CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS
CAMPUS TIMÓTEO

Lucas Bicalho de Freitas

FERRAMENTAS DE SUPORTE A AUTOMATIZAÇÃO DE GERENCIAMENTO DE DEFEITOS: LEVANTAMENTO BIBLIOGRÁFICO

Timóteo

2016

Lucas Bicalho de Freitas

FERRAMENTAS DE SUPORTE A AUTOMATIZAÇÃO DE GERENCIAMENTO DE DEFEITOS: LEVANTAMENTO BIBLIOGRÁFICO

Monografia apresentada à Coordenação de Engenharia de Computação do Campus Timóteo do Centro Federal de Educação Tecnológica de Minas Gerais para obtenção do grau de Bacharel em Engenharia de Computação.

Orientador: Marcelo de Sousa Balbino

Timóteo

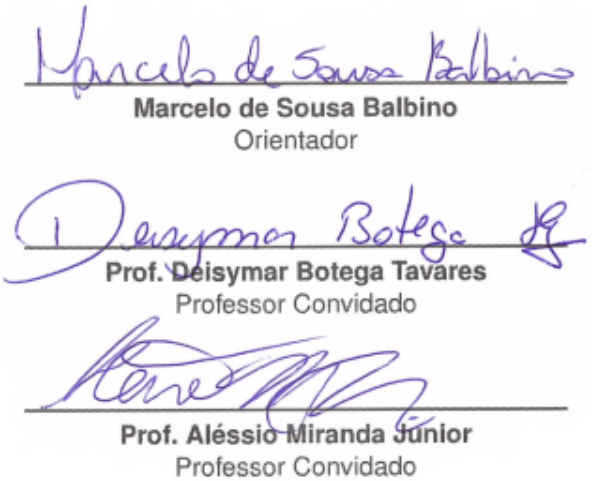
2016

Lucas Bicalho de Freitas

Ferramentas de suporte a automatização de gerenciamento de defeitos: levantamento bibliográfico

Monografia apresentada à Coordenação de Engenharia de Computação do Campus Timóteo do Centro Federal de Educação Tecnológica de Minas Gerais para obtenção do grau de Bacharel em Engenharia de Computação.

Trabalho aprovado. Timóteo, 29 de agosto de 2016:



Marcelo de Sousa Balbino
Orientador

Prof. Deisymar Botega Tavares
Professor Convidado

Prof. Aléssio Miranda Júnior
Professor Convidado

Timóteo
2016

Dedico esse trabalho à minha família e amigos,
sem eles não seria possível.

Agradecimentos

Quero agradecer primeiramente a DEUS, sempre presente na minha vida me dando forças para alcançar meus objetivos.

Aos meus pais, Adenilton e Rosimar, que sempre acreditaram em mim! Aos meus irmãos Matheus e Philipe, pela força e apoio.

Aos meus amigos, principalmente aqueles que me acompanharam durante toda minha trajetória acadêmica.

Agradecer ao meu orientador, Marcelo de Sousa Balbino, pela disposição e atenção em cada momento desse trabalho. Por corrigir e me orientar em cada detalhe, sempre me incentivando para conseguir concluir este trabalho. Obrigado pela parceria.

Aos demais professores do CEFET-MG, por todos conhecimentos transmitidos.

Enfim, quero agradecer a todos aqueles que participaram dessa longa e difícil caminhada. Obrigado!

“Motivação é a arte de fazer as pessoas fazerem o que você quer que elas façam porque elas o querem fazer.”.
Dwight Eisenhower

Resumo

O surgimento de defeitos em sistemas torna-se algo inevitável no processo de desenvolvimento de software. Para empresas esses defeitos são considerados como riscos e por isso descobri-los não é o suficiente. Baseado nesse contexto e pela falta de abordagem de tais assuntos na literatura de engenharia de software, esse estudo objetivou em realizar uma revisão da literatura sobre gerenciamento de defeitos, fornecendo como uma fonte de consulta para o meio acadêmico. Por meio da gestão de defeitos podemos acompanhar a qualidade do software com base nas informações cadastradas pelos usuários, facilitando a equipe de desenvolvimento em manter um histórico, além de facilitar o gerenciamento. A coleta de informações são realizadas pelos repórteres e é feita a partir de ferramentas de gestão de defeitos, conhecido como sistemas de rastreamento de defeitos. Sistema de rastreamento é uma aplicação projetada para auxiliar na garantia de qualidade do produto, além disso permite aos membros da equipe criar relatórios para apoiar os desenvolvedores na correção. Porém sistemas de rastreamento de defeitos ainda dependem da inserção manual dos dados relatados pelos usuários, tornando-se um problema para a obtenção dos dados necessários para a identificação da causa de um defeito. Por esse motivo soluções de melhorias foram estudadas e avaliadas por pesquisadores para conseguir um sistema de rastreamento mais automatizado, interativo, simples e com relatórios de qualidade. Inicialmente foi realizado uma pesquisa exploratória por meio de revisão de literatura de artigos, baseando as pesquisas em palavras de busca que abordavam o tema gerenciamento de defeitos. Após levantar as informações necessárias e selecionar os artigos, foi feito a análise de um questionário a partir dos estudos realizados neste trabalho com o intuito de fornecer uma fonte de estudo, respondendo as questões levantadas. Foi possível concluir que sistemas de rastreamento de defeito é essencial na prática de desenvolvimento de softwares, principalmente por determinar uma relação entre a necessidade de desenvolver um software com o controle e identificação dos defeitos na busca por qualidade.

