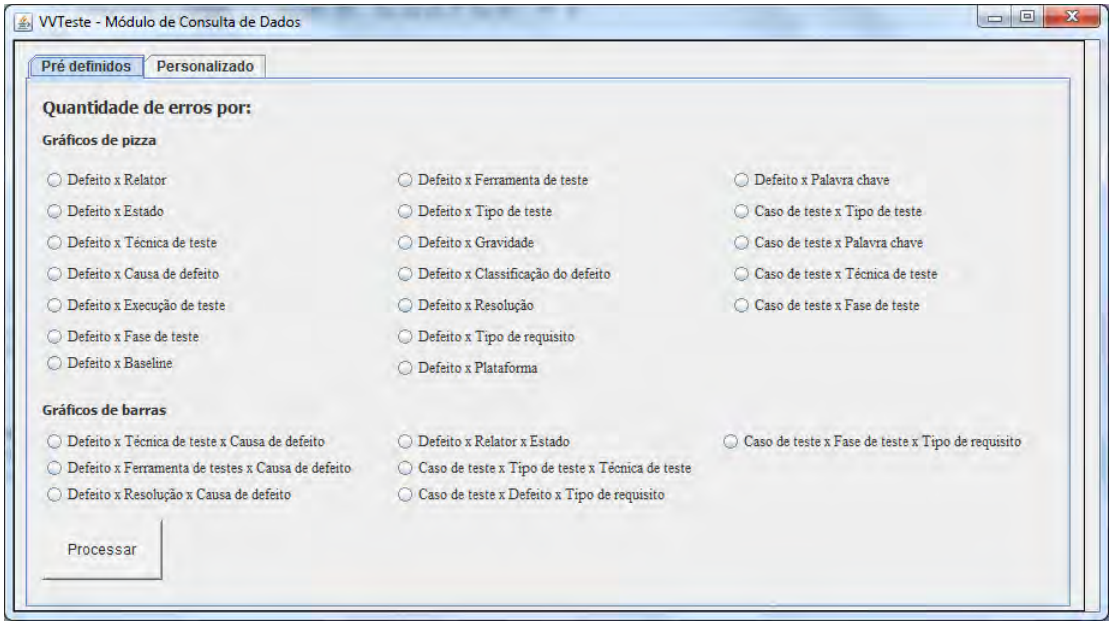

Figura E.1 - MCD - Aba Pré definidos

Assim que o usuário faz a escolha da consulta deseja, basta clicar no botão Processar, que o sistema irá abrir uma nova tela exibindo o gráfico desejado. O usuário pode manter mais de um gráfico aberto simultaneamente. A Figura E.2 apresenta um exemplo do gráfico do tipo pizza, onde é feita uma análise dos defeitos pelos seus respectivos estados.

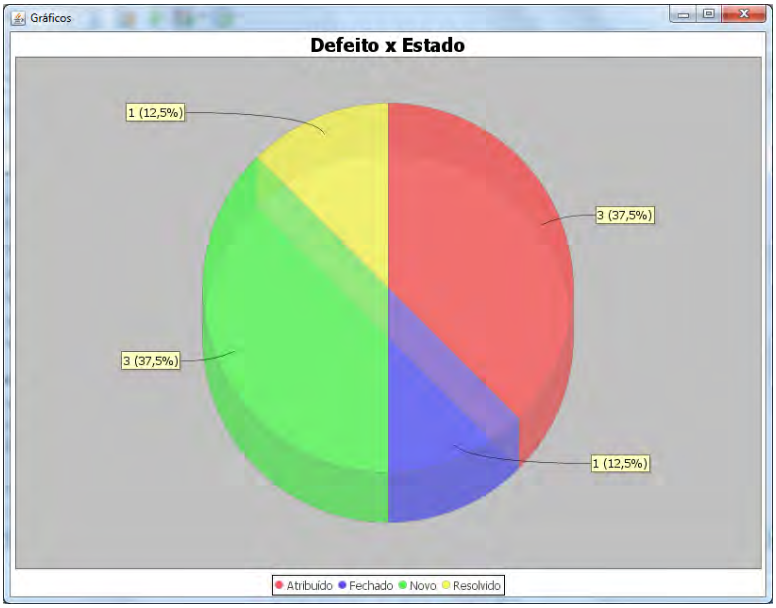

Figura E.2 - MCD - Exemplo de gráfico gerado

Na aba personalizado, Figura E.3, o usuário poderá entrar com um comando SQL de sua escolha, permitindo então que qualquer análise seja feita na base de conhecimento.

O primeiro passo é a escolha do tipo de gráfico. Caso a opção Pizza seja a escolhida, o usuário irá preencher duas das três linhas dos campos da seção SELECT. No caso do gráfico por barras, o usuário irá fazer uma análise com três variáveis, sendo necessário o preenchimento das três linhas da seção SELECT.

Na seção FROM, o usuário irá definir as tabelas e os relacionamentos para a realização da consulta. Ao clicar no botão Processar, o sistema irá validar o comando SQL digitado e caso ele seja válido, irá processar e criar o gráfico.

O título do gráfico será composto pelos apelidos dados aos campos da seção SELECT, ou seja, os valores definidos após a descrição “as”.

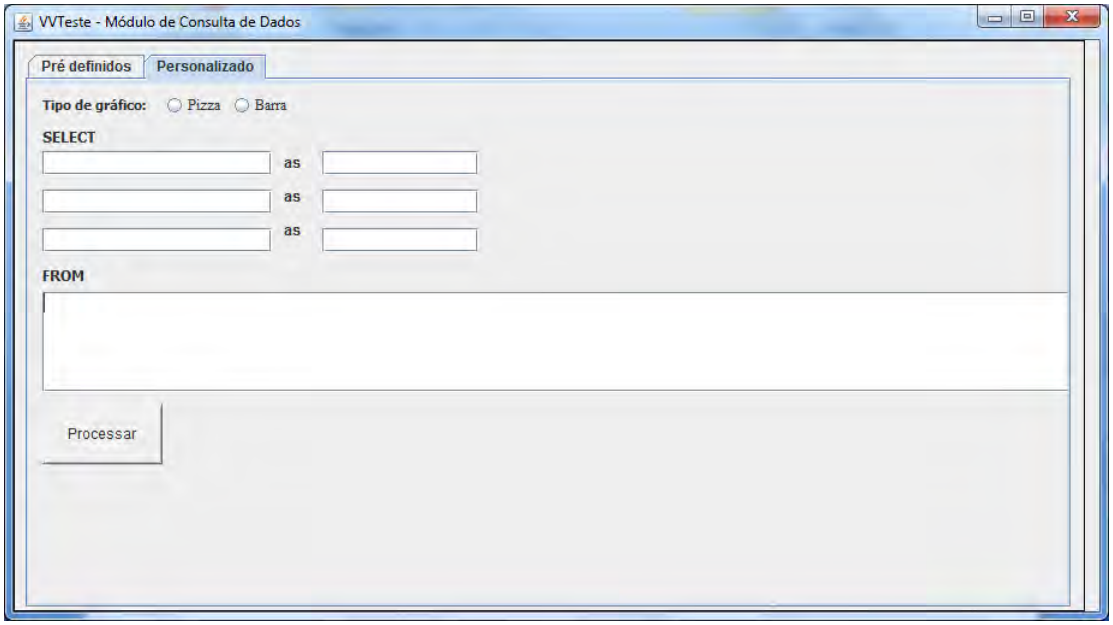

Figura E.3 - MCD - Aba Personalizado

APÊNDICE F - Estudo de caso – Máquina de estados

Este apêndice apresenta as sete MEFEs modeladas para a máquina de café utilizada no estudo de caso. A Figura F.1 apresenta o modelo criado para o modo de operação Normal.

