UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE SISTEMAS DE INFORMAÇÃO – BACHARELADO

FERRAMENTA DE APOIO AO PROCESSO DE GESTÃO DE DEFEITOS

CRISTIANO BENNERTZ

BLUMENAU
2011

2011/2-09

CRISTIANO BENNERTZ

FERRAMENTA DE APOIO AO PROCESSO DE GESTÃO DE DEFEITOS

Trabalho de Conclusão de Curso submetido à Universidade Regional de Blumenau para a obtenção dos créditos na disciplina Trabalho de Conclusão de Curso II do curso de Sistemas de Informação— Bacharelado.

Prof. Everaldo Artur Grahl, Mestre – Orientador

BLUMENAU
2011

2011/2-09

FERRAMENTA DE APOIO AO PROCESSO DE GESTÃO DE DEFEITOS

Por

CRISTIANO BENNERTZ

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

Presidente:	Prof. Everaldo Artur Grahl, Mestre – Orientador, FURB
Membro:	Prof. Ricardo de Alencar Azambuja, Mestre – FURB
Membro:	Prof. Wilson Pedro Carli, Mestre – FURB

Blumenau, 08 de dezembro de 2011.

Dedico este trabalho a minha esposa por sua paciência em relação a minha ausência durante a elaboração deste trabalho e aos amigos que me auxiliaram diretamente na realização deste.

AGRADECIMENTOS

A minha esposa, pelo incentivo e não permitir a minha desistência quando me encontrava diante de problemas que não conseguia resolver.

A Marlon Fernando Dirksen, Rafael Scolari Maciel e Jhony Alceu Pereira por me disponibilizar um pouco de seu tempo livre para tirar dúvidas em relação a problemas de implementação.

Ao meu orientador, Everaldo Artur Grahl, por ter me orientado e auxiliado na escolha dos caminhos a seguir durante a implementação e por ter acreditado na conclusão deste trabalho.

O descontentamento é o primeiro passo na evolução de um homem ou de uma nação.

Oscar Wilde

RESUMO

Este trabalho acadêmico demonstra a construção de uma ferramenta que auxilia no processo de gestão de defeitos. A ferramenta permite a identificação e registro de defeitos encontrados durante um ciclo de testes. Permite ainda que casos de teste e casos de uso sejam registrados possibilitando o agrupamento dos defeitos de acordo com o produto verificado durante os testes. A ferramenta atende alguns dos resultados previstos para o processo de verificação descrito no modelo de melhoria de processo de software brasileiro (MPS.BR). Dentre eles o qual determina que defeitos devem ser registrados e identificados, o que faz parte de um dos objetivos deste trabalho.

Palavras-chave: Gestão de defeitos. Ciclo de testes. Processo de verificação.

ABSTRACT

This scholarly work demonstrates the construction of a tool that assists in the management process defects. The tool allows the identification and recording of defects found during a testing cycle. It also allows test cases and use cases are recorded allowing the grouping of defects according to the product occurred during testing. The tool serves some of the anticipated outcomes of the verification process described in the model of software process improvement Brazil (MPS.BR). Among them which states that defects must be registered and identified, which is part of one of the objectives of this work.

Keywords: Management of defects. Testing cycle. Verification process.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE SIGLAS

API – *Application Programming Interface*

CA – Consultores de Aquisição

CASE – *Computer-Aided Software Engineering*

CMMI – *Capability Maturity Model Integration*

CSS – *Cascading Style Sheets*

ETM – Equipe Técnica do Modelo

FCC – Fórum de Credenciamento e Controle

IA – Instituições Avaliadoras

II – Instituições Implementadoras

IEEE – *Institute of Electrical and Electronic Engineers*

IOGE – Instituições Organizadoras de Grupos de Empresas

ISO – *International Organization for Standardization*

JS – *JavaScript*

JSP – *Java Server Pages*

MA-MPS – Método de Avaliação

MN-MPS – Modelo de Negócio

MR-MPS – Modelo de Referência

MPS.BR – Melhoria de Processos de Software Brasileiro

PDF – *Portable Document Format*

XHTML – *EXtensible HyperText Markup Language*

SUMÁRIO

1 INTRODUÇÃO

A atividade de teste de software vem recebendo uma maior atenção das empresas de desenvolvimento de software. Uma das razões é a maior exigência dos clientes dessas empresas, que já não aceitam receber softwares com problemas de usabilidade, visual desagradável e principalmente que apresentem erros em suas funcionalidades durante a sua utilização.

Um dos objetivos a alcançar com a realização dos testes de software é fazer com que o usuário goste do produto que ele está usando. Segundo Moreira Filho e Rios (2003 p. 147), gostar do produto compreende:

a) atender aos requisitos e especificações;
b) evitar ao máximo que ocorram falhas durante o uso do produto e quando ocorrerem, não causar grandes impactos nos resultados;
c) as interfaces com o usuário devem atender as expectativas de usabilidade.

Já existem alguns processos que estabelecem regras e propõem atividades a serem realizadas desde a fase de projeto, passando pela concepção, desenvolvimento, teste e outras. Tudo visando um produto final com maior qualidade e confiabilidade.

Apesar de toda exigência do mercado, ainda há empresas que desenvolvem software, sem processo de gestão de defeitos definido. Muitas vezes feita em planilhas eletrônicas que não oferecem segurança já que são facilmente alteradas. Outras vezes nem mesmo existe uma forma de gerir os defeitos.

A ferramenta desenvolvida e descrita neste trabalho possui o foco no apoio à gestão de defeitos e procura atender o processo de verificação do modelo de melhoria do processo de software brasileiro (MPS.BR[1]). O processo de verificação tem a finalidade de confirmar que serviços e/ou produtos de trabalho do processo ou do projeto atendem os requisitos especificados (SOFTEX, 2011c p. 42). Neste trabalho forma considerados apenas os resultados relacionados a gestão de defeitos. Para Delfim (2008), a gestão de defeitos permite acompanhar a qualidade de um software que está sendo testado com base nos defeitos que são registrados pelos testadores ao longo de um ciclo de testes. Sendo assim, é de grande importância que processos e ferramentas para gestão de defeitos sejam implementados nas

[1] A sigla MPS.BR está associada ao programa MPS.BR e significa, melhoria do processo de software brasileiro, e a sigla MPS está associada ao modelo MPS que significa, melhoria do processo de software (SOFTEX, 2011a p. 4).

empresas. Além de possibilitar a obtenção de uma base histórica, essas ferramentas impedem que defeitos encontrados não sejam corrigidos por esquecimento ou mesmo leviandade de um desenvolvedor menos preocupado com a qualidade do produto que será entregue ao cliente. Com a utilização de ferramentas com foco na gestão de defeitos os responsáveis por áreas de desenvolvimento de software têm a possibilidade de verificar periodicamente os defeitos ainda não corrigidos e cobrar uma solução a seus subordinados.

Segundo Bastos et al. (2007 p. 184), "a melhoria contínua só poderá ser alcançada quando o material coletado pela gestão de defeitos vier a ser usado pelos desenvolvedores e testadores com a intenção de otimizar os seus trabalhos".

1.1 OBJETIVOS DO TRABALHO

O objetivo deste trabalho foi desenvolver uma ferramenta que auxilie na gestão de defeitos encontrados durante as atividades de teste de software.

Os objetivos específicos do trabalho são:

a) compatibilizar as atividades propostas na ferramenta aos resultados previstos no processo de verificação do modelo de melhoria do processo de software brasileiro (MPS.BR);
b) permitir que os defeitos sejam acompanhados desde o momento de seu registro até a sua correção.

1.2 ESTRUTURA DO TRABALHO

Este trabalho está dividido em quatro capítulos.

No primeiro capítulo apresenta-se a introdução, os objetivos a serem alcançados na conclusão e a estrutura do trabalho.

No segundo capítulo tem-se a fundamentação teórica, onde é destacada a abordagem dos conceitos de testes de software, gestão de defeitos, MPS.BR e trabalhos correlatos.

No terceiro capítulo, está disposto o desenvolvimento da ferramenta, incluindo detalhes sobre a especificação, implementação realizada e operacionalidade das telas do

sistema.

No quarto capítulo apresenta-se a conclusão do trabalho bem como sugestões para trabalhos futuros como extensão deste.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo estão descritas algumas definições importantes acerca de teste de software, gestão de defeitos, MPS.BR e trabalhos correlatos.

2.1 TESTES DE SOFTWARE

O teste de software é um processo executado de forma controlada com o objetivo de avaliar o comportamento de um software levando em conta o que foi especificado (RIOS, 2008 p. 8).

Rios (2008 p. 9), afirma também que é praticamente impossível testar completamente um software e garantir que não existam defeitos nele. Para ele a qualidade de um software está ligada ao investimento realizado no processo de testes. Um software testado inadequadamente pode custar caro para a organização.

Para ressaltar ainda mais a importância dos testes de software, um processo de teste deve ser baseado em uma metodologia que é aderente ao processo de desenvolvimento realizado na empresa, e também em pessoal técnico devidamente capacitado, bem como ambiente operacional e ferramentas adequadas (RIOS, 2008 p. 10).

Conforme Hetzel e Flow Informática (1987 p.172), outro fator de sucesso nas atividades de testes, é a necessidade de gerar informações substanciais a respeito dos defeitos para que seja possível ter um bom andamento nos trabalhos.

2.2 GESTÃO DE DEFEITOS

Existe uma diferença entre o significado de erro e o significado de defeito. Para Bastos et al. (2007 p. 183), o erro é resultado de uma falha humana, já o defeito é o problema causado por um erro. O erro pode estar presente, por exemplo, em um código fonte ou em um documento. De uma forma genérica é possível afirmar que um defeito é o resultado de erros que estão presentes no software ou em qualquer tipo de artefato desenvolvido por pessoas.

Para Bastos et al. (2007 p. 183-184), o processo de gestão de defeitos segue alguns princípios, e estes estão sucintamente descritos a seguir:

a) seu principal objetivo é evitar o aparecimento de defeitos;
b) para evitar defeitos é necessário minimizar os riscos tanto dos projetos de desenvolvimento como os projetos de teste;
c) as ferramentas de gestão de defeitos devem ser utilizadas pelas equipes de qualidade e também pelas equipes de desenvolvimento para que haja um paralelismo entre o projeto de desenvolvimento e o projeto de teste;
d) a automação da gestão de defeitos é fator de sucesso;
e) a melhoria contínua será atingida quando os insumos coletados pela gestão de defeitos for utilizada pelos desenvolvedores e testadores para otimizar os trabalhos;
f) área de testes deve utilizar o *Capability Maturity Model Integration* (CMMI) para alcançar um nível de maturidade que permita o funcionamento adequado do processo.

2.2.1 Conceito de defeito

Os defeitos são em geral um desvio no que foi especificado nos requisitos. Bastos et al. (2007 p. 184) classifica os defeitos em duas formas. Há os defeitos ocorridos por falta de concordância com as especificações do produto e também os decorrentes de situações não previstas e que também não haviam sido definidas nas especificações do produto.

2.2.2 Tipos de defeitos

Tomando a classificação anterior como base é possível subdividir os defeitos em tipos. Entre eles estão os defeitos de interface com o usuário que compreendem:

a) defeitos de funcionalidade;
b) defeitos de usabilidade;
c) defeitos de desempenho;
d) defeitos de saída.

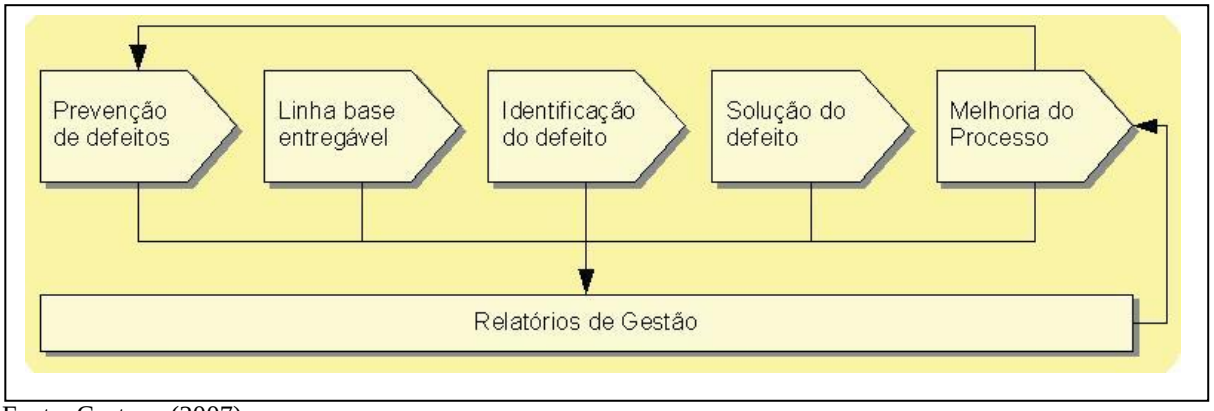
Outro tipo de defeitos são os introduzidos no tratamento de defeito. Nesse tipo enquadram-se os defeitos ocorridos pelo fato do programa não estar preparado para proteger-

se das entradas de dados imprevistas. Da mesma forma o programa também não é capaz de detectar nem de se recuperar de defeitos.

Um terceiro tipo de defeito é o relacionado com valores limite. Nesse tipo de defeito o programa não é capaz, por exemplo, de tratar adequadamente os valores máximo, mínimo, primeiro e último. Estes são apenas alguns dos tipos de erros que podem ser encontrados nas aplicações que estão a nossa disposição para utilização diariamente.