Palavras-chave: Gestão de defeitos, Qualidade de software, Sistemas de rastreamento de defeitos.

Abstract

The emergence of bugs in system has become inevitable in the software development process. For companies, such bugs are considered risks and so discovering them is not enough. Based on this context and the lack of such matters approach in software engineering literature, this study aimed to conduct a review of the defect management on literature, providing a source of consultation for academia. Through the defect management, it is possible to monitor software quality based on informations registered by users, thereby facilitating the development team to maintain a history and to facilitate the management. Information gathering of bugs are carried out by the bugs of reporters and is made from defects management tools, known as bug tracking systems. Bug tracking system, is an application designed to assist in product quality assurance also allows members from the create bugs reports to support developers in correction. However, bug tracking systems still rely on manually entering of data reported by users, becoming a problem for getting data necessary to identify the cause of a bug. Therefore, solutions improvements have been studied and evaluated by researchers to achieve a bug tracking system more automated, interactive, simple and quality bug reports. Initially it was performed an exploratory research through articles of literature review, based on research in search words that addressed the topic defect management. After raising the necessary information and select the items, it was made the analysis of a questionnaire from the studies carried out in this work in order to provide a source of study, answering the questions raised. It was concluded that bug tracking systems is essential in software development practice, mainly for determining a relationship between the need to develop software in the control and identification of bugs in the search quality.

Keywords: Defect management, Software quality, Bug Tracking System.

Lista de ilustrações

Lista de tabelas

Sumário

1 Introdução

Com o passar dos anos a evolução tecnológica ganha cada vez mais espaço no mercado, hoje em dia qualquer empresa utiliza deste recurso para gerenciar sua produção. Devido a essa grande evolução os sistemas foram se tornando maiores, buscando assim a redução de custos e a sua ampliação no mercado (BARTIé, 2002).

Segundo Bartié (2002) esse foi o motivo que tornou cada vez mais difícil produzir software de boa qualidade como o desejado; pode-se notar que ao longo dos anos houveram vários avanços no desenvolvimento de software, como mostra a Tabela 1 abaixo:

Tabela 1 – Evolução das organizações desenvolvedoras de software.

Evolução das organizações desenvolvedoras de software			
Caracteríticas	1960	1980	2000
Tamanho do software	Pequeno	Médio	Muito Grande
Complexidade do software	Baixa	Média	Alta
Tamanho da equipe de desenvolvimento	Pequena	Média	Grande
Padrões e metodologias de desenvolvimento	Interno	Moderado	Sofisticado
Padrões e metodologias de qualidade e testes	Interno	Emergente	Sofisticado
Organizações de qualidade e testes	Muito Poucas	Algumas	Muitas
Reconhecimento da importância da qualidade	Pequeno	Algum	Significativo
Tamanho da equipe de qualidade de testes	Pequena	Pequena	Grande

Fonte: (BARTIé, 2002)

Com esses avanços, os sistemas se tornam mais complexos e assim mais defeitos surgem. Segundo Sandusky e Gasser (2005), o defeito é inevitável quando se trata de software, principalmente por existir vários riscos no desenvolvimento que possam causar esses defeitos, por exemplo, o nível de conhecimento dos desenvolvedores, o levantamento realizado dos requisitos entre diversos fatores que influenciam na ocorrência de defeitos. Em se tratando de software, em qualquer momento alguma mudança poderá ocorrer por parte do desenvolvimento de novas funcionalidades, podendo introduzir defeitos (WEERD; KATCHOW, 2009).

Quando esses defeitos são encontrados, os gerentes de projetos assim como toda equipe de desenvolvimento tem a missão de gerenciá-los. Por meio de ferramentas existentes relacionadas a gestão de defeitos os gerentes podem minimizar esses defeitos e trazer qualidade ao software (JUST; PREMRAJ; ZIMMERMANN, 2008).

Essas ferramentas de gestão de defeitos conhecidos como sistema de rastreamento

de defeito têm como objetivo ajudar as equipes de teste e de desenvolvimento, pelo fato de possuir um relatório de defeito que contém todos os dados encontrados com as principais informações do defeito, facilitando assim a comunicação entre testadores e desenvolvedores pois qualquer defeito encontrado é reportado e assim é solicitado novos meios para correção (JUST; PREMRAJ; ZIMMERMANN, 2008).

Porém, a falta de abordagem de tais assuntos na literatura acadêmica de engenharia de software proporciona um grande interesse e motivação para realizar um levantamento bibliográfico dessas ferramentas que auxiliam no gerenciamento de defeitos, trazendo para o meio acadêmico e também para grandes, médias e pequenas empresas de software, o conhecimento de tais ferramentas, assim como, suas características, vantagens e desvantagens.

1.1 Estrutura do texto

Esse trabalho tem como objetivo estudar os conceitos fundamentais de gestão de defeitos e as ferramentas utilizadas para tal gestão.

No Capítulo 2 são descritos os principais conceitos em gerenciamento de software incluindo qualidade de software, gestão de defeitos dentre outros. No Capítulo 3, “Estado da arte”, são apresentadas as principais técnicas e ferramentas relacionadas a gestão de defeitos. O estudo dos métodos e materiais será abordado no Capítulo 4. As análises e resultados serão apresentados no Capítulo 5, e por fim, algumas considerações no Capítulo 6.