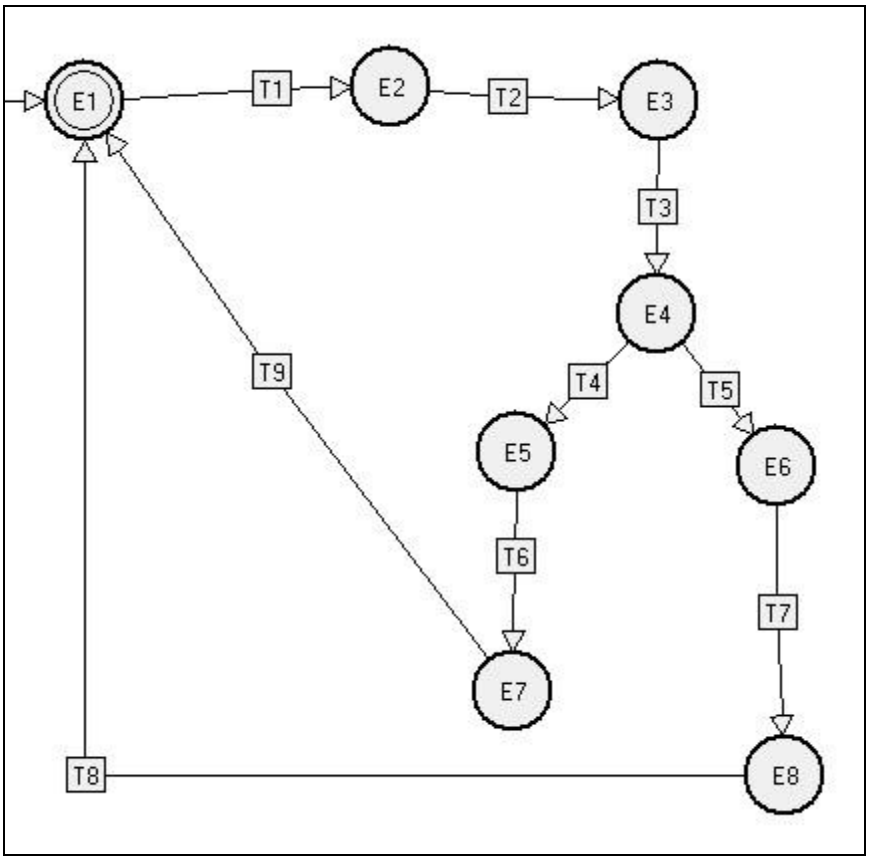

Figura F.1 - MEFE do modo de operação Normal

As figuras F.2 , F.3 e F.4 apresentam os modos de operação das exceções 5, 6 e 134.