2.2.3 Processo de gestão de defeitos

Para Bastos et al. (2007 p.188), um processo de gestão de defeitos bem definido é composto por prevenção, linha de base, identificação, solução, melhoria e relatórios de gestão de defeitos. Caetano (2007), na Figura 1, ilustra os elementos básicos no processo de gestão de defeitos.



Fonte: Caetano (2007).

Figura 1 – Elementos chave de um processo de gestão de defeitos

Para prevenir a ocorrência de defeitos, as atividades de maior importância a serem realizadas são a identificação dos riscos críticos, a estimativa dos impactos esperados e a minimização destes.

Linha de base é um termo comumente utilizado para definir um produto que será entregue. O momento em que um produto é considerado como a linha de base, varia em cada organização. Bastos et al. (2007 p.192), afirma que em uma empresa em que o programador realiza tanto os testes unitários como a codificação, o produto pode ser considerado como uma linha de base após o término dos testes. Já se os testes são realizados por uma equipe independente, a organização pode considerar, por exemplo, que apenas o código fonte é uma

linha de base. No primeiro caso, o defeito será qualquer erro encontrado após os testes unitários e no segundo caso, o defeito será um erro encontrado durante os testes unitários. Para realizar a identificação de um defeito é necessário que haja o reconhecimento do defeito como válido por parte do desenvolvimento. Defeitos podem ser relatados por usuários, mas os desenvolvedores devem aceitá-los.

É importante que depois de aceitos, os defeitos sejam qualificados. Bastos et al. (2007 p. 196) sugere que os defeitos sejam qualificados em 3 níveis:

a) nível 1 – onde ocorre a definição do defeito;
b) nível 2 – onde descreve-se em que fase do ciclo de desenvolvimento de software o defeito ocorreu;
c) nível 3 – onde é informada a categoria do defeito.

Após a correção dos defeitos é hora de dedicar-se a melhoria do processo que é uma das atividades na qual as organizações menos se envolvem, mas que tem os melhores retornos (BASTOS et al., 2007 p. 199). Para melhoria do processo, avalia-se o processo que originou o defeito e o que o causou. Se for pertinente o processo pode ser alterado para minimizar ou eliminar a causa do defeito.

A atividade final do processo de gestão de defeitos é a elaboração dos relatórios de defeitos. Esses relatórios têm a finalidade de listar todos os desvios encontrados durante o processo de teste. Essas informações são utilizadas como base durante o projeto como um termômetro para as medições de qualidade.

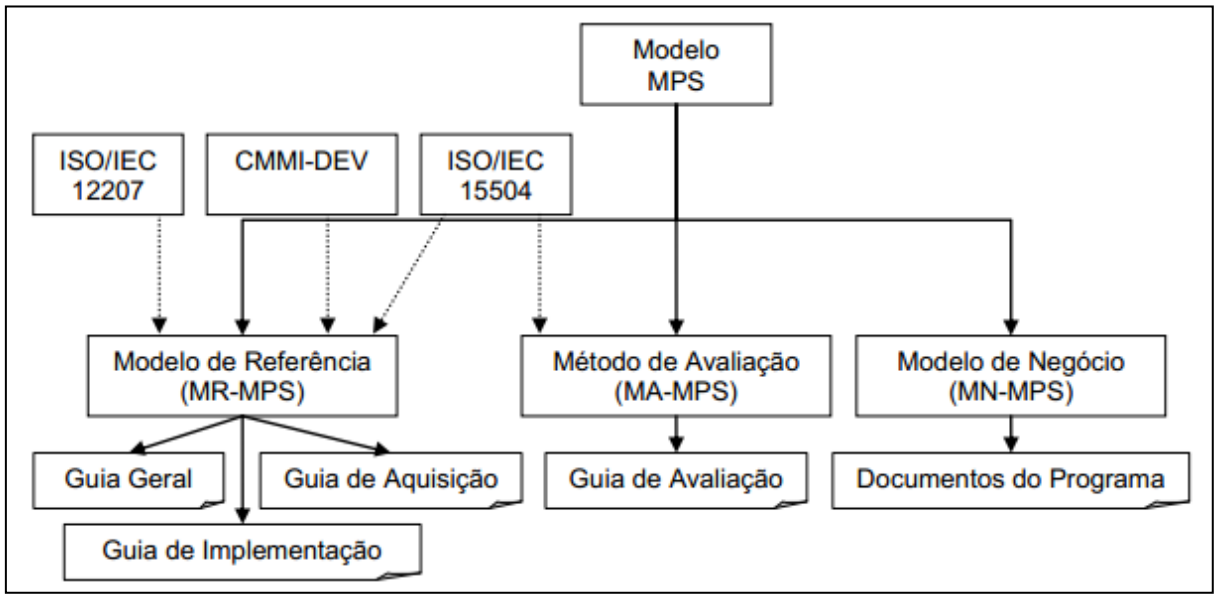
2.3 MPS.BR

O MPS.BR tem o objetivo de melhorar o processo de software brasileiro. Ele tem duas metas a cumprir. A primeira visa apenas à criação e aprimoramento do modelo MPS. A segunda visa à disseminação e adoção do modelo em todas as regiões do país.

Para desenvolver suas atividades, o programa MPS.BR, tem apoio do Fórum de Credenciamento e Controle (FCC) e da Equipe Técnica do Modelo (ETM) (SOFTEX, 2011 a, p. 4).

O modelo MPS está dividido em três componentes conforme a Figura 2, cujos componentes são: Modelo de Referência (MR-MPS), Método de Avaliação (MA-MPS) e Modelo de Negócio (MN-MPS). Cada componente é descrito por meio de guias e/ou

documentos do modelo MPS.



Fonte: Softex (2011 c).

Figura 2 – Componentes do modelo MPS

No modelo de referência encontram-se os requisitos que os processos das unidades organizacionais precisam atender para estar em conformidade com o MR-MPS. A descrição deste modelo está no Guia Geral que contém as definições dos níveis de maturidade, processos e atributos do processo (SOFTEX, 2011 c, p. 13). O Guia de Aquisição é um documento complementar, do modelo de referência, destinado a organizações que pretendam adquirir software e serviços correlatos. Este guia não contém requisitos do MR-MPS, mas contém boas práticas para a aquisição de software e serviços correlatos. O Guia de Implementação, outro documento complementar do modelo de referência, contém sugestões de como implementar cada um dos níveis do MR-MPS. Segundo Softext (2011 c, p.14), as explicações presentes nos guias de implementação não representam requisitos do modelo. Assim devem ser consideradas apenas como informativos.

Conforme Softex (2011 c, p.14), o Guia de Avaliação contém o processo e o método de avaliação MA-MPS, os requisitos para os avaliadores líderes, avaliadores adjuntos e Instituições Avaliadoras (IA).

Já o Modelo de Negócio MN-MPS descreve regras de negócio para implementação do MR-MPS pelas Instituições Implementadoras (II), avaliação seguindo o MA-MPS pelas Instituições Avaliadoras (IA), organização de grupos de empresas pelas Instituições Organizadoras de Grupos de Empresas (IOGE) para implementação do MR-MPS e avaliação

MA-MPS, certificação de Consultores de Aquisição (CA) e programas anuais de treinamento do MPS.BR por meio de cursos, provas e workshops (SOFTEX, 2011 c, p.14).

O Modelo de Referência MR-MPS define níveis de maturidade que são uma combinação entre processos e sua capacidade. Neste modelo é declarado o propósito e os resultados esperados de sua execução. Isso permite avaliar e atribuir graus de efetividade na execução dos processos.

A capacidade do processo é a caracterização da habilidade do processo para alcançar os objetivos de negócio, tanto os atuais como os futuros. Está relacionada com o atendimento dos atributos de processo associados aos processos de cada nível de maturidade.

2.3.1 Níveis de Maturidade

Conforme Softex (2011 c, p.16), os níveis de maturidade estabelecem patamares de evolução de processos, que caracterizam estágios de melhoria da implementação de processos na organização. O nível de maturidade em que se encontra uma organização permite prever o seu desempenho futuro ao executar um ou mais processos.

O MR-MPS define sete níveis de maturidade, A (Em Otimização), B (Gerenciado Quantitativamente), C (Definido), D (Largamente Definido), E (Parcialmente Definido), F (Gerenciado) e G (Parcialmente Gerenciado). A escala de maturidade se inicia no nível G e progride até o nível A.

Nível	**Processos**
A	Em otimização
B	Gerência de Projetos (Evolução)
C	Gerência de Riscos
	Desenvolvimento para Reutilização
	Gerência de Decisões
D	Verificação
	Validação
	Projeto e Construção de Produto
	Integração do Produto
	Desenvolvimento de Requisitos
E	Gerência de Projetos

	Gerência de Utilização
	Gerência de Recursos Humanos
	Definição do Processo Organizacional
	Avaliação e Melhoria do Processo Organizacional
F	Medição
	Garantia da Qualidade
	Gerência de Portifolio de Projetos
	Gerência de Configuração
	Aquisição
G	Gerência de Requisitos
	Gerência de Projetos

Quadro 1 – Níveis de maturidade do MR-MPS

Para cada um destes sete níveis de maturidade é atribuído um perfil de processos que indicam onde a organização deve colocar o esforço de melhoria. O progresso e o alcance de um determinado nível de maturidade do MR-MPS ocorrem quando são atendidos os propósitos e todos os resultados esperados dos respectivos processos e os resultados esperados dos atributos de processo estabelecidos para aquele nível (SOFTEX, 2011 c, p.16).

Segundo Softex (2011 c, p.16), a divisão em sete estágios tem o objetivo de possibilitar uma implementação e avaliação adequada às micros, pequenas e médias empresas. A possibilidade de se realizar avaliações considerando mais níveis também permite uma visibilidade dos resultados de melhoria de processos em prazos mais curtos.

Para a elaboração deste trabalho foi considerada parte do Guia de Implementação do Nível D está sendo considerada. Portanto os demais componentes e documentos não serão abordados aqui. O foco do trabalho são os resultados esperados com relação a gestão de defeitos, item este que é abordado no processo de verificação que é contemplado pelo guia de implementação referenciado acima.

2.3.2 Processo de verificação no MPS.BR

Na visão adotada no MPS.BR, o processo de verificação tem o objetivo de garantir que os serviços ou produtos de trabalho do processo atendam os requisitos especificados. Isso é feito por meio da identificação dos itens a serem verificados, no planejamento da verificação e da execução da verificação (SOFTEX, 2011 b, p. 51).

Na ferramenta aqui especificada e desenvolvida essa garantia é provida por meio do registro dos produtos a serem testados bem como os casos de uso, defeitos e testes que devem ser realizados sobre esse produto. Dessa forma é possível cercar o produto com técnicas que permitem diminuir a incidência de produtos liberados para o mercado com problemas graves.

O processo de verificação do MPS.BR tem seis resultados esperados, que são:

a) VER1 – Produtos de trabalhos a serem verificados são identificados;
b) VER2 – Uma estratégia de verificação é desenvolvida e implementada, estabelecendo cronograma, revisores envolvidos, métodos para verificação e qualquer material a ser utilizado na verificação;
c) VER3 – Critérios e procedimentos para verificação dos produtos de trabalho a serem verificados são identificados e um ambiente para verificação é estabelecido;
d) VER4 – Atividades de verificação, incluindo testes e revisões por pares, são executados;
e) VER5 – Defeitos são identificados e registrados e
f) VER6 – Resultados de atividades de verificação são analisados e disponibilizados para as partes interessadas.

Para atender ao resultado esperado para o VER1 é necessário analisar os produtos de trabalho que serão produzidos ao longo do projeto e selecionar aqueles a serem verificados. Segundo Softex (2011 b, p.54), alguns possíveis produtos de trabalho selecionados para a verificação, por sua importância, podem ser o plano do projeto, o documento de requisitos, o documento de análise, o documento de projeto e o código-fonte.

O alcance do resultado esperado para o VER2 envolve a definição de uma estratégia de verificação que descreva os procedimentos, a infraestrutura necessária e as responsabilidades pelas atividades de verificação. Para Softex (2011 b, p.55), os métodos que serão usados para verificação de cada produto de trabalho selecionado para verificação devem ser identificados, garantindo, em cada projeto, a realização de algum tipo de revisão por pares e testes. As ferramentas que apoiarão a execução das atividades de verificação também devem ser definidas.

Para chegar-se ao resultado esperado do VER3 é necessária a definição dos critérios e procedimentos que serão utilizados para a verificação de cada produto de trabalho e na preparação do ambiente para verificação, disponibilizando ferramentas, recursos de hardware, infraestrutura de rede e outros recursos necessários à execução das atividades planejadas (SOFTEX, 2011 b, p.58).

O resultado esperado do VER4 visa à garantia de que as atividades de verificação são executadas conforme planejado, o que inclui, obrigatoriamente, a realização de revisão por pares e testes (SOFTEX, 2011 b, p.59).

Conforme Softex (2011 b, p.59), o resultado esperado para o VER5, objetiva garantir que os defeitos identificados durante a execução da verificação são documentados e registrados. Para registro dos defeitos identificados pode-se usar uma classificação de defeitos, por exemplo, por severidade (crítico, sério, moderado) ou por origem (requisitos, projeto (design), código, testes).