1.2 Objetivos

1.2.1 Objetivo geral

O presente trabalho consiste em realizar um levantamento bibliográfico sobre gerenciamento de defeitos, proporcionando assim uma fonte de consulta para a literatura acadêmica, para equipes de desenvolvimento de software e também gestores de software.

1.2.2 Objetivos específicos

Como objetivos específicos, este trabalho busca fornecer uma fonte de pesquisa relacionada a comparações de sistemas de rastreamento de defeitos que são apresentados em outros trabalhos, além disso será apresentando os sistemas, suas principais características, vantagens e desvantagens.

1.2.3 Questões de pesquisa

Este trabalho busca responder alguns questionamentos que são levantados referentes a gerenciamento de defeitos. São eles:

- Q1: Quais as contribuições da gestão de defeitos para obtenção de qualidade do software?

- Q2: A gestão de defeitos é feita automatizada ou geralmente se usa métodos não automatizados?
- Q3: Quais são as ferramentas de gestão de defeitos existentes para a gestão de defeitos e quais suas características?
- Q4: Quais são as vantagens de se utilizar tais ferramentas? Elas são eficientes?

2 Fundamentação Teórica

Nesse capítulo serão apresentados os principais conceitos sobre gerenciamento de defeitos. Inicialmente são abordados os conceitos básicos sobre gerenciamento de software, incluindo qualidade de software, testes de software, manutenção em software, gestão de defeitos e, por fim, são apresentados conceitos fundamentais sobre sistemas de rastreamento de defeitos (*Bug Tracking Systems*).

2.1 Qualidade de software

Consumidores de produtos de software esperam sempre um alto nível de qualidade quando adquirem seus programas. Por exemplo, quando um cliente adquiri um software de características multimídia espera que no mínimo os símbolos dos botões para parar, gravar e reproduzir músicas ou vídeos seja semelhantes aos padrões reais de um CD ou DVD PLAYER. Mas nem sempre as empresas de softwares proporcionam a qualidade de software desejada ao cliente (AKINWALE; DASCALU; KARAM, 2006).

Qualidade de software, de acordo com Bartié (2002), é um processo detalhista que foca todas suas etapas produzidas na conciliação de processo e produto, prevenindo e eliminando os defeitos. De uma forma mais clara, pode-se dizer que qualidade de software engloba diversos fatores que satisfazem o cliente como confiabilidade, funcionalidade, usabilidade, eficiência, manutenibilidade, portabilidade e documentação (PRESSMAN, 2002).

O fator mais importante para se ter um produto com alta garantia de qualidade nos dias atuais é avaliar o desempenho do software aprendendo como se comporta, esse é um fator essencial na engenharia de software. Com a complexidade que os software ganham cada vez mais com o passar dos anos torna-se de extrema importância atingir os níveis desejados de qualidade e confiabilidade (ABAEE; GURU, 2010).

Qualidade de software está diretamente relacionada a qualidade de processo de desenvolvimento do software. Porém, reduzir os custos de software é uma das tarefas mais importantes em engenharia de software, pelo fato de testes de software serem bastantes custosos (ABAEE; GURU, 2010).

As pessoas acham que os testes de um software ocorrem apenas na depuração do código fonte. Porém, a análise de defeitos e erros de um produto ocorre desde o início do projeto até o fim do seu ciclo de desenvolvimento (ABAEE; GURU, 2010).

O teste, dentre as outras fases presentes no ciclo de vida de um software, pode ser considerada a mais importante em termos de custo e tempo. De acordo com algumas pesquisas, esta é a parte que mais demanda tempo e dinheiro, por consequência disto, caso se queira reduzir o custo de desenvolvimento de um software é preciso reduzir o custo de testes (ABAEE; GURU, 2010).

As atividades de teste de software são geralmente planejadas com auxílio de ferramentas de automação de testes, podendo classificar os defeitos sempre que forem deparados durante o ciclo de vida do desenvolvimento e assim prever a ocorrência desses tipos de defeitos (ABAEE; GURU, 2010 apud MAIMON; ROKACH, 2005).

2.2 Gestão de defeitos

Quando fala-se em defeito, de forma geral, o que vem em mente é qualquer condição que se desvia do padrão esperado, com base em especificações e experiências anteriores. Como por exemplo uma cadeira, que pode ser considerada um assento que possui 4 pernas de apoio, se houver uma cadeira com apenas 3 pernas ela se torna uma cadeira defeituosa (IEEE. . . , 1996).

Mas em gestão de projeto, defeito pode ser definido como uma falha na lógica da funcionalidade que causa a impossibilidade da realização de alguma ação ou tarefa. Erro e defeito possuem conceitos completamente diferentes, pois o erro surge por meio de uma falha humana, já o defeito é o problema causado por esse erro (CAETANO, 2007).

Para encontrar um defeito e resolvê-lo, uma série de fatores são necessários, por exemplo, a localização do defeito, o tempo de duração e a gravidade. A localização do defeito seria em qual parte do software ocorreu o defeito, o tempo é estimado desde a hora que foi encontrado o defeito até a sua correção e a gravidade é a escala de severidade do defeito e seus impactos no software, podendo ser classificado como um defeito crítico, grande ou pequeno (KORHONEN; SALO, 2008 apud BASILI, 1980).

