A GESTÃO DE DEFEITOS NA REALIZAÇÃO DO TESTE DE *SOFTWARE*

https://doi.org/10.5281/zenodo.15566578

ROMANO, Simone Maria Viana, Especialista*

* Faculdade de Tecnologia de Praia Grande
Departamento de Análise e Desenvolvimento de Sistemas
Pça. 19 de Janeiro, 144, Boqueirão, Praia Grande / SP, CEP: 11700-100
Fone (13) 3591-1303
simone@fatecpg.com.br

RESUMO

Este artigo tem o objetivo de explicar como a atividade de teste de *software* no ciclo de vida do sistema, pode auxiliar na obtenção da qualidade do produto. Para isto, uma das etapas importantes realizadas neste teste é a gestão dos defeitos encontrados com uso do *software open-source* chamado Mantis. Este *software* auxilia o líder do projeto ou de equipe no gerenciamento adequado dos problemas, pois tem uma gestão de configuração, controle de mudanças e visualização de relatórios estatísticos. Assim, o profissional que foi encarregado de resolver o problema, tem onde consultar o histórico dos defeitos semelhantes encontrados anteriormente e as soluções usadas na resolução destes problemas e ainda possa registrar novos problemas com as soluções tomadas para auxiliar na resolução dos defeitos futuros, criando assim uma base de conhecimento do *software.*

PALAVRAS-CHAVE: teste de *software*, qualidade de *software*, gestão de defeitos, *software* Mantis.

ABSTRACT

This article aims to explain how the activity of software testing life cycle of the system, can assist in achieving product quality. For this, one of the important steps taken in this test is the management of defects found

with the use of open-source software called Mantis. This software helps the project leader or team in the proper management of problems, it has a configuration management, change control and visualization of statistical reports. Thus, the professional who has been tasked to solve the problem, you see where the history of similar defects previously found and the solutions used in solving these problems and still be able to register new problems with the solutions adopted to assist in the resolution of future defects, thus creating a knowledge base software.

KEYWORDS: *software testing, software quality, defect management, software Mantis.*

INTRODUÇÃO

Pressman (2005) afirma que a qualidade do *software* é a conformidade com os requisitos funcionais e não funcionais que têm sido explicitamente declarados; padrões de desenvolvimento que tenham sido claramente documentados e características implicitamente esperadas de todo *software* a ser desenvolvido.

Ao longo dos anos, muitas tentativas foram realizadas para o aperfeiçoamento das técnicas no desenvolvimento de sistemas e em especial, na fase de teste de *software.* Pois é nesta fase que inicia o encarecimento do projeto, pois se os testes não forem realizados corretamente, o sistema não atenderá as expectativas do cliente e se transformará em um transtorno dentro da empresa, pois o *software* ficará em manutenção ao invés de atender e agilizar o negócio.

Koomen (1999) estima que entre 25% e 50% dos custos e esforços de um projeto são gastos com testes. E para resolver este problema, podemos efetuar a gestão de defeitos.

Segundo o IEEE *Institute of Electrical and Engineers* (610, 1990), defeito é um ato inconsistente cometido por um indivíduo ao tentar entender uma determinada informação, resolver um problema ou utilizar um método ou uma ferramenta.

Para garantir os benefícios como custo, tempo, controle, dentre outros no momento que o sistema foi testado, a gestão adequada dos defeitos encontrados é uma das formas de amenizar ou sanar

definitivamente estes problemas identificados.

Esta gestão de defeitos, de acordo com o livro "Base do Conhecimento do CSTE" do QAI, possui elementos importantes como: prevenção de defeitos, linha de base "entregável", identificação do defeito, melhoria do processo e relatório de gestão, podem promover o gerenciamento adequado dos problemas encontrados, pois estes poderão ser identificados e armazenados em um repositório para consultas futuras. Uma das diversas soluções para a gestão dos defeitos é a ferramenta Mantis, que será apresentado, por ser open-source vem crescendo no mercado colaborativo.

1 REVISÃO DA LITERATURA

Em 1995, *Standish Group* realizou um estudo com 350 empresas que tinha aproximadamente 8000 projetos e mostrou que somente 16,2% dos projetos atingem o sucesso no término do projeto e de acordo com o custo estabelecido; 52,7% dos projetos são classificados como problemáticos.