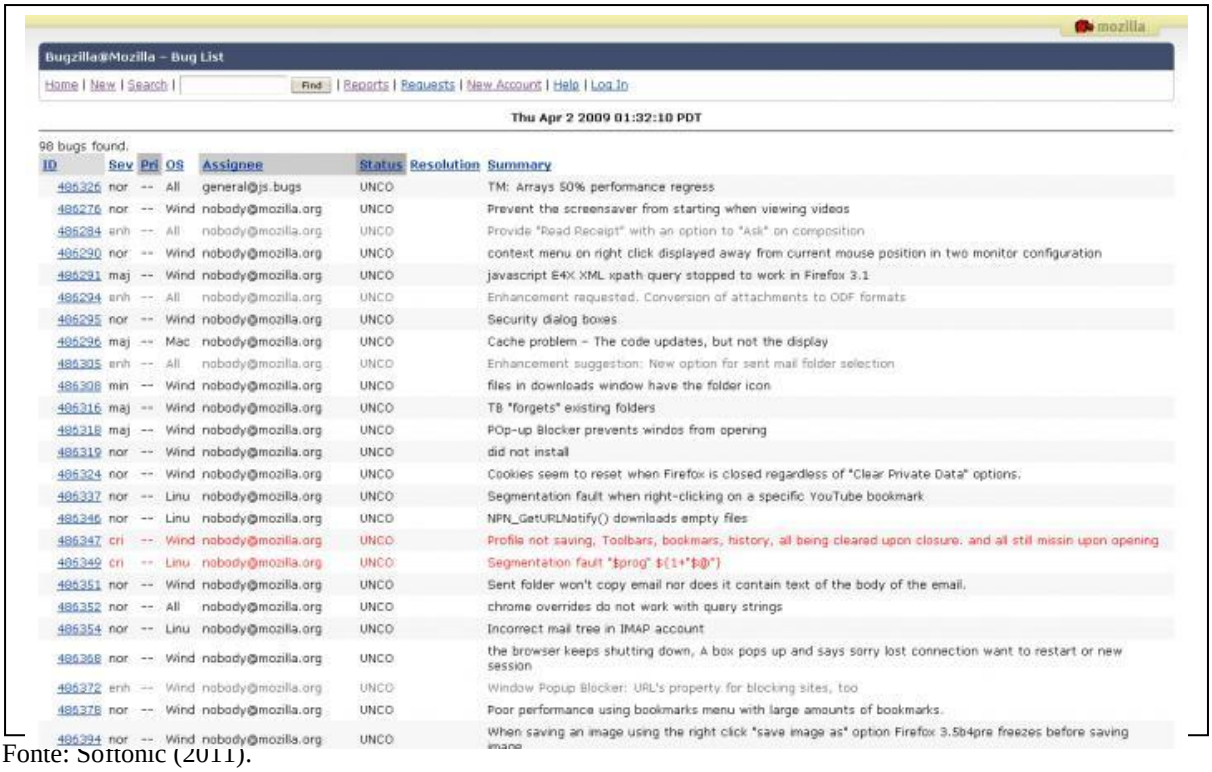
Alcançar o resultado esperado do VER6 envolve realizar uma análise dos resultados obtidos em cada atividade de verificação e disponibilizar estes resultados para as partes interessadas. Assim, uma forma de alcance deste resultado é pela análise de laudos de avaliação e relatórios de testes que contenham informações sobre os resultados obtidos após a realização das atividades de verificação (SOFTEX, 2011 b, p.60). Após a realização da atividade de verificação desse item será possível responder se os critérios definidos foram atendidos, se as ações corretivas planejadas foram concluídas, se a verificação foi executada conforme o planejamento e se os resultados obtidos permitem a aprovação do artefato verificado.

2.4 TRABALHOS CORRELATOS

Foram pesquisados alguns softwares que atuam na área de gestão de defeitos disponíveis no mercado como segue.

O primeiro dos softwares pesquisados é o Bugzilla que é uma ferramenta para gestão de erros e defeitos. Ela permite que programadores individuais ou grupos de programadores acompanhem os relatórios de erros em seus produtos (BUGZILLA, 2009).

Na Figura 3 apresentada a seguir é possível visualizar a lista de defeitos do Bugzilla. Nela são exibidos os registros de defeitos realizados pelos usuários, que durante o ciclo de testes de algum produto de trabalho, encontraram algum defeito que precisasse de ajuste para ter condições de ser liberado ao mercado.



Fonte: Softonic (2011).

Figura 3 – Lista de defeitos no Bugzilla

Outro software encontrado foi o JIRA que é um sistema desenvolvido para monitorar, gerenciar e fechar erros. A ferramenta monitora e acompanha a vida inteira de um erro, possui integração com diversas ferramentas de desenvolvimento além de fornecer relatórios para acompanhamento de projetos e recursos (ATLASSIAN, 2009).

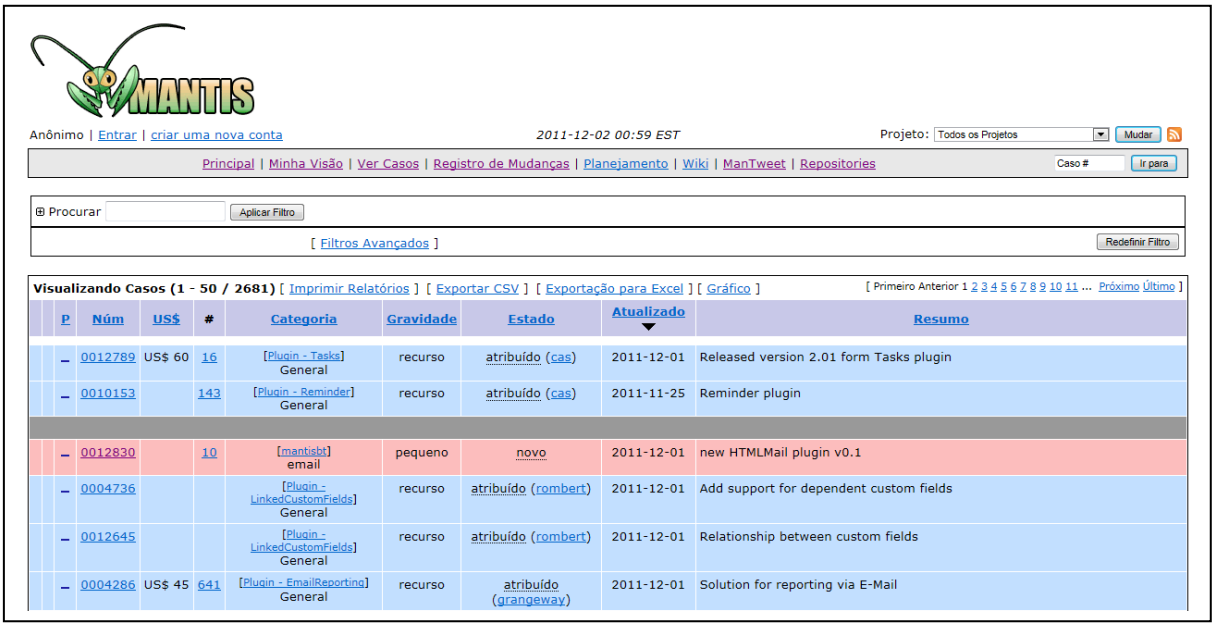
Na Figura 4 está apresentada a tela que exibe a lista de defeitos cadastrados no banco de dados da ferramenta.

Fonte: Atlassian (2011).

Figura 4 – Lista de defeitos no JIRA

Também foi pesquisado um software chamado MantisBT que é um sistema *web*, livre, de rastreamento de erros. É escrito na linguagem PHP e pode ser utilizado com MySQL, MS SQL e PostgreSQL e um servidor *web*. Ele é liberado sob os termos da GNU *General Public License* (GPL). Entre seus recursos estão, notificação por e-mail, anexos que podem ser armazenados no servidor web ou no banco de dados e integração com ferramentas de controle de versão de código (MANTIS, 2009).

Na Figura 5 é apresentada a tela que permite a visualização dos defeitos registrados no banco de dados da ferramenta.



Fonte: Mantis (2011).

Figura 5 – Lista de defeitos no Mantis

A última ferramenta pesquisada é projeto Scarab que tem o objetivo de construir um sistema de rastreamento de artefatos altamente personalizável. Entre os seus recursos estão a entrada de dados, consultas, relatórios, notificações aos interessados, comentários colaborativos e rastreamento de dependências (TIGRIS, 2009).

A Figura 6 apresenta um exemplo de tela com a lista de defeitos existentes no banco de dados do Scarab.

Fonte: Scarab (2011).

Figura 6 – Lista de defeitos no Scarab

3 DESENVOLVIMENTO DA FERRAMENTA

Neste capítulo está descrita a ferramenta e são apresentados alguns dos requisitos funcionais, alguns dos requisitos não funcionais, os principais diagramas de caso de uso e o diagrama entidade relacionamento, bem como a operacionalidade do sistema.

3.1 LEVANTAMENTO DE INFORMAÇÕES

A ferramenta de gestão de defeitos, que durante o trabalho será chamada apenas de GesDef, será utilizada por pessoas que exercem, principalmente os papeis de testador e desenvolvedor entre outros que podem ser responsabilizados pela gerência das atividades realizadas no setor, como por exemplo, analistas ou gerentes de projeto. Para fins de utilização deste sistema trataremos todos apenas como usuário pois todos tem as mesmas permissões com exceção do usuário administrador que tem permissão para editar e excluir registros da tabela de usuários.

O processo de gestão de defeitos, concebido para esta ferramenta, é iniciado pelo usuário. Durante a realização dos testes de software, caso o usuário encontrar defeitos, realiza o registro destes. Cada defeito será armazenado com um número mínimo e padronizado de informações. Essas informações serão descritas adiante.

O segundo passo do processo a ser realizado pelo usuário é verificar os registros de defeito realizados. Em seguida revisar o registro do problema com o desenvolvedor que realizará a correção do defeito para determinar se o problema é mesmo um defeito.

O terceiro passo do processo é realizado pelo usuário depois de o defeito ter sido corrigido, é alterar a situação do defeito para “corrigido”. As situações em que um defeito pode ser classificado serão descritas adiante.

Nesse momento o defeito volta a ser verificado, onde se dá o início do quarto e último passo do processo. Um dos usuários realiza o teste do defeito novamente para que seja certificado de que o defeito foi realmente corrigido e que a correção não tenha gerado novos defeitos.

O defeito será novamente classificado de acordo com a sua situação. Ele pode, por exemplo, ser classificado como “fechado” se o defeito estiver realmente corrigido ou

novamente “aberto” caso o defeito não esteja completamente corrigido. Nesta última situação o defeito deve voltar para o desenvolvedor para que seja analisado e corrigido.

3.2 ESPECIFICAÇÃO

O Quadro 2 apresenta os requisitos funcionais previstos para o sistema e sua rastreabilidade, com o(s) caso(s) de uso associado(s).

Requisitos Funcionais	**Caso de Uso**
RF01: O sistema deve permitir que os usuários acessem o sistema mediante a validação de um apelido e senha.	UC01
RF02: O sistema deve permitir o cadastro de usuários.	UC02
RF03: O sistema deve permitir o cadastro de setores.	UC03
RF04: O sistema deve permitir o cadastro de produtos.	UC04
RF05: O sistema deve permitir o cadastro de versões.	UC05
RF06: O sistema deve permitir o cadastro de casos de uso.	UC06
RF07: O sistema deve permitir o cadastro de testes.	UC07
RF08: O sistema deve permitir o cadastro de defeitos.	UC08
RF09: O sistema deve permitir a alteração da prioridade de correção do defeito.	UC09
RF10: O sistema deve permitir a alteração da severidade do defeito.	UC10
RF11: O sistema deve permitir a alteração da situação de um defeito.	UC11
RF12: O sistema deve permitir a visualização dos defeitos de acordo com a sua prioridade.	UC12
RF13: O sistema deve permitir a visualização dos defeitos de acordo com a sua severidade.	UC13
RF14: O sistema deve permitir a visualização dos defeitos de acordo com a sua situação.	UC14

Quadro 2 – Requisitos funcionais

O Quadro 3 apresenta os requisitos não funcionais previstos para o sistema.

Requisitos Não Funcionais
RNF01: O sistema deve ser *web*, mas com acesso apenas na intranet da empresa.
RNF02: O sistema deve ser desenvolvido com XHTML 1.0, JSF e Java.
RNF03: O sistema deve utilizar o banco de dados MySQL 5.5.

RNF04: O sistema deve permitir acesso apenas aos usuários cadastrados.
RNF05: O sistema deve ser executado em um servidor de aplicações como o Apache Tomcat.

Quadro 3 – Requisitos não funcionais

3.3 MODELAGEM

Nesta seção estão os principais diagramas de caso de uso e o diagrama entidade relacionamento para a ferramenta.

3.3.1 Diagramas de caso de uso

Existem apenas dois atores que interagem nos casos de uso. O primeiro deles é o administrador. Este ator é o único com permissão para editar ou excluir registros referentes aos usuários com contas de acesso cadastradas na ferramenta. O segundo dos atores é o usuário. Por padrão da ferramenta todo usuário é cadastrado como usuário padrão na ferramenta. Para que um usuário padrão possa se tornar administrador é necessário que um usuário que já tenha a permissão de administrador, altere a permissão do usuário padrão. Um usuário padrão tem permissão para inclusão, edição e exclusão de registros, com exceção do acesso para edição ou exclusão dos dados dos usuários.

A Figura 7 exibe os casos de uso referentes às funcionalidades disponíveis na ferramenta. O detalhamento dos principais casos de uso foi feito no Apêndice A.