O principal objetivo da gestão de defeitos é a prevenção destes defeitos, a identificação, analisá-los, resolvê-los e proporcionar melhorias (KORHONEN; SALO, 2008 apud BASILI, 1980).

2.2.1 Processos de gestão de defeitos

O processo de gestão de defeitos tem como objetivo, além de prevenir os defeitos encontrados nos projetos, buscar integração entre os desenvolvedores e o time de testes do projeto, facilitando assim a melhoria do produto. O defeito encontrado tem diferentes valores, podem ser classificados como um defeito mesmo, ou engano e até mesmo uma falha (CAETANO, 2007).

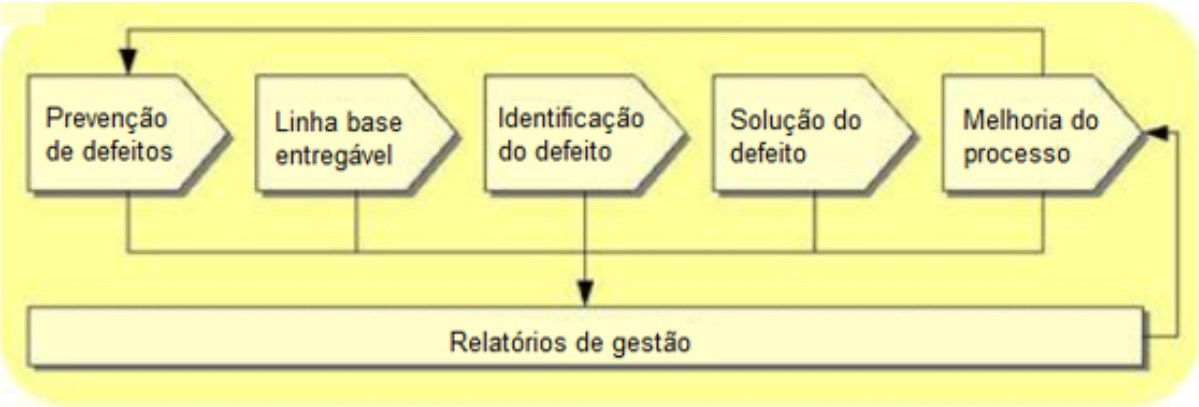
- Defeito: processo incorreto na execução da lógica do código. Exemplo: instrução incorreta na linha de código;
- Engano/Erro: ação humana que causa um resultado incorreto na execução. Exemplo: uma decisão realizada pelo desenvolvedor ou analista.
- Falha: depende tanto do erro quanto do defeito, é um desvio ocasionado por um processo incorreto da aplicação. Exemplo: é uma consequência de um erro ou defeito, gerando assim uma aplicação incorreta.

Existem diferentes tipos de defeitos, dentre os defeitos de usabilidade, que são defeitos que definimos como sendo aqueles no qual o usuário percebe na própria interface, e os defeitos de desempenho, que são aqueles defeitos que afetam o uso da memória e uma grande parte dos recursos do computador (YUSOP; GRUNDY; VASA, 2015).

Para a prevenção de defeitos em um projeto, existe um processo de gestão de defeitos (Figura 1) que é utilizado para auxiliar nesta tarefa, divido em diferentes etapas, como mostrar a seguir (CAETANO, 2007):

- Prevenção de defeitos: para prevenir a ocorrência de defeitos, são levantados os riscos críticos do projeto, e assim planejar minimizar os problemas destes.
- Linha base entregável: conhecido como *baseline*, é um guia que define todos os atributos já planejados. Serve para direcionar um caminho para o projeto fazendo comparações entre os resultados previstos e os realizados.
- Identificação do defeito: definição das técnicas necessárias para encontrar, reportar e classificar os defeitos, classificando-os de forma criteriosa.
- Solução do defeito: definição das ferramentas utilizadas para a correção desses defeitos.
- Melhoria do processo: análise de todo o processo que ocasionou o defeito, para entender a causa principal de sua ocorrência e assim promover melhorias para os demais processos.
- Relatório de gestão: geração de relatório com todos os dados levantados durante o progresso do projeto, definindo todos os desvios encontrados no projeto para que sirva de base para a medição de qualidade do software.

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



Fonte: (CAETANO, 2007).

2.3 Sistema de rastreamento de defeito

Todos os software possuem sistemas de rastreamento de defeitos para poder gerir seus relatórios. Esses sistemas permitem que diversos usuários de diferentes áreas possam relatar os defeitos encontrados (ALENEZI; BANITAAN, 2013).

Sistema de rastreamento de defeito é uma aplicação projetada para auxiliar na garantia de qualidade do produto. É considerada um tipo de sistema de controle de defeitos de código aberto, que permite a adição de defeito. Normalmente esses sistemas possuem um banco de dados que armazena todos os fatores de um defeito: tempo, gravidade e detalhes (ABAEE; GURU, 2010).

Nesses sistemas de rastreamento, os usuários e desenvolvedores podem relatar os defeitos encontrados durante a utilização do sistema por meio de uma interface personalizada e assim relatar os detalhes dos defeitos encontrados criando um relatório de defeitos (SASSO; LANZA, 2014).