1.1 REQUISITOS

Requisitos são características que um produto deve possuir. De acordo com estas características, os requisitos podem ser:

> [...] Características funcionais, que representam os comportamentos que um programa ou sistema deve apresentar diante de certas ações de seus usuários;
> Características não funcionais, que quantificam determinados aspectos do comportamento (PAULA, 2005, p. 5).

Os requisitos funcionais na especificação de requisitos podem ser classificados como:

> [...] Os requisitos explícitos são aqueles descritos em um documento que arrola os requisitos de um produto, ou seja, um documento de especificação de requisitos.
> Os requisitos normativos são aqueles que decorrem de

> leis, regulamentos, padrões e outros tipos de normas a que o tipo de produto deve obedecer.
> Os requisitos implícitos são expectativas dos clientes e usuários, que são cobradas por esses, embora não documentadas (PAULA, 2005, p. 5).

Segundo Ávila (2010), analisar os requisitos é um dos custos mais baixos, mas, também Fé nesta etapa que muitos defeitos são inseridos. É possível analisar ainda o que é menos custoso e corrigir estes defeitos nesta fase. Estes defeitos não são analisados no momento correto e com isto também há um aumento no valor do projeto. Um requisito mal definido ou interpretado é considerado um defeito.

1.2 TESTES DE *SOFTWARE*

Dias (2007), conceitua teste de *software* como o processo executado em produto. No caso do *software*, é usado para verificar as funcionalidades especificadas no levantamento de requisitos. Esta atividade é usada para encontrar as falhas e identificar as causas para que possam ser resolvidas antes da entrega do produto. Testar um *software* pode ser entendido como uma maneira de verificar se o funcionamento está de acordo com o esperado.

O teste de *software* inicia-se após a codificação do sistema com estes objetivos:

> [...] O processo de realização de testes concentra-se nos aspectos lógicos internos do *software*, garantindo que todas as instruções tenham sido testadas, e concentram-se também nos aspectos funcionais externos, ou seja, realizando testes para descobrir erros e garantir que a entrada definida produza resultados reais que concordem com os resultados exigidos [...] (PRESSMAN, 1995, p. 34).

Pressman (1995) ainda diz-se que as etapas na fase de testes são tão preocupantes como a codificação do sistema, pois pode também acrescentar erros e ao corrigir este erro na fase de manutenção, o custo é de 60 vezes maior se compararmos a fase do desenvolvimento.

Para a realização dos testes seria interessante chamar uma

pessoa que não tenha participado do desenvolvimento do *software*, pois poderá identificar os problemas sem estar envolvido com o produto.

1.2.1 Principais conceitos pertinentes ao teste de *software*

O *Institute of Electrical and Electronics Engineers* (IEEE 610, 1990), apresentou uma forma de diferenciar os conceitos de falha, erro e defeito. Onde **falha** é a forma do *software* agir diferente do que o usuário estava esperando; **erro** é a concretização de um defeito num artefato de *software*. Ou seja, é o que foi encontrado entre o valor esperado e o valor entregue; por último, **defeito** que nada mais é que a ação efetuada para resolver um problema ou utilizar uma ferramenta ou método, por exemplo, conforme figura 1, uma instrução de programação.



Figura 1: Diferença entre falha, erro e defeito

Fonte: Disponível em: <http://www.devmedia.com.br/post-8035-Artigo-Engenharia-de-Software-Introducao-a-Teste-de-Software.html> Acesso em: 29 jan. 2012

Relacionado à cobertura dos testes tem-se:

> Um objeto central de toda a metodologia dos testes é maximizar a sua cobertura, ou seja, a quantidade potencial de defeitos que podem ser detectados, por meio de teste. Deseja-se conseguir detectar a maior quantidade possível de defeitos que não são apanhados pelas revisões, dentro de dados limites de custo e prazos (PAULA, 2010, p. 351).

1.2.2 Defeitos

De acordo com Paula (2010, p.461), "um defeito, segundo PMBOK, é uma imperfeição ou deficiência em um componente do projeto na qual esse componente não atende aos seus requisitos ou especificações e precisa ser reparado ou substituído".

Segundo, Sommerville (2007), os defeitos de um sistema após a sua resolução, não melhora a confiabilidade no *software*:

> Mills et al. (1987) descobriram que a remoção de 60% dos erros conhecidos em seu *software* levou a uma melhoria de confiabilidade em 3%. Adams (1984), em um estudo de produtos de *software* da IBM, observou que muitos defeitos nos produtos só eram passíveis de causar falhas após centenas ou milhares de meses de uso do produto (SOMMERVILLE, 2007, p.36).