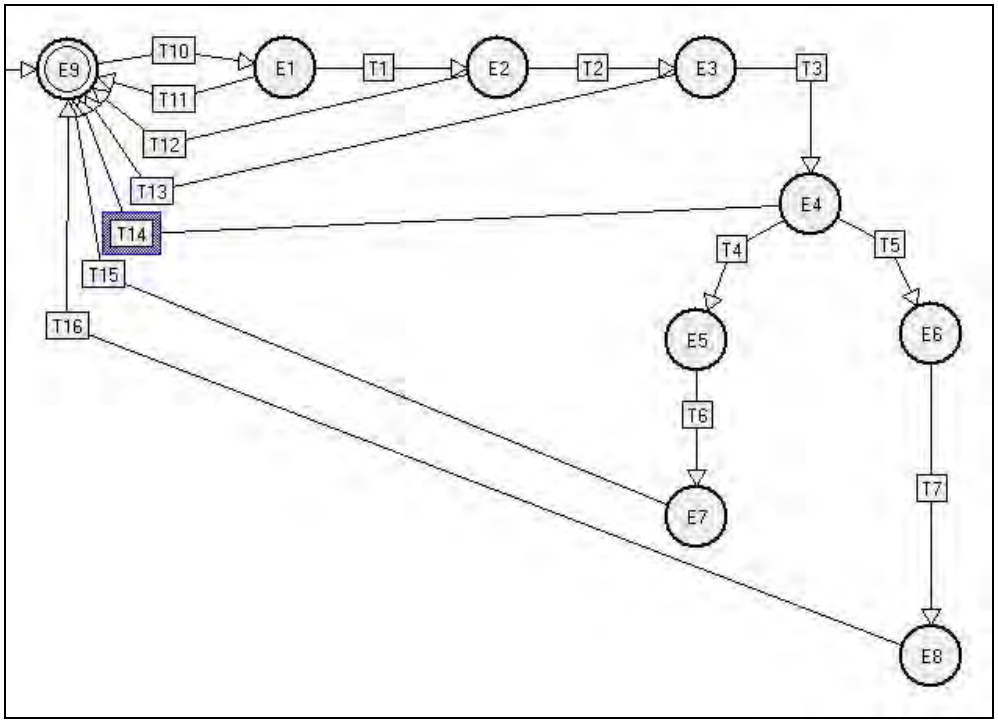

Figura F.2 - MEFE do modo de operação Exceção 5

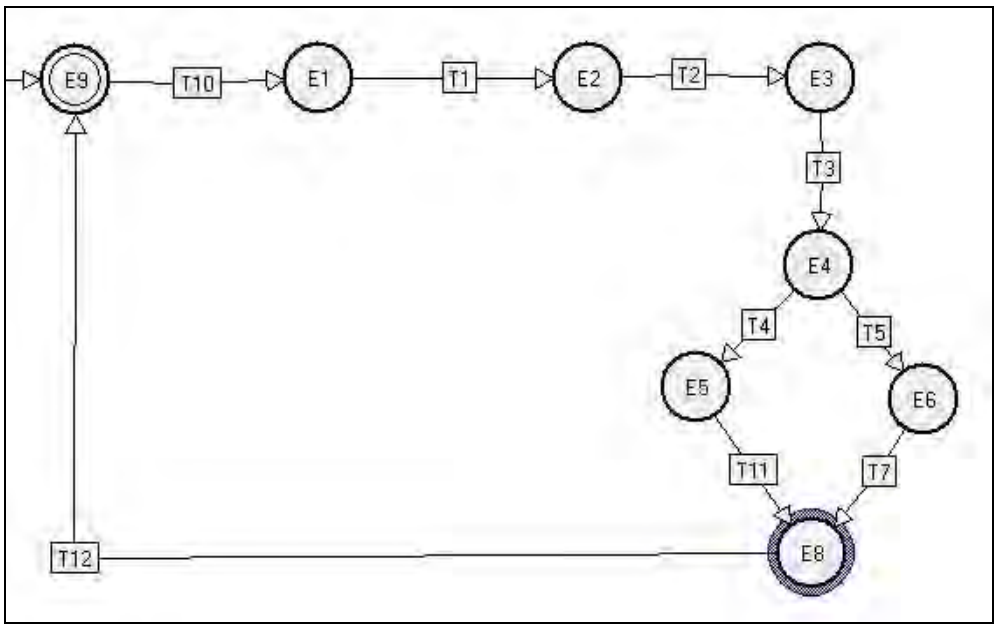

Figura F.3 - MEFE do modo de operação Exceção 6

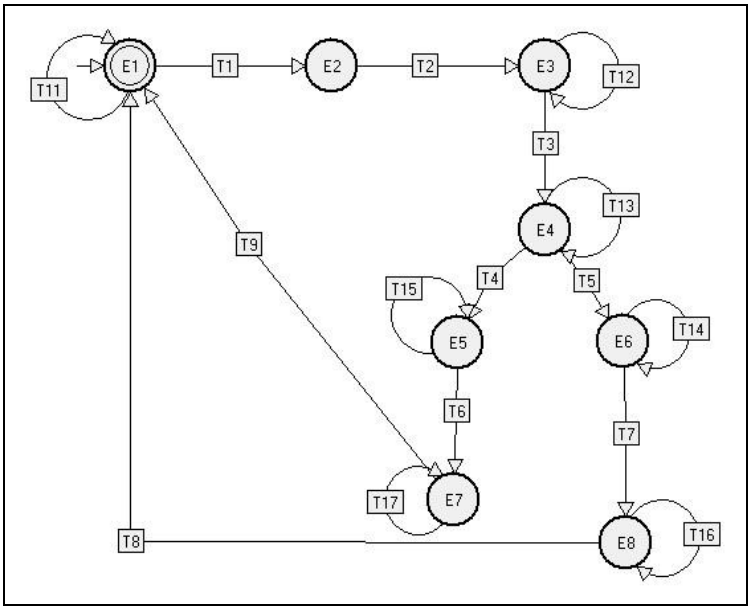

Figura F.4 - MEFE do modo de operação Exceção 134

Os modelos criados para os eventos normais ocorridos em momentos inesperados (caminhos furtivos) são exibidas nas Figuras F.5 e F.6.

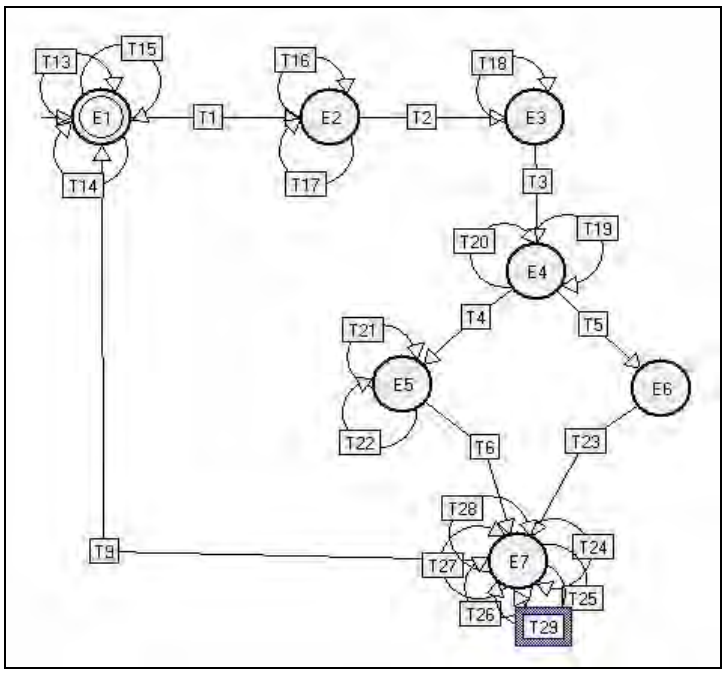

Figura F.5 - MEFE do modo de operação Caminho furtivo 1

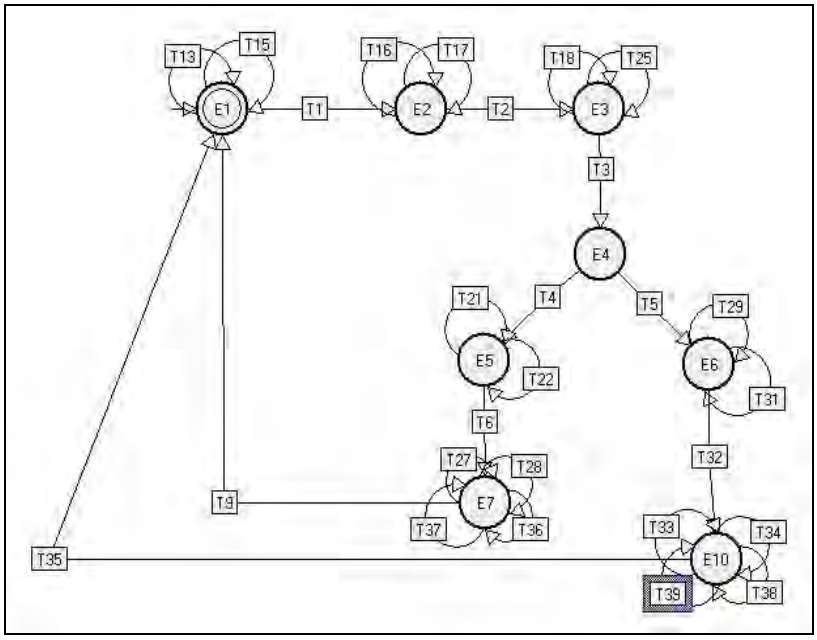

Figura F.6 - MEFE do modo de operação Caminho furtivo 2

A Figura F.7 exibe o modelo criado para representar o modo de operação de tolerância a falhas.

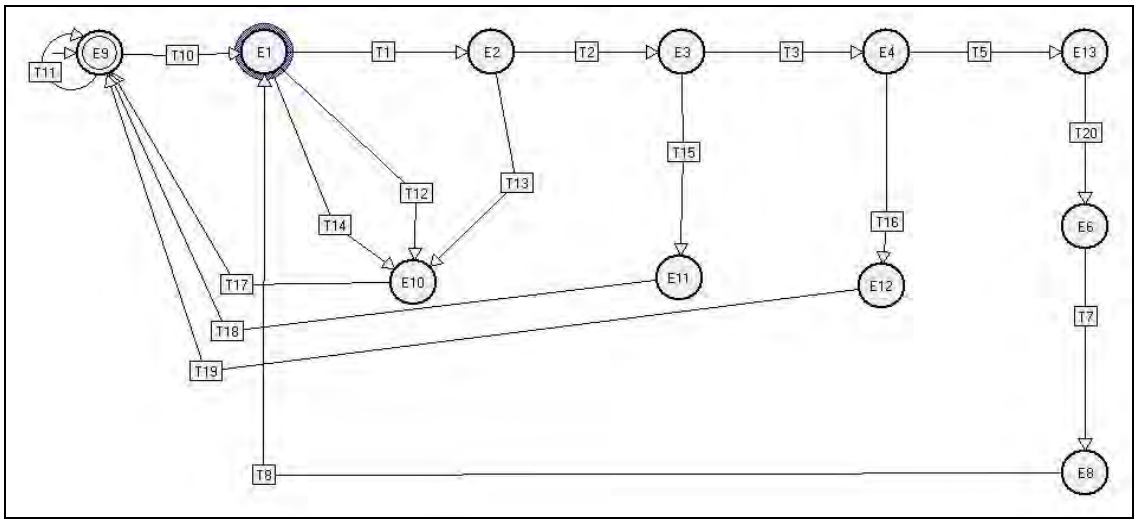

Figura F.7 - MEFE do modo de operação Tolerância a falhas

APÊNDICE G - Estudo de caso – Especificação dos testes

Estudo de caso

Especificar Testes

Projeto: Estudo de caso

Autor: Marcos Reis

Impresso por Testlink em 25/01/2012

Escopo

Este projeto de software visa realizar o estudo de caso do ambiente VVTeste, utilizando um exemplo didático de uma máquina de café.