2.3.1 Relatórios de defeitos

Relatórios de defeitos é praticamente uma descrição mais detalhada dos defeitos encontrados em um software. Objetiva que os usuários consigam de forma clara e sucinta relatar um defeito encontrado no software, para que os desenvolvedores possam entender e corrigir o problema (YUSOP; GRUNDY; VASA, 2015).

Um relatório comum de defeitos contém algumas especificações como, por exemplo o título e ID do defeito, o usuário que reportou, o status do defeito, a data de abertura e fechamento, a última data de modificação, a qual projeto o defeito pertence e os eventos do defeito durante todo o ciclo de vida (SASSO; LANZA, 2014).

Segundo Zimmermann et al. (2009), o principal objetivo desses relatórios de defeitos é facilitar a vida dos desenvolvedores em manipular as informações relatadas, que por sua vez utilizam dessas informações fornecidas para identificar a causa dos defeitos e assim resolvê-los facilmente.

2.3.2 Ciclo de vida

O ciclo da vida é conhecido por muitos como as constantes transformações que uma espécie passa para dar continuidade a vida até a morte. Em sistemas de rastreamento de defeitos, o ciclo de vida de um defeito é baseado conforme apresentado na Figura 2:

Figura 2 – Ciclo de vida de um defeito.

Fonte: (ALENEZI; BANITAAN, 2013).

Segundo Alenezi e Banitaan (2013), quando um novo relatório de defeito é realizado, é atribuído a um novo estado denominado de “novo”. Depois de serem avaliados e atribuídos a um desenvolvedor, seu estado é alterado para “atribuído”. Em seguida, o defeito passa a ser corrigido e seu estado muda para “resolvido”, podendo mudar seu estado para “verificado” ou “fechado” conforme for a resposta. O resultado contido por meio do relatório é usado para registrar a forma com que o defeito foi resolvido, caso contenha uma segunda via é considerado duplicado.

Se esse defeito não foi corrigido, ou não aparenta ser um defeito real ele será definido como inválido. Caso seja resolvido, mas apresentou pendência ele é reaberto, sendo então marcado como “reaberto” (ALENEZI; BANITAAN, 2013).

2.3.3 Manutenção em sistema de rastreamento de defeito

Sistema de rastreamento de defeito ajuda os desenvolvedores a identificar e comunicar os relatórios de defeitos e os problemas de desenvolvimento. Normalmente utilizam-se desses relatórios para orientar na manutenção de softwares, assim produzindo sistemas mais confiáveis (ALENEZI; BANITAAN, 2013).

Segundo Alenezi e Banitaan (2013), a triagem dos defeitos (*bug triaging*) é uma das

atividades de maior importância quando se trata de manutenção de software. A triagem dos defeitos é feita a partir da avaliação do novo relatório de defeito sendo atribuída a um desenvolvedor para que possa corrigi-lo.

No entanto, vários relatórios de defeitos com erros são reportados todos os dias, ocasionando uma demora excessiva no processo de correção, pois os desenvolvedores tentam reorganizar os defeitos como sendo novos ou reabertos (BAYSAL; HOLMES; GODFREY, 2012).

Após o sistema de rastreamento receber esse novo relatório, os desenvolvedores retiram as informações necessárias contidas no defeito e juntamente com seus conhecimentos podem tomar decisões, com base na severidade e no nível de prioridade (BORTIS; HOEK, 2011).

O processo de priorização do defeito é realizado manualmente, o que torna esse trabalho bastante cansativo e com grande chance de erro. Muitas vezes, os relatórios de defeitos foram atribuídos com níveis de prioridade incorretos pois dependem muito da experiência de quem julga (ALENEZI; BANITAAN, 2013).

Para classificar a prioridade desses defeitos são definidas certas etiquetas de classificação, ou seja, define-se como sendo Alto, Médio ou Baixo. Essa classificação ajudaria os desenvolvedores a definir qual defeito resolver primeiro, no caso aqueles com maior prioridade (ALENEZI; BANITAAN, 2013).

A definição do nível de prioridade de um defeito é feita pela análise das descrições textuais dos relatórios de defeitos como título e resumos. Depois dessa avaliação, outros tipos de recursos são verificados como por exemplo: a qual componente o defeito pertence, qual sistema operacional o defeito foi observado e qual a gravidade que representa a severidade desse defeito (ALENEZI; BANITAAN, 2013).

No entanto, os defeitos relatados para manutenção podem ser diferenciados pelo seu problema, sendo assim categorizados para avaliações futuras. Como demonstrado na Tabela 2 a seguir (LINTULA; KOPONEN; HOTTI, 2006):

Tabela 2 – Significado das categorias dos defeitos.

Categorias	Explicação
Erros ou falhas	Os usuários podem solicitar manutenção devido a problemas como se fossem erros ou falhas do software
Solicitação de recurso	Os usuários podem solicitar manutenção para pedir novas funcionalidades
Apoiar o pedido	Os usuários podem solicitar manutenção para apoiar na solução de certas pendências
Patches	Os usuários e desenvolvedores podem solicitar manutenção para atualizar o software com modificações

Fonte: (LINTULA; KOPONEN; HOTTI, 2006).

Após essas informações serem reportadas aos desenvolvedores, os mesmos recuperam o código-fonte do sistema e começam a executar as alterações necessárias. O código alterado por sua vez é atualizado no sistema de rastreamento de defeitos e o relatório é atualizado logo a seguir (LINTULA; KOPONEN; HOTTI, 2006).