Muitos usuários alteram a sua maneira de trabalhar evitando entradas que possam causar falhas e os usuários com mais experiência não utilizam os recursos pouco confiáveis; com isto uma falha pode não chegar a ser um defeito.

De acordo, com Caetano (2007), para o desenvolvimento de *software*, os defeitos, apesar do uso de ferramentas e métodos, ocorrem através das pessoas, por isto, testar o *software* é muito importante, pois é a última etapa antes da entrega do produto ao cliente. Como aumento dos defeitos, podemos citar o tamanho do projeto e o número de pessoas envolvidas. Dizer que um programa falhou é, simplesmente, dizer que não está de acordo com os requisitos estabelecidos.

Caetano (2007) diz ainda que os defeitos aparecem em diversas fases do ciclo de desenvolvimento de *software*, onde em cada fase ocorre uma mudança nas informações e, portanto, gerou a necessidade de em cada fase efetuar testes, conforme a figura 2:

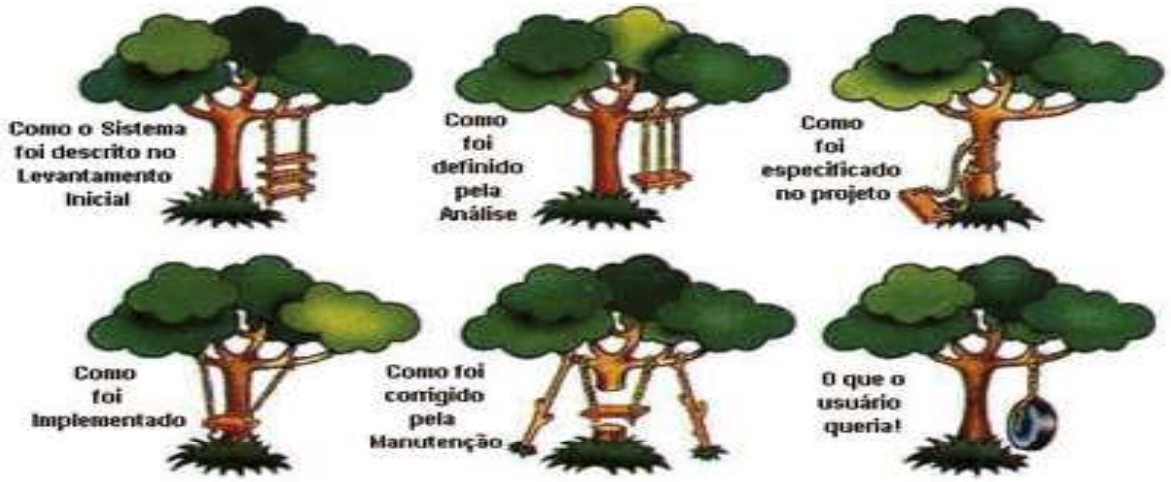


Figura 2: Interpretação dos requisitos no desenvolvimento do *software*
Fonte. Disponível em: <http://www.projectcartoon.com/cartoon/611> Acesso em: 02 fev. 2012

1.2.2.1 Tipos de defeitos

Segundo Pádua (2010), há os seguintes tipos de defeitos:

a) faltante: falta total ou parcial de um requisito;
b) errado: requisito foi implementado de forma incorreta;
c) acréscimo: elemento ou comportamento foi implementado, mas não está presente da especificação de requisitos.

Os testes realizados contribuem para a qualidade do *software*:

> [...] Os testes são indicadores da qualidade de um produto, mas do que meios de detecção e correção de erros. Quanto maior o número de defeitos detectados em um *software*, provavelmente, maior também o número de defeitos não detectados. A ocorrência de um número anormal de defeitos em uma bateria de testes indica uma provável de redesenho dos itens testados (PAULA, 2005, p. 184).

Os defeitos de um *software* geralmente são humanos e quanto maior o projeto e quanto mais pessoas estiverem envolvidas, mais a chance de ter defeitos aumenta. Por isto é importante à execução de testes.

1.3. QUALIDADE DE *SOFTWARE*

Há diversas definições de qualidade de *software*. Define-se qualidade de *software* como:

> Conformidade a requisitos funcionais e de desempenho explicitamente declarados, a padrões de desenvolvimento claramente documentados e a características implícitas que são esperadas de todo *software* profissionalmente desenvolvido (PRESSMAN, 1995, p. 724).