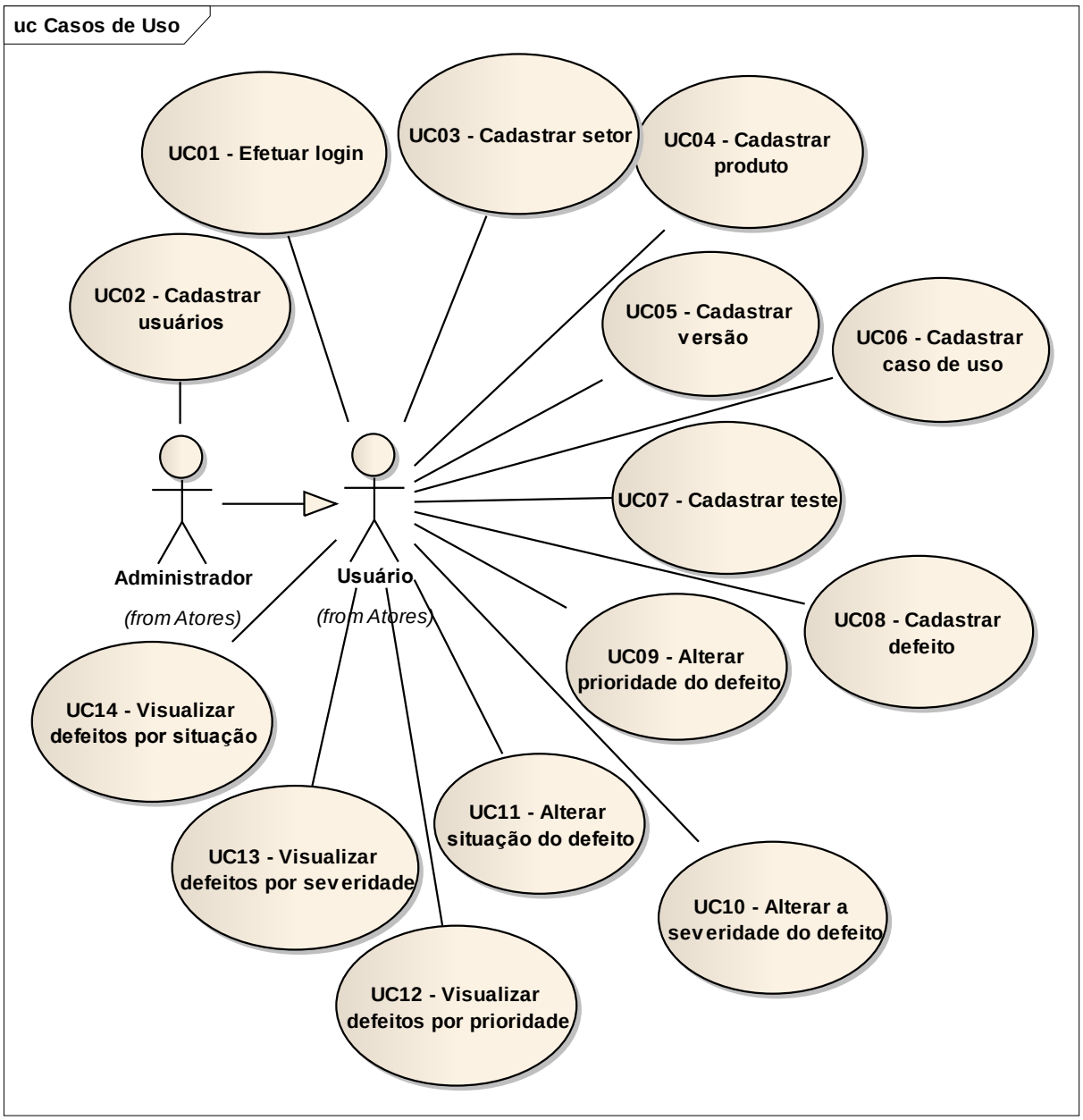


Figura 7 – Casos de uso

3.3.2 Diagrama entidade relacionamento

O diagrama entidade relacionamento deste projeto é composto por oito tabelas cujos propósitos são os seguintes:

a) tabela usuario: nesta tabela são armazenadas as informações a respeito dos usuários

que utilizarão a ferramenta. Nela são armazenados, um código único, um campo ativo que tem a finalidade de representar se o usuário está ativou ou inativo na ferramenta, *login* e a senha, celular, *e-mail*, idioma, nascimento, papel que exerce e o setor a qual a que pertence como chave estrangeira;

b) tabela setor: a tabela setor armazena as informações a respeito do setor de trabalho em que o usuário atua na empresa. Para tanto nela ficam armazenados um código único e o nome do setor da empresa;
c) tabela produto: na tabela de produtos armazena-se os dados dos produtos que serão testados. Os campos dessa tabela são, o código único, o nome dos produtos além da chave estrangeira para a tabela que armazena informações sobre os setores;
d) tabela versao: na tabela de versões ficam armazenadas as informações pertinentes as versões dos produtos que serão testadas. Cada produto pode ter várias versões diferentes então para auxiliar nesse controle, a tabela de versões armazenará um código único, o nome, a data de início e data de término dos testes da versão em questão. Nesta tabela fica ainda a chave estrangeira para a tabela que armazena informações sobre os produtos;
e) tabela teste: a tabela de testes armazena os dados dos testes realizados sobre um produto. Os campos dessa tabela são, um código único, o título, prioridade de execução do teste, status de andamento do teste, resultado da execução do teste e a descrição do teste além de chave estrangeira para a tabela que armazena informações sobre os produtos e também para a tabela que armazena informações sobre os casos de uso;
f) tabela defeito: na tabela de defeito ficam armazenados os dados a respeito dos defeitos que são encontrados durante a execução dos testes do produto. Os campos dessa são, um código único, o título do defeito, a prioridade de resolução do defeito, a situação em que o defeito se encontra, a severidade do defeito, a descrição do defeito e a solução dada ao defeito além de chave estrangeira para a tabela que armazena informações sobre os testes;
g) tabela casouso: a tabela para os casos de uso armazena um código único, um título e a descrição do caso de uso. Essas informações são interessantes, pois auxiliam na organização dos testes em grupos;
h) tabela usuario_permissao: nesta tabela ficam apenas as informações necessárias ao controle de acesso dos usuários. As informações armazenadas são um campo de ligação entre esta tabela e a tabela de usuários e um campo para armazenar o tipo de

permissão de acesso a aplicação.

A Figura 8 exibe o diagrama entidade relacionamento de acordo com as necessidades da ferramenta de gestão de defeitos.

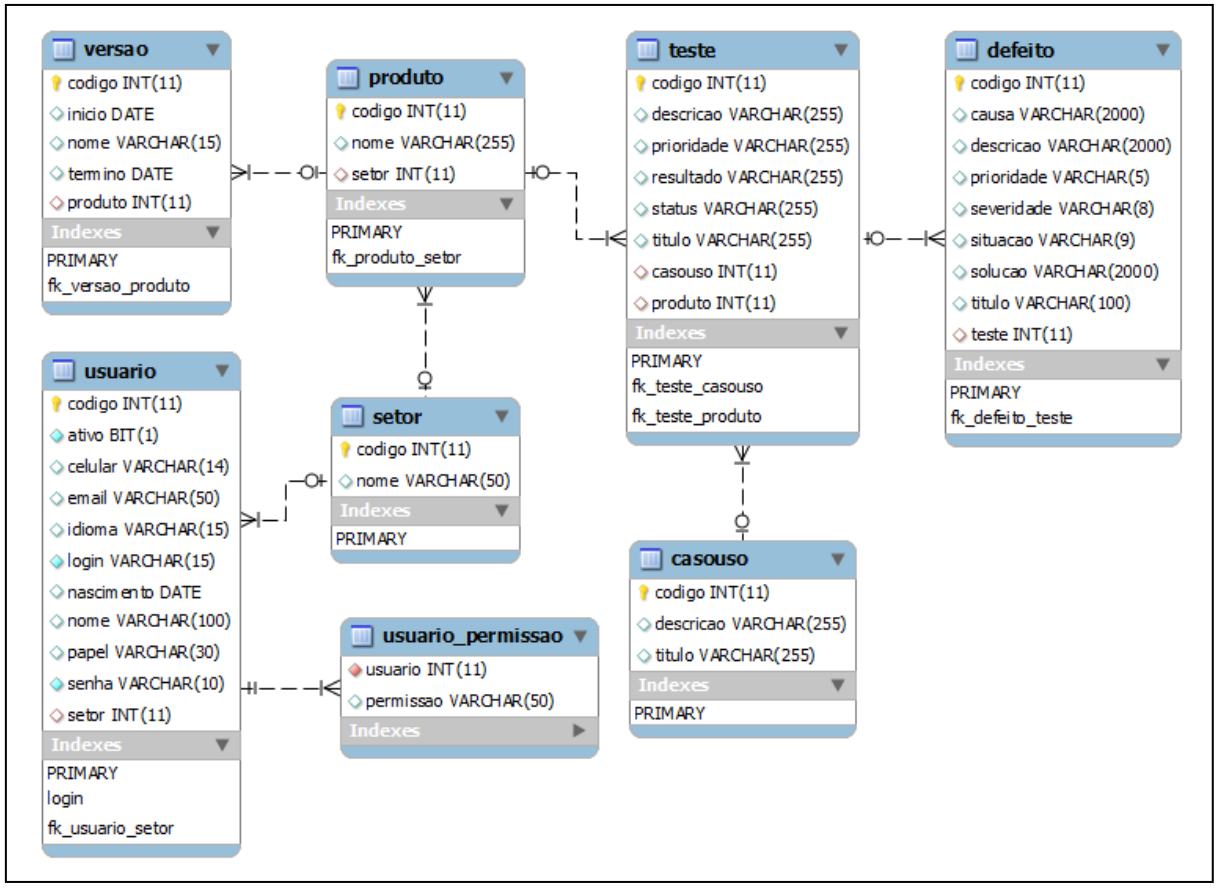


Figura 8 – Diagrama entidade relacionamento da ferramenta

O dicionário de dados, referente as tabelas da ferramenta, que visa oferecer uma explicação de forma textual sobre os campos contidos nas tabelas pode ser encontrado no Apêndice B.

3.4 IMPLEMENTAÇÃO

Nesta seção está o detalhamento sobre a implementação da ferramenta. No tópico inicial identificam-se as técnicas e ferramentas utilizadas para o desenvolvimento da ferramenta. No tópico seguinte apresenta-se um estudo de caso do ponto de vista do usuário,

destacando a funcionalidade ou operacionalidade da ferramenta. No último tópico descreve-se os resultados obtidos após o término da implementação da ferramenta.

3.4.1 Técnicas e ferramentas utilizadas

Para a implementação da ferramenta de apoio ao processo de gestão de defeitos foi utilizada a ferramenta Eclipse Indigo onde foram desenvolvidos os códigos fonte da aplicação. Também foram utilizadas as seguintes bibliotecas:

a) MySQL Connector 5.1.17, biblioteca utilizada para estabelecer a conexão com o banco de dados;
b) iReport 4.1.3, ferramenta utilizada para a criação dos relatórios;
c) Hibernate 3.6.1, biblioteca utilizada para realizar a persistência dos dados;
d) Spring Security 3.0.5; biblioteca utilizada para auxiliar no controle de segurança da aplicação;
e) Mojarra 2.1.1, biblioteca utilizada exibir os logs do servidor de aplicação no *browser*;
f) PrimeFaces 2.2.1, biblioteca utilizada para desenvolver a interface com o usuário;
g) Apache Tomcat 7.0.16, utilizado como servidor de aplicação;
h) Para o armazenamento dos dados foi utilizado o banco de dados MySQL 5.5.15;
i) O gerenciamento de banco de dados modelagem de dados foi criada com a ferramenta MySQL Workbench 5.2.35;
j) Enterprise Architect 7.5, ferramenta utilizada para criar os diagramas de caso de uso.

As telas do sistema foram estruturadas em *EXtensible HyperText Markup Language* (XHTML). Para definir os estilos visuais utilizou-se a biblioteca PrimeFaces que disponibiliza diversos componentes de tela com recursos visuais. Para realizar as operações na base de dados foi utilizada a linguagem de programação Java versão 1.6.0.29.

Para integrar as páginas XHTML com os métodos disponibilizados pelas classes Java foi utilizado *Java Server Faces* (JSF).

3.4.2 Operacionalidade da implementação

A ferramenta de apoio ao processo de gestão de defeitos, daqui em diante chamada apena como GesDef, deve ser disponibilizada em um servidor de aplicações, como por exemplo, o Apache Tomcat ou Oracle Glassfish Server. Para fins de exemplificação, considera-se a utilização do servidor de aplicação Apache *Tomcat*. Neste caso basta colocar o arquivo *GesDef.war* no diretório *webapps* do Apache *Tomcat* para que o servidor de aplicações disponibilize a aplicação.

Para a base de dados é necessário criar um esquema no banco de dados chamado *gesdef*. As tabelas são criadas automaticamente no momento em que o primeiro usuário fizer *logon* na ferramenta. Finalizadas essas configurações é possível utilizar a ferramenta.

Na Figura 9 é exibido o formulário de *login* da ferramenta. Neste formulário os usuários cadastrados devem informar seu apelido e senha para terem acesso à ferramenta. Caso ainda não sejam usuários cadastrados, podem utilizar o botão “Registre-se” para acessar o formulário de cadastro e informar seus dados para que seja criada uma conta de usuário e então utilizar seus dados para acessar o sistema.