Suite de Teste : Serviço 1 - Normal

Caso de Teste EC-40: S1-N-1	
Sumário:: Visa testar o serviço 1, correspondente a preparação de um copo de café, utilizando o modo de operação normal.	
Pré-condições: Não há	
Fase de teste:	Sistema
Tipo de teste:	Caixa Preta
Técnica de teste de caixa branca:	
Técnica de teste de caixa preta:	Baseado em modelos
Execucao de teste:	Manual
Ferramenta de teste:	
Tempo estimado:	50

#:	Ações do Passo:	Resultados Esperados::
1	input(L,ProdutosOk)	output(L,LiberaFicha)
2	input(L,InsereFicha)	output(L,LuzCreditoOn)
3	input(L,BtnCafe)	output(L,LuzCafeOn)
4	input(L,BtnPoucoAcucar)	output(L,LuzPoucoAcucarOn)
5		output(L,LiberaCopo)
6		output(L,LuzProcessamentoOn)
7		output(L,Dispara10s)
8	input(L,Fim10s)	output(L,LuzProntoOn)
9		output(L,DisponibilizaCopoCafePoucoAcucar)
10		output(L,LuzProcessamentoOff)
11	input(L,RetiraCopo)	output(L,LuzPoucoAcucarOff)
12		output(L,LuzCafeOff)
13		output(L,LuzCreditoOff)
14		output(L,LuzProntoOff)

Requisitos	R1: R1 R10: R10 R12: R12 R14: R14 R2: R2 R3: R3 R5: R5 R6: R6 R7: R7 R8: R8 R9: R9
Palavras-chave:	Serviço 1 Modo operação Normal Copo de café

Caso de Teste EC-41: S1-N-2

Sumário::

Visa testar o serviço 1, correspondente a preparação de um copo de café, utilizando o modo de operação normal.

Pré-condições:

Não há

Fase de teste:	Sistema
Tipo de teste:	Caixa Preta
Técnica de teste de caixa branca:	
Técnica de teste de caixa preta:	Baseado em modelos
Execucao de teste:	Manual
Ferramenta de teste:	
Tempo estimado:	60

#:	Ações do Passo:	Resultados Esperados::
1	input(L,ProdutosOk)	output(L,LiberaFicha)

2	input(L,InsereFicha)	output(L,LuzCreditoOn)
3	input(L,BtnCafe)	output(L,LuzCafeOn)
4	input(L,BtnMuitoAcucar)	output(L,LuzMuitoAcucarOn)
5		output(L,LiberaCopo)
6		output(L,LuzProcessamentoOn)
7		output(L,Dispara10s)
8	input(L,Fim10s)	output(L,LuzProntoOn)
9		output(L,DisponibilizaCopoCafeMuitoAcucar)
10		output(L,LuzProcessamentoOff)
11	input(L,RetiraCopo)	output(L,LuzMuitoAcucarOff)
12		output(L,LuzCafeOff)
13		output(L,LuzCreditoOff)
14		output(L,LuzProntoOff)
Requisitos	R1: R1 R10: R10 R12: R12 R14: R14 R2: R2 R3: R3 R5: R5 R6: R6 R7: R7 R8: R8 R9: R9	
Palavras-chave:	Serviço 1 Modo operação Normal Copo de café	

Suite de Teste : Serviço 1 - Exceção 5

Caso de Teste EC-42: S1-Ex5-1		
Sumário: Visa testar o serviço 1, correspondente a preparação de um copo de café, utilizando o modo de operação exceção 5.		
Pré-condições: Não há		
Fase de teste:	Unidade	
Tipo de teste:	Caixa Cinza	
Técnica de teste de caixa branca:		
Técnica de teste de caixa preta:	Transição de estado	
Execucao de teste:	Manual	
Ferramenta de teste:		
Tempo estimado:	90	
#:	Ações do Passo:	Resultados Esperados::

1	input(L,BtnLiga)	output(L,LuzOn)
2		output(L,ConfereProdutos)
3	input(L,ProdutosOk)	output(L,LiberaFicha)
4	input(L,InsereFicha)	output(L,LuzCreditoOn)
5	input(L,BtnCafe)	output(L,LuzCafeOn)
6	input(L,BtnPoucoAcucar)	output(L,LuzPoucoAcucarOn)
7		output(L,LiberaCopo)
8		output(L,LuzProcessamentoOn)
9		output(L,Dispara10s)
10	input(L,Fim10s)	output(L,LuzProntoOn)
11		output(L,DisponibilizaCopoCafePoucoAcucar)
12		output(L,LuzProcessamentoOff)
13	input(L,BtnDesliga)	output(L,LuzProntoOff)
14		output(L,LuzPoucoAcucarOff)
15		output(L,LuzCafeOff)
16		output(L,LuzCreditoOff)
17		output(L,LuzOff)

Requisitos	R11: R11 R12: R12 R2: R2
Palavras-chave:	Serviço 1 Copo de café Modo operação Exceção 5

Caso de Teste EC-43: S1-Ex5-2

Sumário::

Visa testar o serviço 1, correspondente a preparação de um copo de café, utilizando o modo de operação exceção 5.

Pré-condições:

Não há

Fase de teste:	Integração
Tipo de teste:	Caixa Cinza
Técnica de teste de caixa branca:	Fluxo de controle
Técnica de teste de caixa preta:	Baseado em modelos, Transição de estado
Execucao de teste:	Automática
Ferramenta de teste:	Ferramenta 2
Tempo estimado:	500

#:	Ações do Passo:	Resultados Esperados::
1	input(L,BtnLiga)	output(L,LuzOn)
2		output(L,ConfereProdutos)
3	input(L,ProdutosOk)	output(L,LiberaFicha)
4	input(L,InsereFicha)	output(L,LuzCreditoOn)

5	input(L,BtnCafe)	output(L,LuzCafeOn)
6	input(L,BtnMuitoAcucar)	output(L,LuzMuitoAcucarOn)
7		output(L,LiberaCopo)
8		output(L,LuzProcessamentoOn)
9		output(L,Dispara10s)
10	input(L,Fim10s)	output(L,LuzProntoOn)
11		output(L,DisponibilizaCopoCafeMuitoAcucar)
12		output(L,LuzProcessamentoOff)
13	input(L,BtnDesliga)	output(L,LuzProntoOff)
14		output(L,LuzMuitoAcucarOff)
15		output(L,LuzCafeOff)
16		output(L,LuzCreditoOff)
17		output(L,LuzOff)
Requisitos	R11: R11 R12: R12 R2: R2	
Palavras-chave:	Serviço 1 Copo de café Modo operação Exceção 5	

Caso de Teste EC-44: S1-Ex5-3

Sumário::

Visa testar o serviço 1, correspondente a preparação de um copo de café, utilizando o modo de operação exceção 5.