Pressman (1995) diz ainda que a qualidade de *software* é a junção de diversos fatores que de acordo com aplicações e solicitações de clientes distintos sofrerão variações.

Para adquirir qualidade o *software* precisa atender os seguintes fatores:

> Corretitude: à medida que um programa satisfaz sua especificação e cumpre os objetivos visados pelo cliente;
> Confiabilidade: à medida que se pode esperar que um programa execute sua função pretendida com a precisão exigida;
> Eficiência: a quantidade de recursos de computação e de código exigida para que um programa execute sua função;
> Integridade: à medida que o acesso ao *software* ou a dados por pessoas não autorizadas pode ser controlado;
> Usabilidade: o esforço para aprender, operar, preparar a entrada e interpretar a saída de um programa;
> Manutenabilidade: o esforço exigido para localizar e reparar erros num programa;
> Flexibilidade: o esforço exigido para modificar um programa operacional;
> Testabilidade: o esforço exigido para testar um programa a fim de garantir que ele execute sua função pretendida;
> Portabilidade: o esforço exigido para transferir o programa de um ambiente de sistema de hardware e/ou *software* para outro;

> Reusalidade: à medida que um programa (ou partes de um programa) pode ser reusado em outras aplicações – relacionada ao empacotamento e escopo das funções que o programa executa.
> Interoperabilidade: o esforço exigido para se acoplar um sistema a outro (PRESSMAN, 1995, p. 726).

Segundo Kalinowski (2007), para alcançar a qualidade de *software* através do aperfeiçoamento do seu processo de produção, os defeitos dos seus artefatos precisam ser localizados através de uma análise causal, que se mostra de forma eficiente na melhoria dos processos de *software* que é o nosso produto. Analisar os defeitos de forma causal traz melhoria nos processos das empresas elevando o seu nível de maturidade através da diminuição dos defeitos no produto e com aumento do retorno de investimento (ROI) trazendo vantagem competitiva das empresas de *software*.

1.3.1 Qualidade Garantida

Segundo Paula (2005), é incorreto substituir prazo e/ou custo por qualidade. Qualidade é consequência das pessoas, processos e tecnologia. Os defeitos são inseridos em todas as fases do desenvolvimento do sistema e acontecem por causa das limitações humanas, erros de interpretação e lógica, desconhecimento de técnicas, falta de atenção ou ainda falta de motivação.

Para ter qualidade garantida, as etapas retiram parte dos defeitos que foram inseridos, porém, quando as etapas não são efetuadas corretamente deixa-se de serem retirados os defeitos:

> Defeitos que não são removidos precocemente acabam sendo detectados depois. Quando mais tarde um defeito é corrigido mais cara é a sua correção, por várias razões que serão discutidas posteriormente. O pior caso acontece quando o defeito chega ao produto final. Nesse caso, ele só será removido através de uma operação de manutenção. Essa é a forma mais cara de remoção de defeitos. Em certos casos, como acontece em sistemas de missão crítica, defeitos de *software* podem trazer prejuízos irreparáveis (PAULA, 2005, p. 8).

1.3.2 Análise causal dos defeitos

Segundo *Software Engineering Institute* (SEI, 2006), a análise causal dos defeitos auxilia para que os objetivos de produtividade e qualidade dos projetos sejam alcançados.

Para Card (2005), é importante também aprender e gerenciar a partir dos defeitos do *software* e para isto resumiu a análise causal dos defeitos em seis etapas que podem ser efetuadas como parte do processo de prevenção de defeitos:

> a) Identificar a amostragem do problema, por exemplo, através de um controle estatístico de processos;
> b) Classificar os processos selecionados, através do tipo de defeito, o momento ou fase de desenvolvimento que ocorreu este defeito e o que foi feito para extrair este defeito;
> c) Diagnosticar os erros sistemáticos que são aqueles que surgem através da introdução de tipos de defeitos similares em diferentes ocasiões. Estes erros estão relacionados a uma atividade específica ou parte de um produto;
> d) Encontrar as principais causas, pois vários fatores poderiam gerar um erro sistêmico;
> e) Criar propostas de ação para evitar ocorrência em projetos futuros através das principais causas encontradas;
> f) Registrar através de documentos os resultados da reunião para que as ações sejam efetuadas. (CARD, 2005)

Robitaille (2004) diz que é importante que a análise causal seja feita como uma forma de encontrar os ativos dos processos organizacionais.

Shull (1998) classificou os defeitos em ambigüidade, fato erro e informação estranha ou inconsistente.