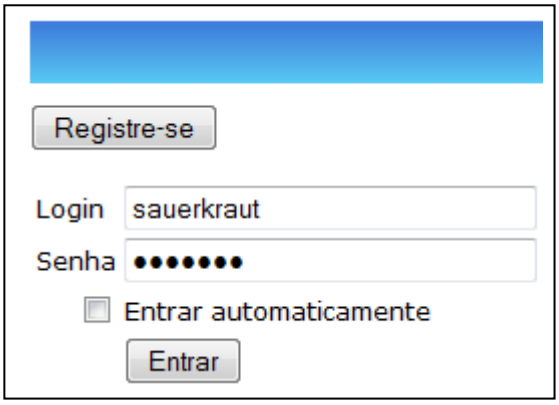


Figura 9 – Formulário de *login*

Caso o usuário informe incorretamente um dos dados, será exibida mensagem informando a causa da falha no *login.*

Na Figura 10, é exibida a mensagem visualizada pelo usuário caso ele informe incorretamente um dos dados.

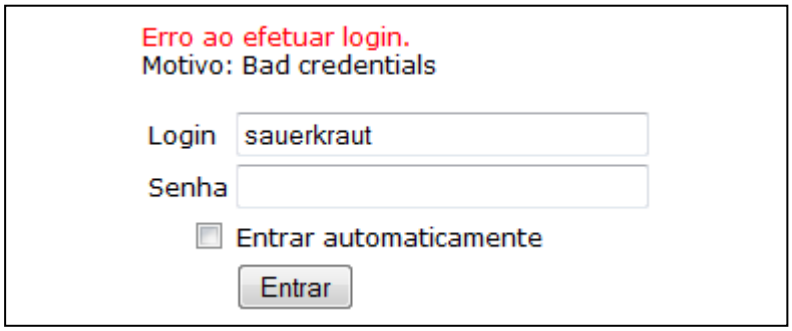


Figura 10 – Mensagem informativa sobre dados inválidos

No Quadro 4 tem-se o código do arquivo *applicationContext-security.xml*. Nele é feita a configuração dos formulários acessíveis pelos usuários definidos como administradores e também dos formulários acessíveis pelos usuários convencionais.

```
<?xml version="1.0" encoding="UTF-8"?>
<b:beans xmlns=http://www.springframework.org/schema/security
  xmlns:b=http://www.springframework.org/schema/beans
  xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
  xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
http://www.springframework.org/schema/security
http://www.springframework.org/schema/security/spring-security-3.0.xsd">
<http>
  <intercept-url pattern="/admin/**" access="ROLE_ADMINISTRADOR" />
  <intercept-url pattern="/restrito/**" access="ROLE_USUARIO" />
  <form-login login-page="/publico/login.jsf"
    always-use-default-target="true"
    default-target-url="/restrito/principal.jsf"
    authentication-failure-url="/publico/login.jsf?login_error=1" />
    <logout/>
    <remember-me />
</http>
<authentication-manager>
  <authentication-provider>
    <jdbc-user-service data-source-ref="gesdefDataSource"
      authorities-by-username-query="SELECT u.login, p.permissao
        FROM usuario u, usuario_permissao p
        WHERE u.codigo = p.usuario
        AND u.login = ?"
      users-by-username-query="SELECT login, senha, ativo
        FROM usuário
        WHERE login = ?" />
  </authentication-provider>
</authentication-manager>
</b:beans>
```

Quadro 4 – Código que verifica dados para *login*

Caso a validação dos dados seja realizada com sucesso, o *login* é realizado e o usuário será redirecionado para a tela principal da ferramenta. Na Figura 11 que será apresentada a seguir, é exibida a tela principal da ferramenta. Nesta tela os usuários tem acesso a botões que levam aos formulários de cadastro de setores, produtos, versões, testes, casos de uso e defeitos além de um ícone que permite, sempre que desejado, voltar à tela principal e outro que permite que o usuário saia da aplicação. Caso o usuário ativo seja um administrador, é exibido ainda um ícone que permite o acesso a uma listagem de usuários. Neste formulário é possível incluir, alterar ou mesmo excluir os usuários cadastrados na base de dados e gerar relatório

contendo a listagem de usuários.

Figura 11 – Tela principal

Nesse ponto o usuário pode iniciar a utilização da ferramenta pressionando um dos botões disponíveis na barra de botões. Posicionando-se o cursor do mouse sobre os botões, é apresentada mensagem informando uma breve descrição da finalidade do botão selecionado.

Seguindo o caminho básico, o primeiro cadastro a ser realizado é o de setores. Neste cadastro os usuários informam a qual setor de trabalho eles pertencem. A Figura 12 apresenta o formulário para o cadastro de setores.

Figura 12 – Cadastro de setores

O usuário informa um nome para o seu setor de trabalho e pressiona o botão (Salvar). O campo nome recebe valores alfanuméricos de até 50 caracteres. Na sequencia o registro é incluído na banco de dados e exibido na listagem de setores que está disponível no mesmo formulário como pode ser verificado na Figura 13 que será exibida adiante.

Figura 13 – Listagem de setores

A listagem dos dados é realizada uma tabela onde é possível fazer a edição ou exclusão dos registros listados. Para isso basta pressionar o ícone referente a opção desejada na coluna (Editar) ou coluna (Excluir) da tabela. Caso o usuário opte por editar, o registro é exibido no formulário de cadastro. Basta alterar e salvar novamente para que a alteração tenha efeito. Caso o usuário opte por excluir o registro, o sistema faz a verificação se o registro está sendo utilizado por outro registro no banco. Caso o registro ainda não esteja vinculado, é excluído sem problemas, do contrário uma mensagem é exibida informando a impossibilidade de excluir o registro.

Ainda no mesmo formulário é possível gerar relatórios com os campos disponíveis na listagem utilizando o formulário específico.

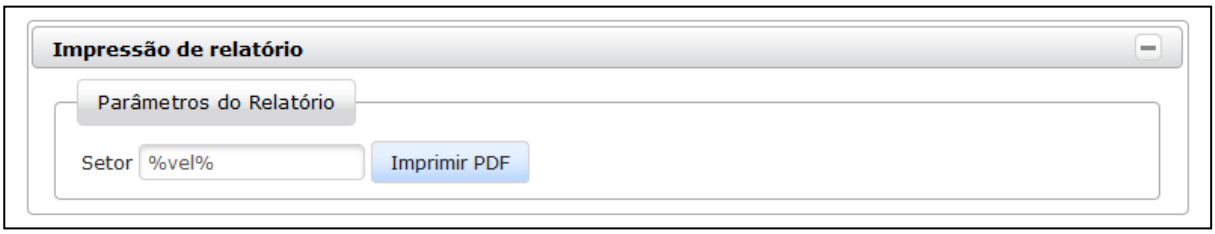


Figura 14 – Formulário para geração de relatório de setores

O resultado da consulta realizada com o parâmetro, conforme a Figura 14 é um relatório no formato *portable document format* (PDF) que será mostrado logo adiante quando o cadastro de defeitos estiver sendo discutido. Com isto estão finalizadas as explicações acerca do cadastro de setores.

O próximo cadastro a ser realizado é o cadastro de produtos. O cadastro de produtos apresenta os seguintes campos:

a) campo Nome: neste campo deve ser informado o nome do produto a ser testado;
b) campo Setor: neste campo deve ser informado o nome do setor ao qual o produto pertence.

Figura 15 – Cadastro de produtos

Da mesma forma como ocorre com o cadastro de setores, o cadastro de produtos e demais cadastros apresentam um formulário para exibir uma listagem dos registros existentes no banco de dados e um formulário para parâmetros de geração de relatório. Estes seguem o mesmo padrão e portando não serão mais exibidos com figuras.

Seguindo com os cadastros, é preciso cadastrar as versões em que os produtos estão disponíveis. O cadastro de versão apresenta os seguintes campos:

a) campo Título: neste campo o usuário informa um título para a versão a ser cadastrada;
b) campo Início: o campo início tem a finalidade de armazenar a data inicial do ciclo de testes para a versão;
c) campo Término: o campo término armazena a data final do ciclo de testes para a versão ;
d) campo Produto: neste campo o usuário seleciona a qual produto a versão pertence. Este é um campo que trás a lista de produtos registrados no banco de dados, portanto não é um campo digitável.

Figura 16 – Cadastro de versões

A continuidade da inclusão dos registros no banco de dados dá-se com a inclusão dos casos de uso. O cadastro de casos de uso apresenta os seguintes campos:

a) campo Título: neste campo o usuário informa um título para o caso de uso;
b) campo Descrição: o campo descrição deve receber o detalhamento do caso de uso.

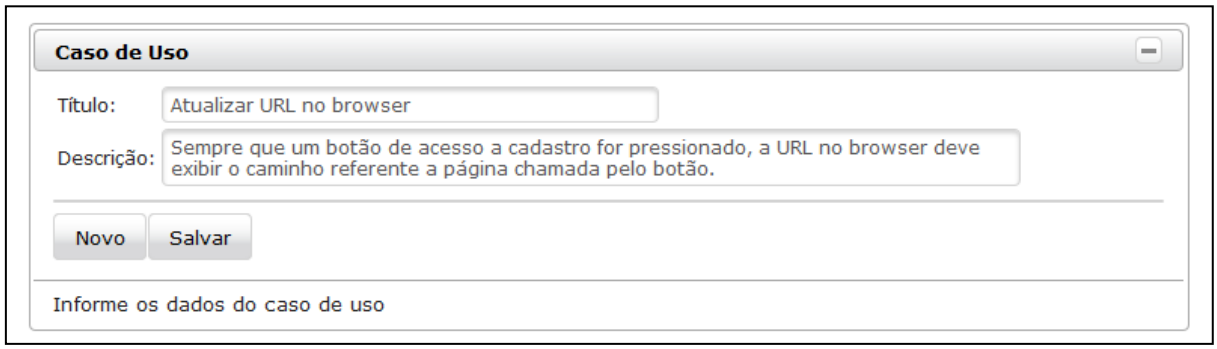


Figura 17 – Cadastro de casos de uso

O ciclo de testes continua com a inclusão de testes no banco de dados. Para isso o cadastro de testes oferece o seguinte conjunto de campos:

a) campo Título: campo para armazenar um título de referência ao teste;
b) campo Descrição: o usuário descreve os passos a serem realizados durante a execução dos testes;
c) campo Prioridade: a prioridade do teste pode ser alta, média ou baixa e serve de auxílio para priorização para da execução do teste;
d) campo Status: neste campo o usuário informa o estado de andamento do teste que pode ser pendente, caso ainda não tenha sido realizado, testando, se algum usuário está realizando o teste no momento ou realizado se o teste já foi executado por algum usuário;

e) campo Resultado: este campo armazena se o teste foi aprovado ou reprovado;
f) campo Caso de Uso: o usuário informa o caso de uso ao qual o teste pertence e
g) campo Produto: o nome do produto ao qual o teste pertence.

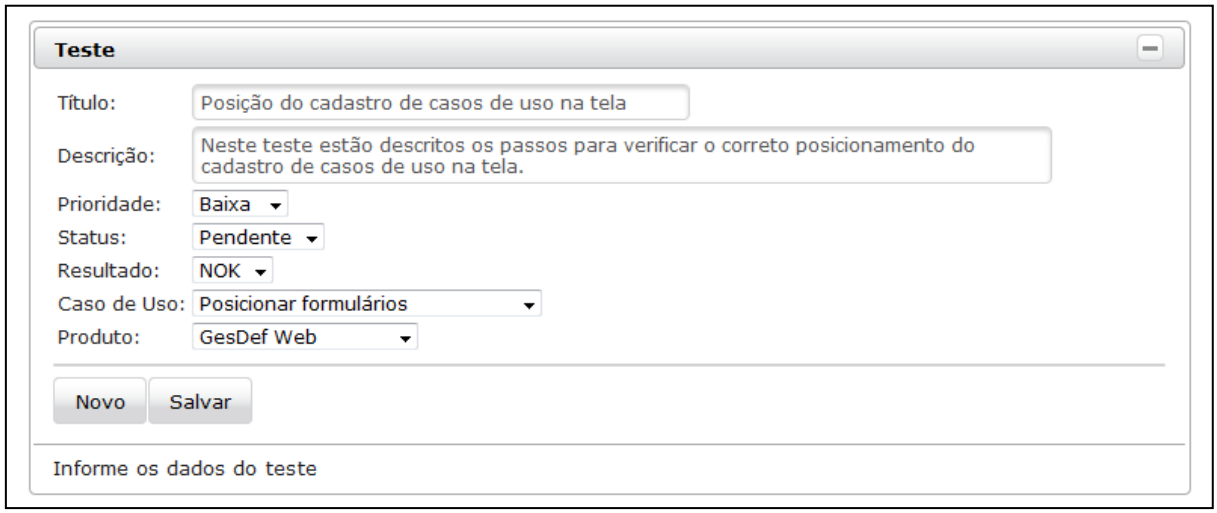


Figura 18 – Cadastro de testes

Para se finalizar os cadastros é necessário incluir os defeitos. Neste formulário estão dispostos os seguintes campos:

a) campo Título: neste campo deve ser informada uma breve descrição do defeito com até 100 caracteres;
b) campo Causa: após a análise do defeito o usuário que fizer a correção do defeito, pode informar o que causou o defeito;
c) campo Descrição: neste campo o usuário que encontrou o defeito durante os testes, pode informar uma descrição detalhada do problema bem como os passos a seguir para que o defeito possa ser reproduzido;
d) campo Prioridade: o campo prioridade apresenta as opções "Alta, Média e Baixa" que representam a urgência da correção do defeito;
e) campo Severidade: o campo severidade apresenta as opções "Crítico, Sério e Moderado" que representam o quão grave é o defeito;
f) campo Situação: o campo situação representa se o defeito ainda não teve tratamento, se foi corrigido mas não testado e se foi testado e portanto fechado. As opções disponíveis no campo são "Aberto, Corrigido e Fechado";
g) campo Solução: neste campo o usuário que realizar a correção do defeito, pode informar quais as alterações que ele realizou para conseguir fazer a correção do

defeito;

h) campo Teste: neste campo deve ser informado o teste (previamente cadastrado) que foi realizado e que possibilitou o registro do defeito em questão.