Pré-condições:

Não há

Fase de teste:	Sistema	
Tipo de teste:	Caixa Cinza	
Técnica de teste de caixa branca:	Fluxo de dados	
Técnica de teste de caixa preta:	Transição de estado	
Execucao de teste:	Automática	
Ferramenta de teste:	Ferramenta 3	
Tempo estimado:	150	
#:	Ações do Passo:	Resultados Esperados::
1	input(L,BtnLiga)	output(L,LuzOn)
2		output(L,ConfereProdutos)
3	input(L,ProdutosOk)	output(L,LiberaFicha)
4	input(L,InsereFicha)	output(L,LuzCreditoOn)
5	input(L,BtnCafe)	output(L,LuzCafeOn)
6	input(L,BtnDesliga)	output(L,LuzCafeOff)
7		output(L,LuzCreditoOff)
8		output(L,LuzOff)

Requisitos	R11: R11 R12: R12 R2: R2	
Palavras-chave:	Serviço 1 Copo de café Modo operação Exceção 5	
Caso de Teste EC-45: S1-Ex5-4		
Sumário::		
Visa testar o serviço 1, correspondente a preparação de um copo de café, utilizando o modo de operação exceção 5.		
Pré-condições:		
Não há		
Fase de teste:	Aceitação	
Tipo de teste:	Caixa Branca	
Técnica de teste de caixa branca:	Cobertura de comandos	
Técnica de teste de caixa preta:		
Execucao de teste:	Automática	
Ferramenta de teste:	Ferramenta 2	
Tempo estimado:	200	
#:	Ações do Passo:	Resultados Esperados::
1	input(L,BtnLiga)	output(L,LuzOn)
2		output(L,ConfereProdutos)
3	input(L,ProdutosOk)	output(L,LiberaFicha)
4	input(L,InsereFicha)	output(L,LuzCreditoOn)
5	input(L,BtnDesliga)	output(L,LuzOff)
6		output(L,LuzCreditoOff)
Requisitos	R11: R11 R12: R12 R2: R2	
Palavras-chave:	Serviço 1 Copo de café Modo operação Exceção 5	
Caso de Teste EC-46: S1-Ex5-5		
Sumário::		
Visa testar o serviço 1, correspondente a preparação de um copo de café, utilizando o modo de operação exceção 5.		
Pré-condições:		
Não há		
Fase de teste:	Sistema	
Tipo de teste:	Caixa Branca	

Técnica de teste de caixa branca:	Fluxo de controle, Fluxo de dados	
Técnica de teste de caixa preta:		
Execucao de teste:	Automática	
Ferramenta de teste:	Ferramenta 1	
Tempo estimado:	90	
#:	Ações do Passo:	Resultados Esperados::
1	input(L,BtnLiga)	output(L,LuzOn)
2		output(L,ConfereProdutos)
3	input(L,ProdutosOk)	output(L,LiberaFicha)
4	input(L,BtnDesliga)	output(L,LuzOff)
Requisitos	R11: R11 R12: R12 R2: R2	
Palavras-chave:	Serviço 1 Copo de café Modo operação Exceção 5	

Caso de Teste EC-47: S1-Ex5-6

Sumário::

Visa testar o serviço 1, correspondente a preparação de um copo de café, utilizando o modo de operação exceção 5.

Pré-condições:

Não há

Fase de teste:	Sistema	
Tipo de teste:	Caixa Cinza	
Técnica de teste de caixa branca:	Fluxo de dados	
Técnica de teste de caixa preta:	Baseado em modelos	
Execucao de teste:	Manual	
Ferramenta de teste:		
Tempo estimado:	50	
#:	Ações do Passo:	Resultados Esperados::
1	input(L,BtnLiga)	output(L,LuzOn)
2		output(L,ConfereProdutos)
3	input(L,BtnDesliga)	output(L,LuzOff)
Requisitos	R11: R11 R12: R12 R2: R2	
Palavras-chave:	Serviço 1 Copo de café Modo operação Exceção 5	

Suite de Teste : Serviço 1 - Exceção 6

Caso de Teste EC-48: S1-Ex6-1		
Sumário:: Visa testar o serviço 1, correspondente a preparação de um copo de café, utilizando o modo de operação exceção 6.		
Pré-condições: Não há		
Fase de teste:	Sistema	
Tipo de teste:	Caixa Cinza	
Técnica de teste de caixa branca:	Por complexidade	
Técnica de teste de caixa preta:	Funcional sistemático	
Execucao de teste:	Manual	
Ferramenta de teste:		
Tempo estimado:	75	
#:	Ações do Passo:	Resultados Esperados::
1	input(L,BtnLiga)	output(L,LuzOn)
2		output(L,ConfereProdutos)
3	input(L,ProdutosOk)	output(L,LiberaFicha)
4	input(L,InsereFicha)	output(L,LuzCreditoOn)
5	input(L,BtnCafe)	output(L,LuzCafeOn)
6	input(L,BtnPoucoAcucar)	output(L,LuzPoucoAcucarOn)
7		output(L,LiberaCopo)
8		output(L,LuzProcessamentoOn)
9		output(L,Dispara10s)
10	input(L,BtnDesliga)	output( )
11	input(L,Fim10s)	output(L,LuzProcessamentoOff)
12		output(L,LuzMuitoAcucarOff)
13		output(L,LuzCafeOff)
14		output(L,LuzCreditoOff)
15		output(L,LuzOff)
Requisitos	R12: R12 R13: R13	
Palavras-chave:	Serviço 1 Copo de café Modo operação Exceção 6	
Caso de Teste EC-49: S1-Ex6-2		
Sumário:: Visa testar o serviço 1, correspondente a preparação de um copo de café, utilizando o modo de operação exceção 6.		

Pré-condições: Não há		
Fase de teste:	Unidade	
Tipo de teste:	Caixa Preta	
Técnica de teste de caixa branca:		
Técnica de teste de caixa preta:	Transição de estado	
Execucao de teste:	Manual	
Ferramenta de teste:		
Tempo estimado:	65	
#:	Ações do Passo:	Resultados Esperados::
1	input(L,BtnLiga)	output(L,LuzOn)
2		output(L,ConfereProdutos)
3	input(L,ProdutosOk)	output(L,LiberaFicha)
4	input(L,InsereFicha)	output(L,LuzCreditoOn)
5	input(L,BtnCafe)	output(L,LuzCafeOn)
6	input(L,BtnMuitoAcucar)	output(L,LuzMuitoAcucarOn)
7		output(L,LiberaCopo)
8		output(L,LuzProcessamentoOn)
9		output(L,Dispara10s)
10	input(L,BtnDesliga)	output( )
11	input(L,Fim10s)	output(L,LuzProcessamentoOff)
12		output(L,LuzMuitoAcucarOff)
13		output(L,LuzCafeOff)
14		output(L,LuzCreditoOff)
15		output(L,LuzOff)
Requisitos	R12: R12 R13: R13	
Palavras-chave:	Serviço 1 Copo de café Modo operação Exceção 6	

Suite de Teste : Serviço 1 - Exceção 134

Caso de Teste EC-50: S1-Ex134-1	
Sumário:: Visa testar o serviço 1, correspondente a preparação de um copo de café, utilizando o modo de operação exceção 134.	
Pré-condições: Não há	
Fase de teste:	Sistema
Tipo de teste:	Caixa Preta

Técnica de teste de caixa branca:		
Técnica de teste de caixa preta:	Baseado em modelos	
Execucao de teste:	Automática	
Ferramenta de teste:	Ferramenta 1	
Tempo estimado:	100	
#:	Ações do Passo:	Resultados Esperados::
1	input(L,ProdutosOk)	output(L,LiberaFicha)
2	input(L,InsereFicha)	output(L,LuzCreditoOn)
3	input(L,BtnCafe)	output(L,LuzCafeOn)
4	input(L,BtnPoucoAcucar)	output(L,LuzPoucoAcucarOn)
5		output(L,LiberaCopo)
6		output(L,LuzProcessamentoOn)
7		output(L,Dispara10s)
8	input(L,Fim10s)	output(L,LuzProntoOn)
9		output(L,DisponibilizaCopoCafePoucoAcucar)
10		output(L,LuzProcessamentoOff)
11	input(L,RetiraCopo)	output(L,LuzPoucoAcucarOff)
12		output(L,LuzCafeOff)
13		output(L,LuzCreditoOff)
14		output(L,LuzProntoOff)
Requisitos	R10: R10 R12: R12 R2: R2 R6: R6	
Palavras-chave:	Serviço 1 Copo de café Modo operação Exceção 134	

Caso de Teste EC-51: S1-Ex134-2

Sumário::

Visa testar o serviço 1, correspondente a preparação de um copo de café, utilizando o modo de operação exceção 134.