1.3.3 Gestão da Qualidade

Temos as seguintes etapas na gestão da qualidade: planejamento da qualidade, verificação, validação, auditoria e a resolução de defeitos

(esta tem as tarefas de submissão, análise, remoção e verificação dos defeitos encontrados no sistema).

1.4. GESTÃO DE DEFEITOS

O objetivo principal do teste de *software* é mensurar o nível de qualidade de um sistema e podemos mensurar esta qualidade através da quantidade de erros encontrados na realização dos testes.

No padrão IEEE (1990), sugere um processo de gestão de defeitos que possui o objetivo de definir práticas para prevenir os defeitos e diminuir os riscos de um projeto.

1.4.1 Princípios para gestão de defeitos

O objetivo principal da gestão de feitos é não causar defeitos e para que isto ocorra é necessário diminuir os riscos do projeto teste e também do projeto de desenvolvimento com o processo automatizado facilita muito o gerenciamento através da melhoria contínua.

A ferramenta de gestão de defeitos, também auxilia na integração entre testadores e desenvolvedores do sistema.

Após a localização do defeito é importante documentar este defeito através de algum mecanismo definido na gestão de defeitos que poderá ser uma planilha ou até uma ferramenta automatizada. Após a identificação do defeito este deverá ser submetido novamente ao ciclo de vida definido na gestão de defeitos até a sua finalização.

1.4.2 Documentação dos defeitos

Segundo Cristalli (2007), a documentação de um defeito é muito importante para o processo de gestão de defeitos, mas nem sempre é dada a devida atenção e para isto abaixo temos dicas importantes para a elaboração deste documento:

> a) Evidenciar: deixar claro a existência do defeito através de arquivos de saída, etc.;
> b) Generalizar: compreender o problema de genericamente, pois o mesmo pode ocorrer em outras situações;
> c) Neutralizar: descrever os fatos evitando opiniões

alheias;

d) Precisão: o defeito encontrado deve ser um desvio de comportamento esperado e não falha por entendimento;

e) Reproduzir: garantir que o defeito possa ser reproduzido através de etapas;

f) Revisar: descrever os passos para reproduzir o defeito, pois a documentação do defeito também é um documento relevante para o projeto;

g) Resumir: descrever o defeito de forma sucinta, porém objetiva (CRISTALLI, 2007).

1.4.3 Resolvendo Defeitos

Os defeitos podem ser classificados por meio de critérios e classificação que são formas de ser feito um critério de teste que também é um tipo de critério de aprovação. Nem todas as anomalias são chamadas de defeitos, mas que são confirmadas como defeitos precisam ser removidas, mesmo que às vezes a remoção possa ser demorada. Os que são localizados na apreciação devem ser removidos em um prazo pequeno. Este processamento dos defeitos é chamado de resolução de defeitos é realizada conforme ocorre o problema e é uma das etapas da gestão de qualidade.

Após a identificação dos defeitos, algumas atividades devem ser executadas para que os defeitos sejam removidos. Esta etapa não faz parte do projeto e, portanto não foi devidamente planejada. Cada defeito após a resolução deve ser finalizado e nada impede que possa ser inicializado novamente.

1.5 FERRAMENTAS PARA GESTÃO DE DEFEITOS

As ferramentas para gestão de defeitos ou também chamadas de *bug tracker*[1], segundo Roessler (2010)[2], são usadas para coleta\r informações sobre o defeito e cadastrar na ferramenta de gestão para que outras pessoas da equipe possam ter acesso e também para ser

1 Bug Tracker significa acompanhamento de erros.

2 Disponível em <http://testersoftware.blogspot.com/2010/10/ferramenta-para-gestao-de-defeito.html> acesso em 07 fev. 2012).

criada uma base de conhecimento para resolução de problemas futuros.

Segundo Paula (2010), geralmente uma ferramenta para a resolução dos defeitos possuem um modo cliente ou normal para a resolução dos defeitos. O registro do defeito fica armazenado em um banco de dados relacional e os processos da ferramenta incluem visualizar relatórios, pesquisar informações encontradas na resolução dos defeitos e até enviar mensagens através do email para os profissionais envolvidos.

Há diversas ferramentas – pagas ou *open-source*[3] no mercado que possuem gestão de defeitos, dentre elas podemos destacar: Bugzilla, Jira, Scarab, BugNET, TRAC e o Mantis, tema deste artigo.