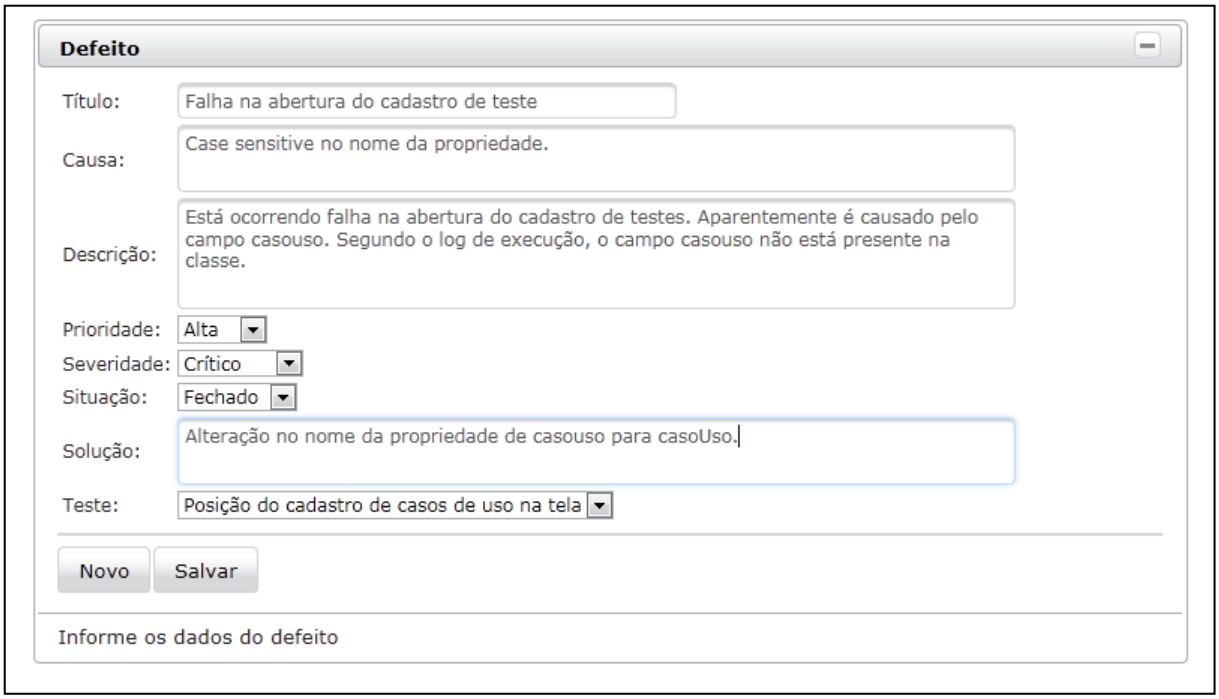


Figura 19 – Formulário de registro de defeitos

Nos trechos de código que seguem, é possível observar como a aplicação realiza a inclusão do registro na base de dados. No Quadro 5 está o código necessário para criar o formulário que o usuário utilizará para inserir os dados e que está representado pela Figura 19.

```
<h:panelGrid columns="2">
    <h:outputLabel value="Título:" for="titulo" />
    <p:inputText id="titulo" value="#{defeitoBean.defeito.titulo}" />
    <h:outputLabel value="Causa:" for="causa" />
    <p:inputTextarea id="causa" value="#{defeitoBean.defeito.causa}" />
    <h:outputLabel value="Descrição:" for="descricao" />
    <p:inputTextarea id="descricao"
value="#{defeitoBean.defeito.descricao}" />
    <h:outputLabel value="Prioridade:" for="prioridade" />
    <h:selectOneMenu id="prioridade"
value="#{defeitoBean.defeito.prioridade}">
      <f:selectItem id="alta" itemLabel="Alta" itemValue="Alta" />
      <f:selectItem id="media" itemLabel="Média" itemValue="Média" />
      <f:selectItem id="baixa" itemLabel="Baixa" itemValue="Baixa" />
    </h:selectOneMenu>
    <h:outputLabel value="Severidade:" for="severidade" />
    <h:selectOneMenu id="severidade"
value="#{defeitoBean.defeito.severidade}">
      <f:selectItem id="critico" itemLabel="Crítico" itemValue="Crítico"
/>
      <f:selectItem id="serio" itemLabel="Sério" itemValue="Sério" />
      <f:selectItem id="moderado" itemLabel="Moderado"
itemValue="Moderado"/>
    </h:selectOneMenu>
    <h:outputLabel value="Situação:" for="situacao" />
    <h:selectOneMenu id="situacao"
value="#{defeitoBean.defeito.situacao}">
      <f:selectItem id="aberto" itemLabel="Aberto" itemValue="Aberto" />
      <f:selectItem id="corrigido" itemLabel="Corrigido"
itemValue="Corrigido"/>
      <f:selectItem id="fechado" itemLabel="Fechado" itemValue="Fechado"
/>
    </h:selectOneMenu>
    <h:outputLabel value="Solução:" for="solucao" />
    <p:inputTextarea id="solucao" value="#{defeitoBean.defeito.solucao}"
/>
    <h:outputLabel value="Teste:" for="teste" />
    <h:selectOneMenu id="teste" value="#{defeitoBean.defeito.teste}"
      converter="testeConverter">
      <f:selectItems value="#{testeBean.lista}" var="teste"
itemValue="#{teste}" itemLabel="#{teste.titulo}"/>
    </h:selectOneMenu>
</h:panelGrid>
<p:commandButton value="Novo" action="#{defeitoBean.novo}"
process="@this" update="edicao" />
<p:commandButton value="Salvar" action="#{defeitoBean.salvar}"
update="listagem, edicao" />
```

Quadro 5 – Formulário de cadastro de defeitos

Após o usuário pressionar o botão “Salvar” no formulário de cadastro o método salvar existente na classe DefeitoBean é acionado. No Quadro 6 está exposto o método salvar.

```
public String salvar() throws DAOException {
    DefeitoRN defeitoRN = new DefeitoRN();
    defeitoRN.salvar(this.defeito);
    return this.destinoSalvar;
}
```

Quadro 6 – Método salvar da classe DefeitoBean

O método salvar tem a finalidade de instanciar um objeto DefeitoRN. A classe DefeitoRN é a classe responsável pelas regras de negócio que controlam as especificidades relacionadas com os defeitos. É nela que o método salvar realmente é executado para então incluir o registro no banco de dados. No Quadro 7 está apresentado o método salvar da classe DefeitoRN.

```
public void salvar(Defeito defeito) {
    Integer codigo = defeito.getCodigo();
    if (codigo == null || codigo == 0) {
        this.defeitoDAO.salvar(defeito);
    } else {
        this.defeitoDAO.atualizar(defeito);
    }
}
```

Quadro 7 – Método salvar da classe DefeitoRN

Neste método é possível observar que caso o código do defeito obtido do formulário de cadastro seja nulo ou seja igual a zero, um novo defeito é salvo no banco de dados. Caso contrário, o registro é apenas atualizado. Para que seja possível gravar o registro no banco de dados é ainda utilizada a classe *DefeitoDAOHibernate* cujo código do método salvar segue a seguir como pode ser visto no Quadro 8. Neste ponto é o *hibernate* que assume o processamento e realiza a inclusão no banco de dados.

```
public void salvar(Defeito defeito) {
    this.session.save(defeito);
}
```

Quadro 8 – Método salvar da classe *DefeitoDAOHibernate*

Para obter uma listagem de dos registros existentes no banco de dados como a listagem apresentada na Figura 13 utilizou-se o código apresentado no Quadro 9.

```
<p:dataTable scrollable="false" value="#{defeitoBean.lista}"
var="defeito" paginator="true" rows="5"
paginatorTemplate="{CurrentPageReport} {FirstPageLink}
{PreviousPageLink} {PageLinks} {NextPageLink} {LastPageLink}
{RowsPerPageDropdown}" rowsPerPageTemplate="5,10,15">
  <p:column style="width:5%">
    <f:facet name="header">Editar</f:facet>
    <p:commandLink action="#{defeitoBean.editar}" update="edicao">
      <h:graphicImage library="imagens" name="editar16.png" />
      <f:setPropertyActionListener target="#{defeitoBean.defeito}"
value="#{defeito}" />
      <f:setPropertyActionListener target="#{defeitoBean.destinoSalvar}"
value="/restrito/defeito" />
    </p:commandLink>
  </p:column>
  <p:column style="width:5%">
    <f:facet name="header">Excluir</f:facet>
    <p:commandLink action="#{defeitoBean.excluir}" update="listagem"
onclick="if (!confirm('Confirma a exclusão do defeito
#{defeito.titulo}?')) return false;">
      <h:graphicImage library="imagens" name="excluir16.png" />
      <f:setPropertyActionListener target="#{defeitoBean.defeito}"
value="#{defeito}" />
    </p:commandLink>
  </p:column>
</p:dataTable>
```

Quadro 9 – Montagem do formulário de listagem

O Quadro 9 apresenta apenas o código necessário para montar a parte visual da listagem. O código necessário para preencher a grade com os registros existentes na base de dados está apresentada no Quadro 10.

```
<p:column style="width:5%">
  <f:facet name="header">Código</f:facet>
  <h:outputText value="#{defeito.codigo}" />
</p:column>
<p:column style="width:20%">
  <f:facet name="header">Título</f:facet>
  <h:outputText value="#{defeito.titulo}" />
</p:column>
<p:column style="width:5%">
  <f:facet name="header">Prioridade</f:facet>
  <h:outputText value="#{defeito.prioridade}" />
</p:column>
<p:column style="width:5%">
  <f:facet name="header">Severidade</f:facet>
  <h:outputText value="#{defeito.severidade}" />
</p:column>
<p:column style="width:5%">
  <f:facet name="header">Situação</f:facet>
  <h:outputText value="#{defeito.situacao}" />
</p:column>
<p:column style="width:20%">
  <f:facet name="header">Teste</f:facet>
  <h:outputText value="#{defeito.teste.titulo}" />
</p:column>
```

Quadro 10 – Grid para exibição de defeitos

Outro recurso interessante e disponível é a possibilidade do usuário gerar relatórios. Nesta ferramenta este recurso está à disposição também no próprio formulário de cadastro/edição/exclusão/listagem. Ele está logo ao final do formulário. Na Figura 20, está exposto o formulário referente ao relatório de defeitos. Neste formulário há quatro campos em que o usuário deve informar o que pretende filtrar do banco de dados. Há ainda um botão que ao ser pressionado, envia os parâmetros informados no formulário para a aplicação que faz a consulta no banco de dados e retorna o resultado em um relatório no formato PDF.

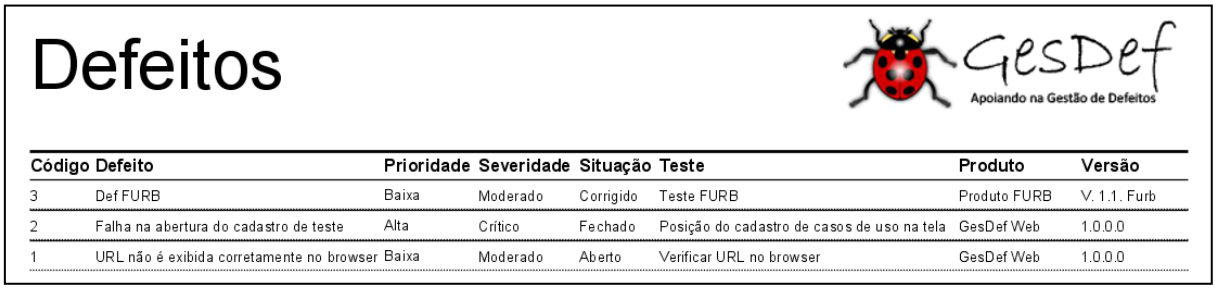
Figura 20 – Filtro para relatório de setores

O código necessário para gerar o formulário de impressão de relatórios está apresentado no Quadro 11.

```
<h:panelGroup>
    <p:fieldset legend="Parâmetros do Relatório">
      <h:panelGrid columns="2">
        <h:outputLabel value="Defeito" />
        <p:inputText value="#{defeitoBean.defeito.titulo}"
required="true" requiredMessage="O campo 'Defeito' não pode estar vazio.
Informe ao menos o caractere % para uma consulta em todos os registros."
/>
        <h:outputLabel value="Prioridade" />
        <p:inputText value="#{defeitoBean.defeito.prioridade}"
required="true" requiredMessage="O campo 'Prioridade' não pode estar
vazio. Informe ao menos o caractere % para uma consulta em todos os
registros." />
        <h:outputLabel value="Severidade" />
        <p:inputText value="#{defeitoBean.defeito.severidade}"
required="true" requiredMessage="O campo 'Severidade' não pode estar
vazio. Informe ao menos o caractere % para uma consulta em todos os
registros." />
        <h:outputLabel value="Situação" />
        <p:inputText value="#{defeitoBean.defeito.situacao}"
required="true" requiredMessage="O campo 'Situação' não pode estar
vazio. Informe ao menos o caractere % para uma consulta em todos os
registros." />
        <p:commandButton ajax="false" value="Imprimir PDF"
update="edicao, listagem, relatorio">
          <f:setPropertyActionListener
target="#{defeitoBean.tipoRelatorio}" value="1" />
          <p:fileDownload value="#{defeitoBean.arquivoRetorno}" />
        </p:commandButton>
      </h:panelGrid>
    </p:fieldset>
  </h:panelGroup>
```

Quadro 11 – Formulário de parâmetros para relatórios

O resultado da pesquisa apresentado na Figura 20 é um arquivo *portable document format* (PDF) como pode ser observado na figura do modelo a seguir.