Pré-condições:

Não há

Fase de teste:	Unidade
Tipo de teste:	Caixa Branca
Técnica de teste de caixa branca:	Fluxo de dados
Técnica de teste de caixa preta:	
Execucao de teste:	Automática
Ferramenta de teste:	Ferramenta 2
Tempo estimado:	67

#:	Ações do Passo:	Resultados Esperados::
1	input(L,ProdutosOk)	output(L,LiberaFicha)
2	input(L,InsereFicha)	output(L,LuzCreditoOn)
3	input(L,BtnCafe)	output(L,LuzCafeOn)
4	input(L,BtnMuitoAcucar)	output(L,LuzMuitoAcucarOn)
5		output(L,LiberaCopo)
6		output(L,LuzProcessamentoOn)
7		output(L,Dispara10s)
8	input(L,Fim10s)	output(L,LuzProntoOn)
9		output(L,DisponibilizaCopoCafeMuitoAcucar)
10		output(L,LuzProcessamentoOff)
11	input(L,RetiraCopo)	output(L,LuzMuitoAcucarOff)
12		output(L,LuzCafeOff)
13		output(L,LuzCreditoOff)
14		output(L,LuzProntoOff)

Requisitos	R10: R10 R12: R12 R2: R2 R6: R6
Palavras-chave:	Serviço 1 Copo de café Modo operação Exceção 134

Caso de Teste EC-52: S1-Ex134-3

Sumário::

Visa testar o serviço 1, correspondente a preparação de um copo de café, utilizando o modo de operação exceção 134.

Pré-condições:

Não há

Fase de teste:	Sistema
Tipo de teste:	Caixa Preta
Técnica de teste de caixa branca:	
Técnica de teste de caixa preta:	Funcional sistemático
Execucao de teste:	Manual
Ferramenta de teste:	
Tempo estimado:	80

#:	Ações do Passo:	Resultados Esperados::
1	input(L,ProdutosNOK)	output(L,ConfereProdutos)
2		output(L,NaoAceitaFicha)

Requisitos	R10: R10 R12: R12 R2: R2 R6: R6

Palavras-chave:	Serviço 1 Copo de café Modo operação Exceção 134

Suite de Teste : Serviço 1 - Caminho Furtivo 1

Caso de Teste EC-53: S1-CF1-1		
Sumário:: Visa testar o serviço 1, correspondente a preparação de um copo de café, utilizando o modo de operação caminhos furtivos 1.		
Pré-condições: Não há		
Fase de teste:	Sistema	
Tipo de teste:		
Técnica de teste de caixa branca:		
Técnica de teste de caixa preta:	Baseado em modelos	
Execucao de teste:	Manual	
Ferramenta de teste:		
Tempo estimado:	200	
#:	Ações do Passo:	Resultados Esperados::
1	input(L,ProdutosOk)	output(L,LiberaFicha)
2	input(L,InsereFicha)	output(L,LuzCreditoOn)
3	input(L,BtnCafe)	output(L,LuzCafeOn)
4	input(L,BtnPoucoAcucar)	output(L,LuzPoucoAcucarOn)
5		output(L,LiberaCopo)
6		output(L,LuzProcessamentoOn)
7		output(L,Dispara10s)
8	input(L,RetiraCopo)	output(L,Ignora)
9	input(L,Fim10s)	output(L,LuzPoucoAcucarOff)
10		output(L,LuzCafeOff)
11		output(L,LuzCreditoOff)
12		output(L,LuzProntoOff)
Requisitos	R12: R12 R2: R2 R4: R4 R5: R5	
Palavras-chave:	Serviço 1 Copo de café Modo operação Caminhos Furtivo 1	
Caso de Teste EC-54: S1-CF1-2		
Sumário::		

Visa testar o serviço 1, correspondente a preparação de um copo de café, utilizando o modo de operação caminhos furtivos 1.		
Pré-condições: Não há		
Fase de teste:	Integração	
Tipo de teste:	Caixa Cinza	
Técnica de teste de caixa branca:	Cobertura de decisão	
Técnica de teste de caixa preta:	Baseado em modelos	
Execucao de teste:	Automática	
Ferramenta de teste:	Ferramenta 3	
Tempo estimado:	50	
#:	Ações do Passo:	Resultados Esperados::
1	input(L,ProdutosOk)	output(L,LiberaFicha)
2	input(L,InsereFicha)	output(L,LuzCreditoOn)
3	input(L,BtnCafe)	output(L,LuzCafeOn)
4	input(L,BtnMuitoAcucar)	output(L,LuzMuitoAcucarOn)
5		output(L,LiberaCopo)
6		output(L,LuzProcessamentoOn)
7		output(L,Dispara10s)
8	input(L,RetiraCopo)	output(L,Ignora)
9	input(L,Fim10s)	output(L,LuzPoucoAcucarOff)
10		output(L,LuzCafeOff)
11		output(L,LuzCreditoOff)
12		output(L,LuzProntoOff)
Requisitos	R12: R12 R2: R2 R4: R4 R5: R5	
Palavras-chave:	Serviço 1 Copo de café Modo operação Caminhos Furtivo 1	
Caso de Teste EC-55: S1-CF1-3		
Sumário:: Visa testar o serviço 1, correspondente a preparação de um copo de café, utilizando o modo de operação caminhos furtivos 1.		
Pré-condições: Não há		
Fase de teste:	Unidade	
Tipo de teste:	Caixa Branca	
Técnica de teste de caixa branca:	Fluxo de dados	

Técnica de teste de caixa preta:		
Execucao de teste:	Automática	
Ferramenta de teste:	Ferramenta 1	
Tempo estimado:	400	
#:	Ações do Passo:	Resultados Esperados::
1	input(L,BtnCafe)	output(L,Ignora)
Requisitos	R12: R12 R2: R2 R4: R4 R5: R5	
Palavras-chave:	Serviço 1 Copo de café Modo operação Caminhos Furtivo 1	

Caso de Teste EC-56: S1-CF1-4

Sumário::

Visa testar o serviço 1, correspondente a preparação de um copo de café, utilizando o modo de operação caminhos furtivos 1.

Pré-condições:

Não há

Fase de teste:	Integração	
Tipo de teste:	Caixa Preta	
Técnica de teste de caixa branca:		
Técnica de teste de caixa preta:	Partição de equivalência	
Execucao de teste:	Manual	
Ferramenta de teste:		
Tempo estimado:	90	
#:	Ações do Passo:	Resultados Esperados::
1	input(L,InsereFicha)	output(L,NaoAceitaFicha)
Requisitos	R12: R12 R2: R2 R4: R4 R5: R5	
Palavras-chave:	Serviço 1 Copo de café Modo operação Caminhos Furtivo 1	

Caso de Teste EC-57: S1-CF1-5

Sumário::

Visa testar o serviço 1, correspondente a preparação de um copo de café, utilizando o modo de operação caminhos furtivos 1.