3 Estado da arte

Quando uma empresa de projetos de software precisa gerenciar diversos sistemas, o número de defeitos que podem ser encontrados aumentam, dessa forma, gerenciar defeitos por planilhas torna-se praticamente impossível. Uma opção é passar a usar sistemas de rastreamento de defeitos (SERRANO; CIORDIA, 2005).

A maioria das empresas de software, independente da sua maturidade, usa de ferramentas de rastreamento de defeitos, por exemplo, o Bugzilla. No entanto, engenheiros de software são conhecidos por não gostarem de utilizar processos de software (AKINWALE; DASCALU; KARAM, 2006).

Há diversos tipos de ferramentas de rastreamento de defeitos disponíveis, serão apresentadas algumas características desses tipos de ferramentas e comparações nas seções a seguir.

3.1 Ferramentas *bug tracker*

Ferramentas *bug tracker* auxiliam os desenvolvedores a gerenciar seus projetos, permitindo aos usuários utilizar de relatórios para reportar os defeitos encontrados. Algumas dessas ferramentas foram levantadas neste trabalho como Mantis, TestLink, Bugzilla, In*Bug, iTracker e JIRA.

3.1.1 Mantis

Mantis é uma ferramenta que auxilia na gestão de defeitos, é de código aberto, desenvolvida em PHP (*Personal Home Page*), pode ser implantado em Windows, Linux ou MacOS, suporta qualquer navegador web e controla o ciclo de vida de um defeito do início ao fim (KSHIRSAGAR; CHANDRE, 2015).

Essa ferramenta traz um grande ganho de produtividade para a equipe de desenvolvimento, por ter uma série de características benéficas segundo a Caetano (2007):

- Execução em qualquer plataforma;
- Suportar vários bancos de dados;
- Suportar múltiplos mecanismos de autenticação;
- Criação ilimitada de projetos e relatos de defeitos;
- Visão geral da quantidade de defeitos apresentados;
- Concentração de informações em relação as mudanças dos defeitos;
- Controle de acesso e níveis de permissão para cada usuário;

- Histórico de evolução do projeto e relatórios;
- Notificações por e-mail;
- Interface *webservice*;
- MantisWAP – Suporte para dispositivos móveis.

Como dito, a ferramenta Mantis possui uma visão geral de todos os seus defeitos relatados como mostra a Figura 3 abaixo:

Figura 3 – Visão geral dos casos.

Fonte: (CAETANO, 2007).

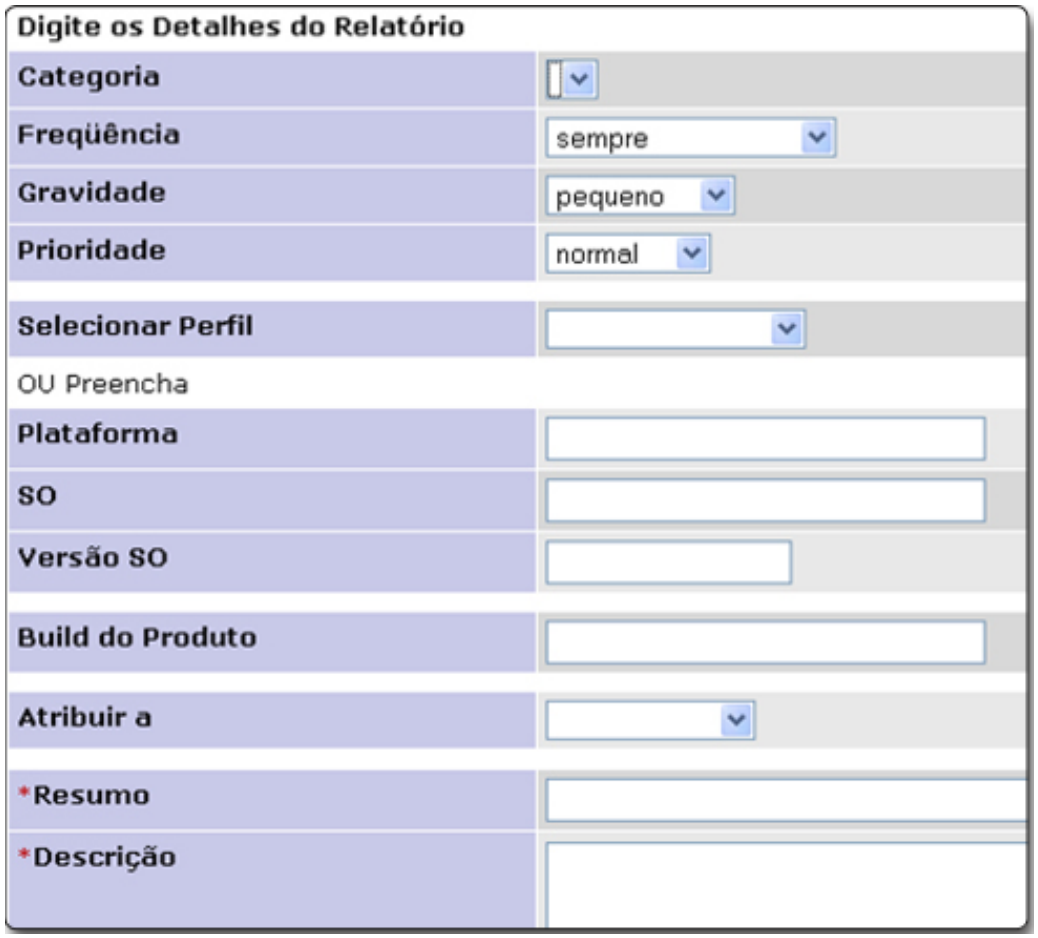
A ferramenta apresenta todos os casos, desde os mais importantes até mesmo os de baixa prioridade, relatando os casos que estão aguardando correção, os que já foram corrigidos, casos que foram fechados e os que foram retornados (CAETANO, 2007).