1.5.1 Mantis

A ferramenta *open-source* Mantis, desenvolvida em PHP, está sob os termos da licença GPL (*General Public License*), pode ser encontrado no site http://www.mantis.org, tem como objetivo controlar a gestão de defeitos através da gestão do ciclo de vida de um defeito que inicia-se na notificação do defeito até o momento que é finalizado através de *workflows*[4].

A ferramenta Mantis foi desenvolvida para ser usada na WEB, através do browser, utiliza-se de banco de dados MySQL, DB2, Sql Server ou PostgreSQL e pode ser instalado no sistema operacional Linux, Windows, Mac OS, entre outros. Após a instalação é possível escolher o idioma português.

De acordo com Silva (2009), é um aplicativo gratuito desde novembro de 2000 e sua função é registrar incidentes ou mudanças durante a manutenção do *software*. Gerencia a configuração de sistema, gera e acompanha estatísticas das mudanças. A alteração pode ser registrada e enviada para as pessoas envolvidas no desenvolvimento do sistema.

De acordo com Delfim (2008)[5], a ferramenta *open source* Mantis possui várias funcionalidades, dentre elas destacam-se:

3 Código aberto.

4 Fluxos.

5 Disponível em <http://portalarquiteto.blogspot.com/2008/02/gesto-de-defeitos.html> acessado em 07/02/2012.

- Pode ser executado em qualquer plataforma;
- Suporta vários bancos de dados;
- Suporta múltiplos mecanismos de autenticação;
- Ciclo de vida personalizável;
- Gerador interno de relatório de defeitos;
- Controle de acesso e permissão por usuário;
- Mecanismo para a criação de campos personalizáveis;
- Notificação por meio de e-mails automáticos;
- Integração com ferramentas de controle de versão;
- Interface *webservices* para integração com outras ferramentas.

1.5.1.1 Relatório de Gestão de Defeitos

Há uma norma que define como deverá ser feito o relatório para gerir os defeitos bem como a forma de mensurar os defeitos:

> A norma IEEE Std 829-1998 (IEEE Standard for *Software Test Documentation*) define a documentação e relatórios necessários para a execução de um projeto de teste de *software*. Dentre os relatórios sugeridos, existe um relatório chamado "Relatório de Incidente de Teste". Este relatório registra e consolida as ocorrências que precisam de algum tipo de investigação. Ele é normalmente utilizado para registrar os defeitos encontrados durante a execução dos testes (CAETANO, 2007).

É possível através de relatórios estatísticos que podem ser usados pelo editor de texto *Microsoft Word* ou *Microsoft Excel* para visualizar as alterações através de relatórios sintéticos e relatórios analíticos com estatística contendo os números sobre as mudanças atribuídas e o tempo gasto na resolução (Figura 3).

Resumo

Por Projeto	Aberto	Resolvido	Fechado	Total
Sistema de Matricula	2	2	0	4
Sistema	1	0	0	1

Por Status	Aberto	Resolvido	Fechado	Total
novo	1	0	0	1
admitido	1	0	0	1
atribuído	1	0	0	1
resolvido	0	2	0	2

Por Gravidade	Aberto	Resolvido	Fechado	Total
trivial	1	0	0	1
pequeno	0	2	0	2
grande	1	0	0	1
travamento	1	0	0	1

Figura 3: Relatório de Resumo de Alterações

Fonte: Disponível em: < http://www.devmedia.com.br/articles/viewcomp.asp?comp=15462>. Acesso em: 17 Fev. 2012

1.5.1.2 Registro de Mudanças

O registro de mudanças é registro no item RELATAR CASO no menu principal e para esta mudança ser registrada precisa estar vinculada a um projeto e este por sua vez, precisa ter sido inserido no cadastro de projetos.

Segundo Silva (2009), o Mantis associa os usuários do sistema aos projetos que eles estão alocados e isto é uma característica importante, pois uma alteração registrada, todos os desenvolvedores do projeto ficam cientes daquele registro.

Para criar o registro, é necessário preencher os campos com asterisco, pois estes são obrigatórios. Alguns campos são relevantes serem preenchidos como o campo CATEGORIA onde é preenchido o módulo do projeto, como por exemplo, cadastro, cálculos, configuração do *software* ou ainda relatórios. Após a finalização do registro das alterações solicitadas, toda a equipe de desenvolvimento é avisada por email e pode rejeitar ou confirmar a alteração.

É possível controlar o status das alterações através do ciclo de vida de uma mudança. A cada novo status a alteração altera a cor deixando bem visível a situação da alteração.