Pré-condições:

Não há		
Fase de teste:	Unidade	
Tipo de teste:	Caixa Preta	
Técnica de teste de caixa branca:		
Técnica de teste de caixa preta:	Transição de estado	
Execucao de teste:	Manual	
Ferramenta de teste:		
Tempo estimado:	75	
#:	Ações do Passo:	Resultados Esperados::
1	input(L,BtnLiga)	output(L,Ignora)
Requisitos	R12: R12 R2: R2 R4: R4 R5: R5	
Palavras-chave:	Serviço 1 Copo de café Modo operação Caminhos Furtivo 1	

Suite de Teste : Serviço 1 - Caminho Furtivo 2

Caso de Teste EC-58: S1-CF2-1		
Sumário:: Visa testar o serviço 1, correspondente a preparação de um copo de café, utilizando o modo de operação caminhos furtivos 2.		
Pré-condições: Não há		
Fase de teste:	Unidade	
Tipo de teste:	Caixa Branca	
Técnica de teste de caixa branca:	Fluxo de dados	
Técnica de teste de caixa preta:		
Execucao de teste:	Automática	
Ferramenta de teste:		
Tempo estimado:	80	
#:	Ações do Passo:	Resultados Esperados::
1	input(L,ProdutosOk)	output(L,LiberaFicha)
2	input(L,InsereFicha)	output(L,LuzCreditoOn)
3	input(L,BtnCafe)	output(L,LuzCafeOn)
4	input(L,BtnPoucoAcucar)	output(L,LuzPoucoAcucarOn)
5		output(L,Dispara10s)
6		output(L,LiberaCopo)

7		output(L,LuzProcessamentoOn)
8	input(L,Fim10s)	output(L,DisponibilizaCopoCafePoucoAcucar)
9		output(L,LuzProntoOn)
10		output(L,LuzProcessamentoOff)
11	input(L,RetiraCopo)	output(L,LuzPoucoAcucarOff)
12		output(L,LuzCafeOff)
13		output(L,LuzCreditoOff)
14		output(L,LuzProntoOff)
Requisitos	R12: R12 R2: R2 R4: R4 R5: R5	
Palavras-chave:	Serviço 1 Copo de café Modo operação Caminhos Furtivo 2	
Caso de Teste EC-59: S1-CF2-2		
Sumário:: Visa testar o serviço 1, correspondente a preparação de um copo de café, utilizando o modo de operação caminhos furtivos 2.		
Pré-condições: Não há		
Fase de teste:	Integração	
Tipo de teste:	Caixa Preta	
Técnica de teste de caixa branca:		
Técnica de teste de caixa preta:	Grafo Causa-Efeito	
Execucao de teste:	Automática	
Ferramenta de teste:	Ferramenta 1	
Tempo estimado:	230	
#:	Ações do Passo:	Resultados Esperados::
1	input(L,ProdutosOk)	output(L,LiberaFicha)
2	input(L,InsereFicha)	output(L,LuzCreditoOn)
3	input(L,BtnCafe)	output(L,LuzCafeOn)
4	input(L,BtnMuitoAcucar)	output(L,LuzMuitoAcucarOn)
5		output(L,Dispara10s)
6		output(L,LiberaCopo)
7		output(L,LuzProcessamentoOn)
8	input(L,Fim10s)	output(L,DisponibilizaCopoCafeMuitoAcucar)
9		output(L,LuzProntoOn)
10		output(L,LuzProcessamentoOff)
11	input(L,RetiraCopo)	output(L,LuzProntoOff)
12		output(L,LuzMuitoAcucarOff)

13		output(L,LuzCafeOff)
14		output(L,LuzCreditoOff)
Requisitos	R12: R12 R2: R2 R4: R4 R5: R5	
Palavras-chave:	Serviço 1 Copo de café Modo operação Caminhos Furtivo 2	

Caso de Teste EC-60: S1-CF2-3

Sumário::

Visa testar o serviço 1, correspondente a preparação de um copo de café, utilizando o modo de operação caminhos furtivos 2.

Pré-condições:

Não há

Fase de teste:	Integração	
Tipo de teste:	Caixa Preta	
Técnica de teste de caixa branca:		
Técnica de teste de caixa preta:	Tabela de decisão	
Execucao de teste:	Manual	
Ferramenta de teste:		
Tempo estimado:	20	
#:	Ações do Passo:	Resultados Esperados::
1	input(L,BtnPoucoAcucar)	output(L,Ignora)
Requisitos	R12: R12 R2: R2 R4: R4 R5: R5	
Palavras-chave:	Serviço 1 Copo de café Modo operação Caminhos Furtivo 2	

Caso de Teste EC-61: S1-CF2-4

Sumário::

Visa testar o serviço 1, correspondente a preparação de um copo de café, utilizando o modo de operação caminhos furtivos 2.

Pré-condições:

Não há

Fase de teste:	Unidade
Tipo de teste:	Caixa Branca
Técnica de teste de caixa branca:	Por complexidade

Técnica de teste de caixa preta:		
Execucao de teste:	Automática	
Ferramenta de teste:	Ferramenta 2	
Tempo estimado:	50	
#:	Ações do Passo:	Resultados Esperados::
1	input(L,BtnMuitoAcucar)	output(L,Ignora)
Requisitos	R12: R12 R2: R2 R4: R4 R5: R5	
Palavras-chave:	Serviço 1 Copo de café Modo operação Caminhos Furtivo 2	

Suite de Teste : Serviço 1 - Falha

Caso de Teste EC-62: S1-Fa-1		
Sumário:: Visa testar o serviço 1, correspondente a preparação de um copo de café, utilizando o modo de operação Falha.		
Pré-condições: Não há		
Fase de teste:	Sistema	
Tipo de teste:	Caixa Preta	
Técnica de teste de caixa branca:		
Técnica de teste de caixa preta:	Baseado em modelos	
Execucao de teste:	Manual	
Ferramenta de teste:		
Tempo estimado:	100	
#:	Ações do Passo:	Resultados Esperados::
1	input(L,BtnLiga)	output(L,LuzOn)
2		output(L,ConfereProdutos)
3	input(L,ProdutosOk)	output(L,LiberaFicha)
4	input(L,InsereFicha)	output(L,LuzCreditoOn)
5	input(L,BtnCafe)	output(L,LuzCafeOn)
6	input(L,fACU)	output(L,NaoPossuiAcucar)
7	input(L,BtnMuitoAcucar)	output(L,LuzMuitoAcucarOn)
8		output(L,LiberaCopo)
9		output(L,LuzProcessamentoOn)
10		output(L,Dispara10s)
11	input(L,Fim10s)	output(L,LuzProntoOn)