Figura 4 – Tela "Ver casos".

Fonte: (CAETANO, 2007).

Como pode-se ver, na Figura 4 é apresentado todos os casos, suas categorias e a gravidade dos mesmos. É nessa tela que é selecionada a opção para resolver o caso (CAETANO, 2007).

Figura 5 – Tela "Relatar caso".



Fonte: (MATOS; SILVA; SILVA, 2010).

É por meio dessa tela (Figura 5) que o usuário relata o defeito encontrado no software e assim a equipe de desenvolvimento pode avaliar e dar o suporte necessário para resolver esse defeito (MATOS; SILVA; SILVA, 2010).

O Mantis auxilia no gerenciamento dos defeitos durante o projeto, pois oferece relatórios e resumos consolidados de todos os defeitos, contendo *plug-ins* (módulo de extensão) que permite a visualização gráfica das principais métricas utilizadas na gestão de defeitos, em tempo real, como apresentado na figura a seguir (CAETANO, 2007):

Figura 6 – Tela "Relatórios".

Resumo

Por Status	Aberto	Resolvido	Fechado	Total
fechado	-	-	1	1

Por Gravidade	Aberto	Resolvido	Fechado	Total
pequeno	0	0	1	1

Por Categoria	Aberto	Resolvido	Fechado	Total
Defeito	0	0	1	1

Estatísticas de Tempo para Casos Resolvidos (dias)	
Caso aberto há mais tempo	0000001
Aberto há quanto tempo	0.13
Tempo médio	0.13
Tempo total	0.13

Estatísticas de Desenvolvedores	Aberto	Resolvido	Fechado	Total
desenvolvedor-1	0	0	1	1

Por Dia	
1	1
2	1
3	1
7	1
30	1
60	1
90	1
180	1
365	1

Por Resolução	Aberto	Resolvido	Fechado	Total
corrigido	0	0	1	1

Por Prioridade	Aberto	Resolvido	Fechado	Total
urgente	0	0	1	1

Estatísticas de Relatores	Aberto	Resolvido	Fechado	Total
administrator	0	0	1	1

Fonte: (CAETANO, 2007).

O relatório (Figura 6) mostra detalhadamente os casos que mais acontecem no software, separando-os por categorias, gravidade e *status*. Mostra também a evolução da equipe em tratar esses casos em relação aos dias abertos para cada caso (CAETANO, 2007).

A ferramenta Mantis pode ser utilizada para: identificar, registrar, reportar e acompanhar os defeitos encontrado no software.