Defeitos

GesDef
Apoiando na Gestão de Defeitos

Código	Defeito	Prioridade	Severidade	Situação	Teste	Produto	Versão
3	Def FURB	Baixa	Moderado	Corrigido	Teste FURB	Produto FURB	V. 1.1. Furb
2	Falha na abertura do cadastro de teste	Alta	Crítico	Fechado	Posição do cadastro de casos de uso na tela	GesDef Web	1.0.0.0
1	URL não é exibida corretamente no browser	Baixa	Moderado	Aberto	Verificar URL no browser	GesDef Web	1.0.0.0

Figura 21 – Relatório de defeitos

Voltando a observar a Figura 20 é importante informar que há duas formas de informar os parâmetros a serem utilizados para filtrar o relatório. A primeira é informando exatamente o que se deseja buscar do banco. No caso do relatório de defeitos isso é equivalente a informar o título exato de um dos defeitos existentes na base de dados além dos demais parâmetros. Isso é útil quando o resultado do relatório é conhecido, mas não prático quando se trata de campos com descrição longa como é o caso do campo título na tabela de defeitos. Dessa forma é interessante utilizar uma maneira mais simples de filtro. Caso não se saiba quais os títulos de defeitos existentes na base de dados ou mesmo se queria economizar tempo ao informar um campo longo, é possível informar parte do que se pretende pesquisar. Basta informar o caractere "%" antes e após a parte a ser pesquisada. Dessa forma, todos os registros no banco de dados que contiverem essa sequencia em seu título, serão trazidos na consulta.

3.5 RESULTADOS E DISCUSSÃO

Percebeu-se ao longo deste trabalho que o processo de verificação do MPS.BR é mais amplo que o escopo da gestão de defeitos. Porém fez-se uma análise dos resultados previstos no modelo que constam no Quadro 12.

Resultado	**Atende**	**Observação**
VER1	Parcialmente	Sugere-se a verificação de itens de trabalho como o plano de

		projeto, documento de requisitos, documento de análise, documentos de projeto e código-fonte. Destes apenas o código fonte é verificado com o auxílio da ferramenta desenvolvida.
VER2	Não	É sugerida a descrição de procedimentos, infraestrutura necessária e responsabilidades pelas atividades de verificação que não são cobertas pela ferramenta desenvolvida.
VER3	Não	Para alcançar esse resultado é necessário definir os critérios e procedimentos que serão utilizados para a verificação de cada produto de trabalho e na preparação do ambiente para verificação, disponibilizando ferramentas, recursos de hardware, infraestrutura de rede e outros recursos. A ferramenta também não contempla.
VER4	Parcialmente	O resultado esperado visa a garantia de que as atividades de verificação são executadas, o que inclui realização de revisão por pares e testes. A ferramenta contempla os testes necessários à esse processo.
VER5	Atende	Este resultado esperado tem o objetivo de garantir que os defeitos identificados durante a execução da verificação são documentados e registrados. A ferramenta desenvolvida permite o registro do defeito incluído dados como título, descrição, prioridade entre outros. Além do registro do defeito permite o registro do teste a ser executado para validar que a correção do defeito ocorreu sem problemas.
VER6	Parcialmente	Alcançar este resultado envolve realizar uma análise dos resultados obtidos em cada atividade de verificação e disponibilizar estes resultados para as partes interessadas. A ferramenta atende parcialmente este resultado esperado. Ela oferece alguns relatórios parametrizáveis para auxiliar na análise dos defeitos encontrados.

Quadro 12 – Resultados atendidos

A seguir apresenta-se o Quadro 13, que é um quadro comparativo com as principais características das ferramentas pesquisadas e o GesDef. Nele estão descritos os tipos de licença de cada ferramenta bem como a linguagem em que cada uma foi desenvolvida além dos bancos de dados em que elas operam e o tipo de interface com o usuário.

Rec. / Fer.	**Licença**	**Linguagem**	**Banco de Dados**	**Interfaces**
Bugzilla	MPL	Pearl	MySQL, Oracle, PostgreeSQL	*Web*, *e-mail*, linha de comando, GUI, Mylyn
JIRA	Proprietário/Livre para uso não comercial	Java	MySQL, PostgreSQL, Oracle, SQL Server	*Web*, *e-mail*, linha de comando, GUI, Mylyn
MantisBT	GPLv2	PHP	ADOdb (MySQL, PostgreSQL, MS SQL, etc)	*Web*, Mylyn
GesDef	Livre	Java, JSF	MySQL	*Web*

Quadro 13 – Comparativo de ferramentas

No Quadro 14 apresenta-se o comparativo das funcionalidades e diferenciais entre as ferramentas pesquisadas. Dentre as funcionalidades disponíveis nas ferramentas, o GesDef oferece dois recursos interessantes que não foram observados em nenhuma das outras ferramentas pesquisadas. É a possibilidade de criar a especificação para casos de uso e também de testes, o que permite rastrear os defeitos pelo agrupamento em relação ao teste executado.

Aplicativo	Campos Personalizáveis	Fluxo de Trabalho	Notificação de acompanhamento	Relatórios	Definição de teste	Definição de casos de uso
Mantis	Sim	Sim	Sim	Sim	Não	Não
Bugzilla	Sim	Sim	Sim	Sim	Não	Não
Scarab	Sim	Não encontrou-se dados a respeito	Sim	Não encontrou-se dados a respeito	Não	Não
GesDef	Não	Sim	Não	Sim	Sim	Sim

Quadro 14 – Funcionalidades e características

4 CONCLUSÕES

Ao término da implementação da ferramenta de apoio ao processo de gestão de defeitos, é possível verificar que se chegou a uma ferramenta de utilização bastante intuitiva. Um diferencial importante em relação a outras ferramentas é o fato de permitir a gestão dos defeitos e também dos testes que são realizados sobre esses erros. A maior parte das ferramentas pesquisadas contempla apenas os defeitos ou os testes.

Em relação aos objetivos previamente levantados, pode-se concluir que a ferramenta os atende adequadamente. Possibilita a sua utilização em empresas de pequeno porte. A aderência pretendida ao processo de verificação do modelo MPS.BR foi bastante limitada, porém as atividades relacionadas a gestão de defeitos foram atendidas.

Algumas das tecnologias utilizadas não foram inteiramente adequadas. Ao invés de JSP, utilizado no início do desenvolvimento, optou-se por substituir por *Java Server Faces* (JSF) pois oferece mais recursos e há bibliotecas que auxiliam no desenvolvimento. Esta é uma tecnologia mais nova e mais documentada. A utilização de diversas tecnologias diferentes também dificultou a implementação. Um número menor de tecnologias poderia ter facilitado e agilizado o desenvolvimento.

As dificuldades encontradas durante todo o processo de desenvolvimento foram de grande valia para o aprimoramento e crescimento pessoal.

Relacionando ao meio profissional, é possível afirmar que a ferramenta permitirá a substituição de planilhas eletrônicas utilizadas para controle de testes e defeitos. A ferramenta permite a padronização das informações e armazenamento em um único local o que diminui o risco de perda de informações.

4.1 EXTENSÕES

Como sugestões para trabalhos futuros recomenda-se:

a) incluir suporte ao envio de *e-mail* para que, por exemplo, usuários possam ser notificados do registro de um novo defeito;
b) permitir o anexo de arquivos ou imagens no registros de defeitos pois em casos específicos um arquivo de importação ou mesmo uma imagem podem ser úteis na

solução de um defeito;

c) incluir recurso para notificação de usuário para o recebimento de novos defeitos registrados para que os registros não fiquem sem tratamento por um período muito longo;
d) incluir permissões para os usuários de forma que apenas determinados recursos da ferramenta estejam disponíveis;
e) implementar outras atividades definidas no modelo MPS.BR para que a ferramenta fique mais adequada ao modelo.

REFERÊNCIAS BIBLIOGRÁFICAS

ATLASSIAN. **JIRA**: Bug and issue tracker. [S.L.], 2009. Disponível em: <http://www.atlassian.com/software/jira>. Acesso em: 12 jul. 2011.

BASTOS, Aderson et al. **Base de conhecimento em teste de software**.2. ed. São Paulo : Martins Fontes, 2007. 263 p, il.

BUGZILLA. **Bugzilla**. [S.L.], 2009. Disponível em: <http://www.bugzilla.org>. Acesso em: 12 jul. 2011.

CAETANO, Cristiano. **Gestão de defeitos**. [S.L.], 2007. Disponível em: < http://www.devmedia.com.br/articles/viewcomp.asp?comp=8036>. Acesso: 12 jul. 2011.

DIAS, André Felipe. **Comparação entre ferramentas de controle de mudança**. [S.L.], 2009. Disponível em: < http://www.pronus.eng.br/artigos_tutoriais/analises/ferramentasControleMudanca.php?pagNum=0>. Acesso: 20 out. 2011.

DELFIM, Samuel Martins. **Portal do arquiteto**: gestão de defeitos. [S.L.], 2008. Disponível em: <http://portalarquiteto.blogspot.com/2008/02/gesto-de-defeitos.html>. Acesso: 12 jul. 2011.

HETZEL, William; FLOW INFORMATICA. **Guia completo ao teste de software**. Rio de Janeiro : Campus, 1987. 206p, il.

MANTIS. **MantisBT**. [S.L.], 2009. Disponível em: <http://www.mantisbt.org>. Acesso em: 12 jul. 2011.

MOREIRA FILHO, Trayahú Rodrigues; RIOS, Emerson. **Projeto & engenharia de software**: teste de software. Rio de Janeiro : Alta Books, c2003. 210 p, il.

RIOS, Emerson. **Documentação de teste de software**: dissecando o padrão IEEE 829. Niterói: Imagem Art Studio, 2008. 88 p, il.

SOFTEX (a). **MPS.BR – Melhoria de Processo do Software Brasileiro**: Guia de aquisição. [S.L.], 2011. Disponível em: < http://www.softex.br/mpsbr/_guias/guias/MPS.BR_Guia_de_Aquisicao_2011.pdf>. Acesso em: 30 out. 2011.

SOFTEX (b). **MPS.BR – Melhoria de Processo do Software Brasileiro**: Guia de implementação – parte 4: fundamentação para implementação do nível D do MPS.BR. [S.L.], 2011. Disponível em: <http://www.softex.br/mpsbr/_guias/guias/MPS.BR_Guia_de_Implementacao_Parte_4_2011.pdf>. Acesso em: 12 jun. 2011.

SOFTEX (c). **MPS.BR – Melhoria de Processo do Software Brasileiro**: Guia geral. [S.L.], 2011. Disponível em: < http://www.softex.br/mpsbr/_guias/guias/MPS.BR_Guia_Geral_2011.pdf>. Acesso em: 12 jun. 2011.

SOFTONIC. **Enjoy Software**. [S.L.], 2011. Disponível em: <http://en.softonic.com>. Acesso em: 12 jul. 2011.

SCARAB. **Scarab**: *The self issue tracker*. [S.L.], 2011. Disponível em: < http://www.solitone.org/scarab/issues/curmodule/100/tqk/0/template/IssueList.vm/eventsubmit_dogetissues/foo/issuetype/100/action/Search>. Acesso em: 12 jul. 2011.

TIGRIS. **Scarab**: *Artifact tracking system*. [S.L.], 2011. Disponível em: < http://scarab.tigris.org>. Acesso em: 12 jul. 2011.

APÊNDICE A – Detalhamento dos casos de uso

Nesta seção está o detalhamento dos casos de uso previstos nos diagramas apresentados na seção 3.3.1.

No Quadro 15 apresenta-se o caso de uso “Efetuar *login*".

Caso de Uso – Efetuar *login* **Ator**: Usuário **Objetivo**: Permite que um usuário utilize o sistema mediante uma identificação positiva **Pré-condições**: A aplicação deve estar hospedada em um servidor de aplicações e o usuário deve estar cadastrado no banco de dados **Pós-condições**: O usuário acessou a aplicação **Cenário Principal**: **1**. O sistema apresenta tela solicitando o apelido e a senha do usuário **2**. O usuário preenche os campos e confirma **3**. O sistema valida o apelido e senha fornecidos **4**. O sistema apresenta a tela principal do sistema **Cenário Alternativo**: Se no passo 3 o usuário ou a senha não puderem ser validados, o sistema apresenta uma mensagem informado que os dados informados não estão corretos e retorna ao passo 2

Quadro 15 – Descrição do caso de uso Efetuar *login*

No Quadro 16 apresenta-se o caso de uso “Cadastrar usuários”.