2 CONCLUSÕES

A falha no sistema é conseqüência de um defeito, engano ou de um erro que pode ser devido a uma falha no sistema operacional, na configuração do equipamento ou humana (desenvolvedor ou usuário).

Como a maioria dos defeitos é inserida na manipulação das informações do sistema, se faz necessário o uso de ferramentas para gerenciar os testes sobre tudo quando refere se a diagnosticar e identificar os defeitos e solicitar a sua resolução de maneira eficaz e eficiente para que não prejudique o restante do sistema.

Os defeitos precisam ser identificados, datados, descritos e classificados para possa ter um melhor gerenciamento através de relatórios estatísticos, consultas em casos semelhantes que podem auxiliar os desenvolvedores novos da organização.

Conclui-se que as ferramentas *open-source*, em particular a Mantis devido à quantidade de funcionalidades e ser um ambiente de multiplataforma, está ganhando o mercado corporativo devido aos benefícios alcançados com o uso contínuo. É uma das excelentes opções no gerenciamento dos defeitos e, sobretudo, na identificação e resolução destes defeitos, pois atende todos os requisitos necessários de acordo com o que foi estabelecido junto com a empresa e os usuários, criando uma base de conhecimento para futuras consultas neste ou em produtos semelhantes.

REFERÊNCIAS

AVILA, A. L.; S, R. O. **Introdução à Engenharia de Requisitos** Revista Engenharia de *Software Magazine,* Devmedia Editora, Rio de Janeiro: 1ª Edição, Ano I, p. 46-53, publicado em nov. 2007, disponível em: <http://www.devmedia.com.br/post-8028-Revista-Engenharia-de-Software.html> acesso em 12 jan. 2012.

BUGNET, disponível < http://sourceforge.net/projects/bugnet/> acesso em 13 fev. 2012.

BUGZILLA, disponível <http://www.bugzilla.org> acesso em 13 fev. 2012.

CAETANO, C., **Automação e Gerenciamento de Testes: Aumentando a Produtividade com as Principais Soluções Open Source e Gratuitas,** e-book disponível <http://www.linhadecodigo.com.br/ebook.aspx?id=2951> acessado em 03 fev. 2012, 2007.

CAETANO, C., **Gestão de Defeitos**, Revista Engenharia de *Software Magazine,* Devmedia Editora, Rio de Janeiro: 1ª Edição, Ano I, p. 60-67, publicado em nov. 2007, disponível em: <http://www.devmedia.com.br/post-8028-Revista-Engenharia-de-Software.html> acesso em 20 jan. 2012.

CAETANO, C., **Gestão De Testes**, Revista Engenharia de *Software Magazine,* Devmedia Editora, Rio de Janeiro: 3ª Edição, Ano I, p. 58-66, publicado em jan. 2008, disponível em: <http://www.devmedia.com.br/post-8028-Revista-Engenharia-de-Software.html> acesso em 10 jan. 2012.

CARD, D. N., ***Defect Analysis: Basic Techniques for Management and Learning****, Advances in Computers* 65: 260-297, 2005.

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

CRISTALLI, R. S. et. al., **Base de Conhecimento em Testes de *Software*,** Martins Fontes, 2ª Edição, 2007.

COSTA, M. N.; KALINOWSKI, M., **Melhorando processos de *software* através de análise causal de defeitos**, Revista Engenharia de *Software Magazine*, Devmedia Editora, Rio de Janeiro: 3ª Edição, Ano I, p. 32-37, publicado em jan. 2008, disponível em: <http://www.devmedia.com.br/post-8028-Revista-Engenharia-de-Software.html> acesso em 10 jan. 2012.

DELFIM, S. M., Gestão de Defeitos, publicado em fev. 2008, disponível em <http://portalarquiteto.blogspot.com/2008/02/gesto-de-defeitos.html> acessado em 07/02/2012

DIAS, A. C. N., **Introdução a Teste de *Software***, Revista Engenharia de *Software Magazine*, Devmedia Editora, Rio de Janeiro: 1ª Edição, Ano I, p. 54-59, publicado em nov. 2007, disponível em: <http://www.devmedia.com.br/post-8028-Revista-Engenharia-de-Software.html> acesso em 20 jan. 2012.

FURASTE, P. A., **Normas Técnicas para o Trabalho Científico. Explicação das Normas da ABTN.** Porto Alegre: 13ª Edição. s.n., 2005.