12		output(L,DisponibilizaCopoSemAcucar)
13		output(L,LuzProcessamentoOff)
14	input(L,RetiraCopo)	output(L,LuzMuitoAcucarOff)
15		output(L,LuzCafeOff)
16		output(L,LuzCreditoOff)
17		output(L,LuzProntoOff)
18	input(L,fSCF)	output(L,ProdutosNOK)
19	input(L,BtnDesliga)	output(L,LuzOff)
Requisitos	R12: R12	
Palavras-chave:	Serviço 1 Copo de café Modo operação Falha	
Caso de Teste EC-63: S1-Fa-2		
Sumário:: Visa testar o serviço 1, correspondente a preparação de um copo de café, utilizando o modo de operação Falha.		
Pré-condições: Não há		
Fase de teste:	Sistema	
Tipo de teste:	Caixa Preta	
Técnica de teste de caixa branca:		
Técnica de teste de caixa preta:	Baseado em modelos	
Execucao de teste:	Manual	
Ferramenta de teste:	Ferramenta 1	
Tempo estimado:	70	
#:	Ações do Passo:	Resultados Esperados::
1	input(L,BtnLiga)	output(L,LuzOn)
2		output(L,ConfereProdutos)
3	input(L,ProdutosOk)	output(L,LiberaFicha)
4	input(L,InsereFicha)	output(L,LuzCreditoOn)
5	input(L,BtnCafe)	output(L,LuzCafeOn)
6	input(L,fACU)	output(L,NaoPossuiAcucar)
7	input(L,BtnMuitoAcucar)	output(L,LuzMuitoAcucarOn)
8		output(L,LiberaCopo)
9		output(L,LuzProcessamentoOn)
10		output(L,Dispara10s)
11	input(L,Fim10s)	output(L,LuzProntoOn)
12		output(L,DisponibilizaCopoSemAcucar)
13		output(L,LuzProcessamentoOff)
14	input(L,RetiraCopo)	output(L,LuzMuitoAcucarOff)
15		output(L,LuzCafeOff)

16		output(L,LuzCreditoOff)
17		output(L,LuzProntoOff)
18	input(L,fSCP)	output(L,ProdutosNOK)
19	input(L,BtnDesliga)	output(L,LuzOff)
Requisitos	R12: R12	
Palavras-chave:	Serviço 1 Copo de café Modo operação Falha	

Caso de Teste EC-64: S1-Fa-3

Sumário::

Visa testar o serviço 1, correspondente a preparação de um copo de café, utilizando o modo de operação Falha.

Pré-condições:

Não há

Fase de teste:	Sistema	
Tipo de teste:	Caixa Cinza	
Técnica de teste de caixa branca:	Cobertura de decisão	
Técnica de teste de caixa preta:	Funcional sistemático	
Execucao de teste:	Manual	
Ferramenta de teste:		
Tempo estimado:	240	
#:	Ações do Passo:	Resultados Esperados::
1	input(L,BtnLiga)	output(L,LuzOn)
2		output(L,ConfereProdutos)
3	input(L,ProdutosOk)	output(L,LiberaFicha)
4	input(L,InsereFicha)	output(L,LuzCreditoOn)
5	input(L,BtnCafe)	output(L,LuzCafeOn)
6	input(L,fBTN)	output(L,SemResposta)
7	input(L,BtnDesliga)	output(L,LuzCreditoOff)
8		output(L,LuzCafeOff)
9		output(L,LuzOff)
Requisitos	R12: R12	
Palavras-chave:	Serviço 1 Copo de café Modo operação Falha	

Caso de Teste EC-65: S1-Fa-4

Sumário::

Visa testar o serviço 1, correspondente a preparação de um copo de café, utilizando o modo de operação Falha.

Pré-condições:

Não há		
Fase de teste:	Unidade	
Tipo de teste:	Caixa Branca	
Técnica de teste de caixa branca:	Potenciais-Usos	
Técnica de teste de caixa preta:		
Execucao de teste:	Automática	
Ferramenta de teste:	Ferramenta 3	
Tempo estimado:	30	
#:	Ações do Passo:	Resultados Esperados::
1	input(L,BtnLiga)	output(L,LuzOn)
2		output(L,ConfereProdutos)
3	input(L,ProdutosOk)	output(L,LiberaFicha)
4	input(L,InsereFicha)	output(L,LuzCreditoOn)
5	input(L,fBTN)	output(L,SemResposta)
6	input(L,BtnDesliga)	output(L,LuzCreditoOff)
7		output(L,LuzOff)
Requisitos	R12: R12	
Palavras-chave:	Serviço 1 Copo de café Modo operação Falha	
Caso de Teste EC-66: S1-Fa-5		
Sumário:: Visa testar o serviço 1, correspondente a preparação de um copo de café, utilizando o modo de operação Falha.		
Pré-condições: Não há		
Fase de teste:	Unidade	
Tipo de teste:	Caixa Branca	
Técnica de teste de caixa branca:	Cobertura de decisão	
Técnica de teste de caixa preta:		
Execucao de teste:	Automática	
Ferramenta de teste:	Ferramenta 3	
Tempo estimado:	80	
#:	Ações do Passo:	Resultados Esperados::
1	input(L,BtnLiga)	output(L,LuzOn)
2		output(L,ConfereProdutos)
3	input(L,ProdutosOk)	output(L,LiberaFicha)
4	input(L,fFCH)	output(L,NaoAceitaFicha)
5	input(L,BtnDesliga)	output(L,LuzOff)

Requisitos	R12: R12	
Palavras-chave:	Serviço 1 Copo de café Modo operação Falha	
Caso de Teste EC-67: S1-Fa-6		
Sumário::		
Visa testar o serviço 1, correspondente a preparação de um copo de café, utilizando o modo de operação Falha.		
Pré-condições:		
Não há		
Fase de teste:	Aceitação	
Tipo de teste:	Caixa Preta	
Técnica de teste de caixa branca:		
Técnica de teste de caixa preta:	Funcional sistemático	
Execucao de teste:	Manual	
Ferramenta de teste:		
Tempo estimado:	80	
#:	Ações do Passo:	Resultados Esperados::
1	input(L,BtnLiga)	output(L,LuzOn)
2		output(L,ConfereProdutos)
3	input(L,fSCF)	output(L,ProdutosNOK)
4	input(L,BtnDesliga)	output(L,LuzOff)
Requisitos	R12: R12	
Palavras-chave:	Serviço 1 Copo de café Modo operação Falha	
Caso de Teste EC-68: S1-Fa-7		
Sumário::		
Visa testar o serviço 1, correspondente a preparação de um copo de café, utilizando o modo de operação Falha.		
Pré-condições:		
Não há		
Fase de teste:	Aceitação	
Tipo de teste:	Caixa Preta	
Técnica de teste de caixa branca:		
Técnica de teste de caixa preta:	Baseado em modelos	
Execucao de teste:	Manual	
Ferramenta de		

teste:		
Tempo estimado:	80	
#:	Ações do Passo:	Resultados Esperados::
1	input(L,BtnLiga)	output(L,LuzOn)
2		output(L,ConfereProdutos)
3	input(L,fSCP)	output(L,ProdutosNOK)
4	input(L,BtnDesliga)	output(L,LuzOff)
Requisitos	R12: R12	
Palavras-chave:	Serviço 1 Copo de café Modo operação Falha	

Caso de Teste EC-69: S1-Fa-8

Sumário::

Visa testar o serviço 1, correspondente a preparação de um copo de café, utilizando o modo de operação Falha.

Pré-condições:

Não há

Fase de teste:	Sistema	
Tipo de teste:	Caixa Preta	
Técnica de teste de caixa branca:		
Técnica de teste de caixa preta:	Funcional sistemático	
Execucao de teste:	Manual	
Ferramenta de teste:	Ferramenta 1	
Tempo estimado:	100	
#:	Ações do Passo:	Resultados Esperados::
1	input(L,fBTN)	output(L,SemResposta)
Requisitos	R12: R12	
Palavras-chave:	Serviço 1 Copo de café Modo operação Falha	