Caso de Uso - Cadastrar usuários
Ator: Usuário
Objetivo: Cadastrar usuários para que tenham permissão de acesso a aplicação
Pré-condição: Usuário deve ter realizado *login* na aplicação
Pós-condição: Um novo usuário é cadastrado na base de dados
Cenário Principal:
1. O sistema apresenta tela solicitando os dados do usuário a ser cadastrado
2. O usuário informa os dados e pressiona o botão “Salvar”
3. O sistema verifica se os campos obrigatórios foram informados
4. O sistema grava o novo registro no banco de dados
5. O sistema deixa a tela pronta para a inserção de um novo registro
Cenário Alternativo:
No passo 2, se o usuário não informar dados no formulário e pressionar o botão “Salvar”, o sistema apresenta mensagens informando os campos obrigatórios que não foram informados.

Quadro 16 – Descrição do caso de uso Cadastrar usuários

No Quadro 17 apresenta-se o caso de uso “Visualizar defeitos por situação”.

Caso de Uso – Visualizar defeitos por situação

Ator: usuário

Objetivo: Permite a visualização de defeitos encontrados de acordo com a situação do defeito

Pré-condição: Devem existir defeitos cadastrados

Pós-condições: Um relatório contendo defeitos na situação informada no filtro é gerado

Cenário Principal:

1. O sistema apresenta formulário solicitando o título, prioridade, severidade e situação do defeito

2. O usuário informa o caractere “%” em cada um dos campos e pressiona o botão “Imprimir PDF”

3. O sistema gera arquivo PDF contendo a listagem de todos os defeitos registrados na base de dados

Cenário Alternativo:

No passo 2, caso o usuário informe a situação, “Aberto”, o sistema gera relatório contendo a listagem dos defeitos que ainda não foram corrigidos

Cenário Alternativo:

No passo 2, caso o usuário informe a situação, “Corrigido”, o sistema gera relatório contendo a listagem dos defeitos já corrigidos

Cenário Alternativo:

No passo 2, caso o usuário informe a situação, “Fechado”, o sistema gera relatório contendo a listagem dos defeitos já corrigidas

Cenário Alternativo;

No passo 2, caso o usuário não informa parâmetros para o relatório, o arquivo é gerado vazio

Quadro 17 – Descrição do caso de uso Visualizar defeitos por situação

No Quadro 18 apresenta-se o caso de uso “Visualizar defeitos por prioridade”.

Caso de Uso – Visualizar defeitos por prioridade

Ator: Usuário

Objetivo: Permite a visualização de defeitos encontrados de acordo com a prioridade de correção

Pré-condição: Devem existir defeitos cadastrados

Pós-condição: Um relatório no formato PDF é gerado

Cenário Principal:

1. O sistema apresenta formulário solicitando o título, prioridade, severidade e situação do defeito

2. O usuário informa o caractere “%” em cada um dos campos e pressiona o botão “Imprimir PDF”

3. O sistema gera arquivo PDF contendo a listagem de todos os defeitos registrados na base de dados

Cenário Alternativo:

No passo 2, caso o usuário informe a prioridade, “Alta”, o sistema gera relatório contendo a listagem dos defeitos que precisam ser corrigido imediatamente

Cenário Alternativo:

No passo 2, caso o usuário informe a prioridade, “Média”, o sistema gera relatório contendo a listagem dos defeitos graves mas que não impedem o funcionamento da aplicação

Cenário Alternativo:

No passo 2, caso o usuário informe a prioridade, “Baixa”, o sistema gera relatório contendo a listagem dos defeitos que não causam grandes impactos negativos nos resultados esperados

Cenário Alternativo;

No passo 2, caso o usuário não informa parâmetros para o relatório, o arquivo é gerado vazio

Quadro 18 – Descrição do caso de uso Visualizar defeitos por prioridade

No Quadro 19 apresenta-se o caso de uso “Cadastrar testes”.

Caso de Uso – Cadastrar testes

Ator: Usuário

Objetivo: Permite o cadastro dos testes que serão executados no ciclo de testes

Pré-condição: O produto e o caso de uso ao qual o teste pertence devem estar previamente cadastrados

Pós-condição: Um novo teste é cadastrado no sistema

Cenário Principal:

1. O sistema apresenta a tela de cadastro de testes e disponibiliza caixas de seleção onde o usuário escolhe o nome do produto e o caso de uso para qual o teste será cadastrado

2. O usuário informa o produto e o caso de uso

3. O usuário informa um título e a descrição do teste

4. O usuário informa a prioridade de execução (alta, média, baixa)

5. O usuário informa o status (pendente, testando, realizado)

6. O usuário informa o resultado do teste (OK, NOK)

7. O usuário confirma a inclusão do registro

8. O sistema armazena as informações no banco de dados

9. O sistema limpa os campos ficando preparado para uma nova inclusão

Cenário Alternativo:

No passo 2, após o usuário informar o produto:

2.1. O usuário poderá selecionar um teste cadastrado

2.2. O usuário poderá excluir o teste

2.3. O sistema verifica se este teste possui algum defeito cadastrada

2.4. Se possuir defeitos cadastrados, o sistema emite mensagem (Teste não pode ser excluído, pois possui defeitos cadastrados!)

2.5. O sistema vai para o passo 9

Quadro 19 – Descrição do caso de uso Cadastrar testes

No Quadro 20 apresenta-se o caso de uso “Cadastrar defeitos”.

Caso de Uso – Cadastrar defeitos

Ator: Usuário

Objetivo: Permite a manutenção do cadastro de defeitos referente aos problemas encontrados nos produtos testados

Pós-condição: Um novo defeito é incluído no banco de dados

Pré-condição: Um produto precisa estar cadastrado no sistema

Cenário Principal:

1. O sistema apresenta tela solicitando que seja selecionado um teste

2. O usuário informar o título, descrição, prioridade de correção, severidade e situação

3. O usuário confirma a inclusão pressionando o botão “Salvar”

4. O sistema salva os dados no banco de dados e libera a tela para inserir um novo defeito

Cenário Alternativo:

No passo 2, se o usuário estiver alterando o registro de defeito, pode informar a causa do defeito e a solução dada ao defeito.

Quadro 20 – Descrição do caso de uso Cadastrar defeitos

No Quadro 21 apresenta-se o caso de uso “Cadastrar produtos”.

Caso de Uso – Cadastrar produtos

Ator: Usuário

Objetivo: Permite a manutenção do cadastro dos produtos testados

Pré-condição: Um produto deve estar cadastrado no sistema.

Pós-condição: Um novo produto deve ser criado no banco de dados.

Cenário Principal:

1. O sistema apresenta tela solicitando que sejam informados um nome de produto e o setor ao qual o produto pertence

2. O usuário informa um nome e o setor

3. O usuário confirma a inclusão pressionando o botão “Salvar”

4. O sistema grava o novo registro no banco de dados

Cenário Alternativo:

No passo 3, se o usuário perceber que salvou o registro com nome errado:

3.1. Seleciona o ícone de alteração disponível na listagem de produtos

3.2. Altera o nome do produto

3.3. Pressiona o botão “Salvar” novamente

Quadro 21 – Descrição do caso de uso Cadastrar produtos

No Quadro 22 apresenta-se o caso de uso “Alterar a situação do defeito”.

Caso de Uso – Alterar a situação do defeito
Ator: Usuário
Objetivo: Permite a alteração da situação de um defeito de aberto para, corrigido ou fechado
Pré-condição: Deve haver defeitos cadastrados no sistema
Pós-condição: Situação do defeito alterada
Cenário Principal:
1. O sistema apresenta uma tela solicitando que o desenvolvedor informe o defeito a ser alterado
2. O desenvolvedor informa o defeito a ser alterado e altera a situação do defeito de aberto para corrigido e confirma a alteração
3. O sistema grava as alterações no banco de dados
4. O sistema limpa os campo do formulário e fica esperando de uma nova ação do usuário
Cenário Alternativo:
No passo 1, se o testador verificar que o defeito foi corrigido, ele pode realizar as atividades de teste e então alterar a situação do defeito de corrigido para fechado
Cenário Alternativo:
No passo 1, se o testador verificar que o defeito ainda ocorre, ele pode alterar a situação de corrigido para aberto

Quadro 22 – Descrição do caso de uso Alterar situação do defeito

No Quadro 23 apresenta-se o caso de uso “Alterar a prioridade do defeito”.

Caso de Uso – Alterar a prioridade do defeito
Ator: Usuário
Objetivo: Permite a alteração entre os níveis de prioridade de um defeito.
Pré-condição: Deve haver defeitos cadastrados no sistema.
Pós-condição: A prioridade de um defeito é alterada
Cenário Principal:
1. Após o cadastro do defeito o mesmo é analisado pelo desenvolvedor e caso seja necessário, ele pode alterar o nível de prioridade de correção do defeito.

Quadro 23 – Descrição do caso de uso Alterar situação do defeito

No Quadro 24 apresenta-se o caso de uso “Alterar a severidade do defeito”.

Caso de Uso – Alterar a severidade do defeito
Ator: Usuário
Objetivo: Permite a alteração de severidade de um defeito entre alta, média e baixa
Pré-condição: Deve haver defeitos cadastrados no sistema
Pós-condição: A severidade de um defeito alterada
Cenário Principal:
1. Após o cadastro do defeito o mesmo é analisado pelo desenvolvedor e caso seja necessário, ele pode alterar a severidade do defeito de acordo com o resultado de sua análise.

Quadro 24 – Descrição do caso de uso Alterar severidade do defeito

APÊNDICE B – Dicionário de dados

O dicionário de dados é apresentado nos quadros a seguir:

Tabela usuario_permissao		
Nome	**Tipo**	**Descrição**
usuario	INT (11)	Chave primário auto-incremental
permissao	VARCHAR (50)	Tipo de usuário administrador ou comum

Quadro 25 – Tabela usuario_permissao

Tabela usuario		
Nome	**Tipo**	**Descrição**
codigo	INT (11)	Chave primário auto-incremental
ativo	BIT (1)	Ativo ou inativo
celular	VARCHAR (14)	Número do telefone celular
email	VARCHAR (50)	Endereço eletrônico
idioma	VARCHAR (15)	Idioma português ou inglês
login	VARCHAR (15)	*Login* para acesso a aplicação
nascimento	DATE	Data de nascimento
nome	VARCHAR (100)	Descrição do nome
papel	VARCHAR (30)	Papel desempenhado no setor de trabalho
senha	VARCHAR (10)	Senha para acesso a aplicação
setor	INT (11)	Chave estrangeira para tabela setor

Quadro 26 – Tabela usuario

Tabela setor		
Nome	**Tipo**	**Descrição**
codigo	INT (11)	Chave primário auto-incremental
Nome	VARCHAR (50)	Descrição do nome

Quadro 27 – Tabela setor

Tabela produto		
Nome	**Tipo**	**Descrição**
codigo	INT (11)	Chave primário auto-incremental
nome	VARCHAR (255)	Descrição do nome
setor	INT (11)	Chave estrangeira para tabela setor

Quadro 28 – Tabela produto

Tabela versao		
Nome	**Tipo**	**Descrição**
codigo	INT (11)	Chave primário auto-incremental
inicio	DATE	Data de início do ciclo de testes
nome	VARCHAR (15)	Descrição curta da versão
termino	DATE	Data de término do ciclo de testes
produto	INT (11)	Chave estrangeira para a tabela produto

Quadro 29 – Tabela versão

Tabela teste		
Nome	**Tipo**	**Descrição**
codigo	INT (11)	Chave primário auto-incremental
descricao	VARCHAR (255)	Descrição do teste
prioridade	VARCHAR (255)	Descrição da prioridade de execução do teste
resultado	VARCHAR (255)	Descrição do resultado da execução do teste
status	VARCHAR (255)	Descrição do estado de execução do teste
titulo	VARCHAR (255)	Descrição curta do teste
casouso	INT (11)	Chave estrangeira para a tabela casouso
produto	INT (11)	Chave estrangeira para a tabela produto

Quadro 30 – Tabela teste

Tabela casouso		
Nome	**Tipo**	**Descrição**
codigo	INT (11)	Chave primário auto-incremental
descricao	VARCHAR (255)	Descrição do teste
prioridade	VARCHAR (255)	Descrição da prioridade de execução do teste
resultado	VARCHAR (255)	Descrição do resultado da execução do teste
status	VARCHAR (255)	Descrição do estado de execução do teste
titulo	VARCHAR (255)	Descrição curta do teste

casouso	INT (11)	Chave estrangeira para a tabela casouso
produto	INT (11)	Chave estrangeira para a tabela produto

Quadro 31 – Tabela casouso

Tabela defeito		
Nome	**Tipo**	**Descrição**
codigo	INT (11)	Chave primário auto-incremental
causa	VARCHAR (2000)	Descrição da causa do defeito
descricao	VARCHAR (2000)	Descrição detalhada do defeito
prioridade	VARCHAR (5)	Prioridade de correção do defeito
severidade	VARCHAR (8)	Descrição da severidade do defeito
situacao	VARCHAR (9)	Descrição da situação de correção do defeito
solucao	VARCHAR (2000)	Descrição da solução dada ao defeito
titulo	VARCHAR (100)	Descrição curta do defeito
teste	INT (11)	Chave estrangeira para a tabela teste

Quadro 32 – Tabela defeito