IEEE Standard 610-1990: ***IEEE Standard Glossary of Software Engineering Terminology,*** **IEEE Press.**

KALINOWSKI, M., ***Defect Causal Analysis: An Opportunity for Product Focused Software Process Improvement, Invited paper (keynote)*** at the V CICIS (*Congreso Internacional de Computación y Ingenieria de Sistemas*), Moquegua, Peru, 2007.

KALINOWSKI, M., SPINOLA, R.O., DIAS NETO, A.C., BOTT, A., TRAVASSOS,

G.H., **Inspeções de Requisitos em Desenvolvimento Incremental: Uma Experiência Prática**, VI Simpósio Brasileiro de Qualidade de *Software*, Porto de Galinhas – PE, Brasil, 2007.

KOOMEN, Tim M. Pol ***Teste Process Improvement: a step-by-step guide to structured testing***, 1999.

JIRA, disponível: <www.atlassian.com/software/jira> acesso em: 13 fev. 2012.

MAIA, E., **Testador de *Software* é profissão em alta. Veja como ser um,** disponível <http://teteraconsultoria.com.br/blog/como-ser-um-testador-de-software/> publicado: 4 ago. 2008, acesso em 13 fev. 2012.

MANTIS, disponível: < http://www.mantis.org > acesso em: 15 dez. 2011.

MARTINHO, F., **TMAP Next (Test Management Approach) – Gerenciamento de Defeitos com TMap Next – Parte 5,** data da publicação; mai. 2011, Disponível <http://www.testexpert.com.br/?q=taxonomy/term/10>, acesso em 07 fev. 2012

MELO, A. C., **Desenvolvendo aplicações com UML 2.0 – do conceitual à implementação.** Rio de Janeiro: 2ª Edição. Brasport, 2004.

MOLINARI, L., **Testes de *Software* – Produzindo Sistemas Melhores e Mais Confiáveis.** São Paulo: 3ª edição. Erica, 2003.

PAULA, W. P. F., **Engenharia de *Software* – Fundamentos, Métodos e Padrões.** Rio de Janeiro: 2ª edição. GEN LTC, 2005.

PAULA, W. P. F., **Engenharia de *Software* – Fundamentos, Métodos e Padrões.** Rio de Janeiro: 3ª edição. GEN LTC, 2010.

PRESSMAN, R. S., **Engenharia de *Software*.** São Paulo: 3ª Edição. Makron Books, 1995.

RIBEIRO, C. Disponível <http://www.bugbang.com.br/?p=366#more-366> acessado em 04 fev. 2012.

ROBITAILLE, D., ***Root Cause Analysis – Basic Tools and Techniques***, Paton Press, 2004.

ROCHA, A. R. C., MALDONADO, J. C., WEBER, K. C. et al., **Qualidade de software – Teoria e prática**, Prentice Hall, São Paulo, 2001.

ROESSLER, R. G., **Ferramenta para Gestão de Defeito,** site Teste de Software, Francisco Beltrão (Paraná): Ano 2010, data da publicação 29 out. 2010, disponível em: < http://testersoftware.blogspot.com/2010/10/ferramenta-para-gestao-de-defeito.html>. Acesso em 07 fev. 2012.

SEI. ***SOFTWARE ENGINEERING INSTITUTE. CMMI for Development (CMMI-DEV***), Version 1.2*, Technical report* CMU/SEI-2006-TR-008. Pittsburgh, PA: *Software Engineering Institute, Carnegie Mellon University*, 2006.

SCARAB, disponível <http://scarab.tigris.org/ > acesso em 13 fev. 2012.

SHULL, F., ***Developing Techniques for Using Software Documents: A Series of Empirical Studies***, Ph.D. thesis, *University of Maryland*, College Park, 1998.

SILVA, L. D, SILVA, E. O, **Introdução ao Mantis,** Revista Engenharia de *Software Magazine,* Devmedia Editora, Rio de Janeiro: 20ª Edição, Ano II, publicado em jun. 2009, disponível em: <http://www.devmedia.com.br/articles/viewcomp.asp? comp=15462> acesso em 10 jan. 2012.

SOMMERVILLE, I., **Engenharia de *Software***. São Paulo: 8ª Edição. Pearson, 2007.
TRAC, disponível <http://trac.edgewall.org/>, acesso em 13 fev. 2012.

VIEIRA, Liliane dos Santos. **Pesquisa e Monografia Jurídica: na era da informática**. Brasília: 3ª edição. Brasília Jurídica, 2005.