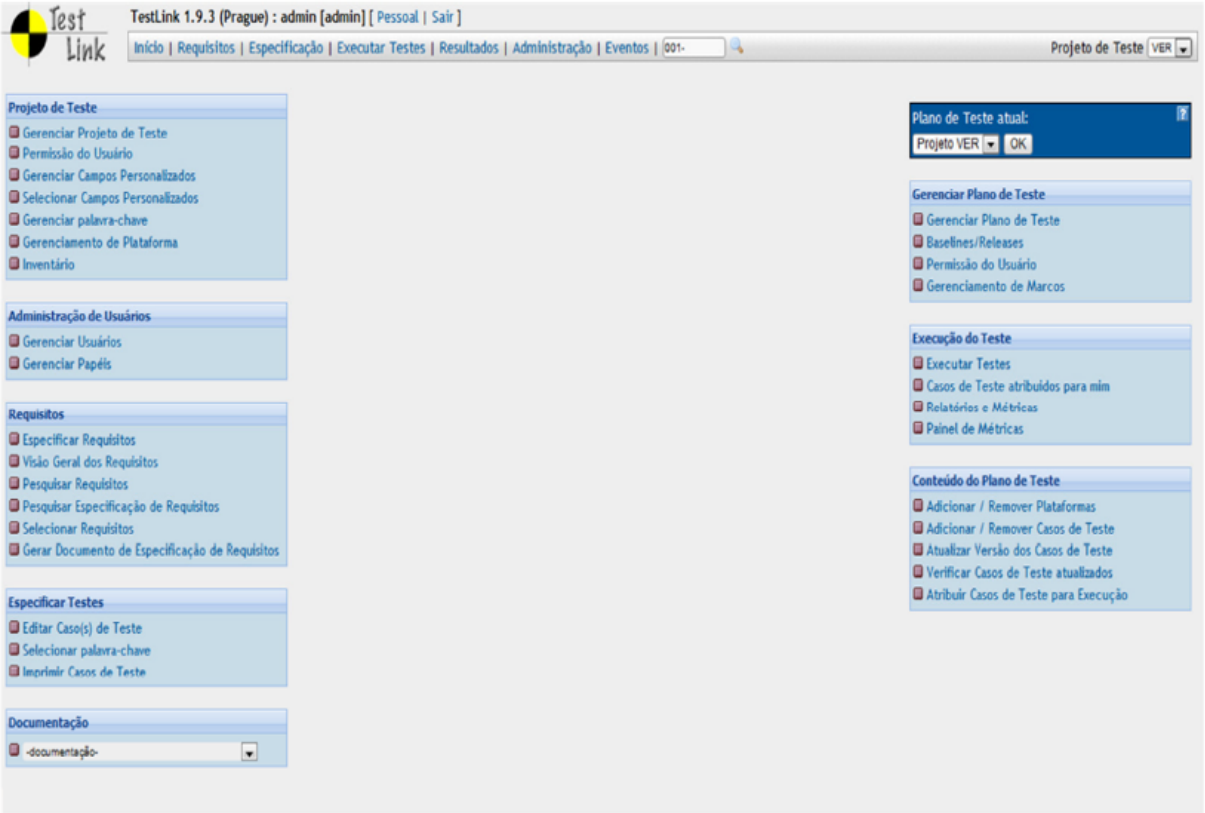
3.1.2 TestLink

TestLink é uma ferramenta que auxilia no gerenciamento de testes é de código aberto, desenvolvida em PHP. É possível cadastrar planos de testes, casos de testes, controlar os testes já executados, em execução e os que serão executados. Suas principais características são (TESTLINK, 2005):

- Diversos perfis de acesso;
- Multiusuário;
- Controle de execução;
- Histórico de execução para tomada de decisão;
- Cadastro de requisitos, planos de teste e casos de teste;
- Vários tipos de relatórios.

O TestLink é uma ferramenta que possui diversas funcionalidades (Figura 7) específicas para gestão como “Gerenciar plano de teste”, “Especificar”, “Inventário”, “Executar testes” e “Relatórios e métricas”, facilitando assim a organização e o acompanhamento do ciclo do projeto (NETO; OLIVEIRA, 2013).

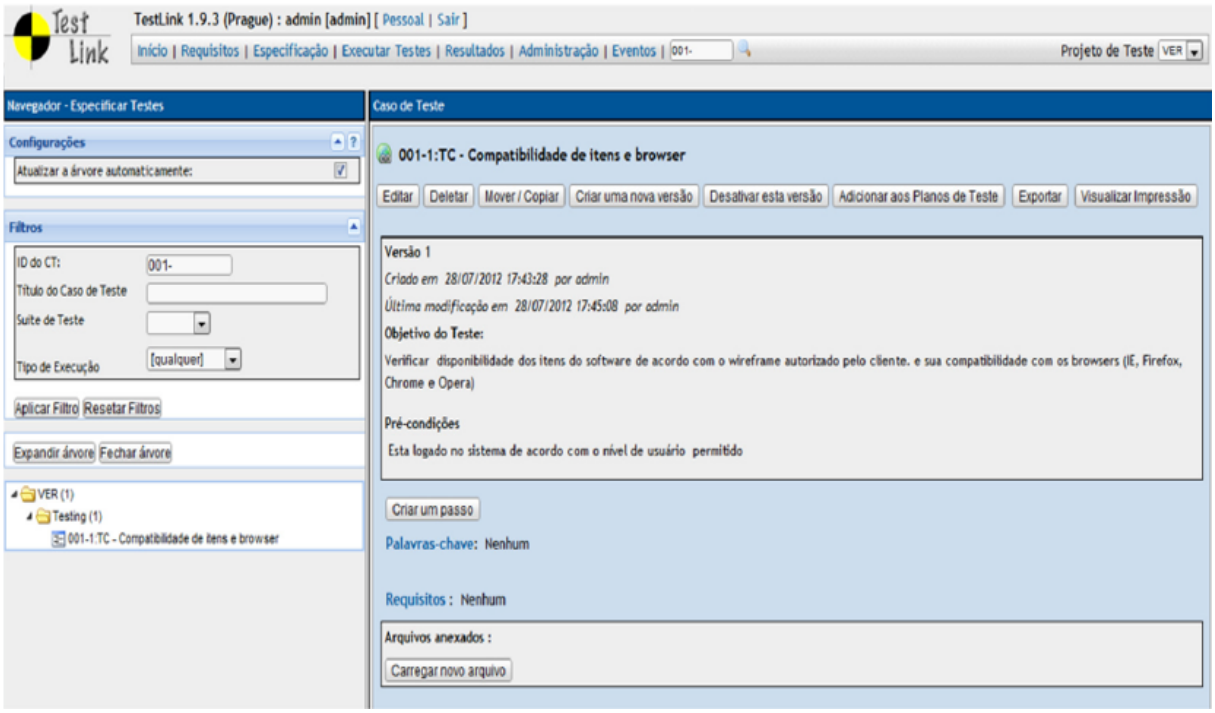
Figura 7 – Tela inicial do TestLink.



Fonte: (NETO; OLIVEIRA, 2013).

Durante a execução do planejamento dos testes, os critérios e os procedimentos serão definidos na etapa “Especificação” onde é responsável pela descrição dos casos de testes e suas especialidades como pré-condição, pós condição, resumo, ação entre outras funções que facilitam o TestLink na execução dos testes, como mostra a Figura 8 (NETO; OLIVEIRA, 2013).

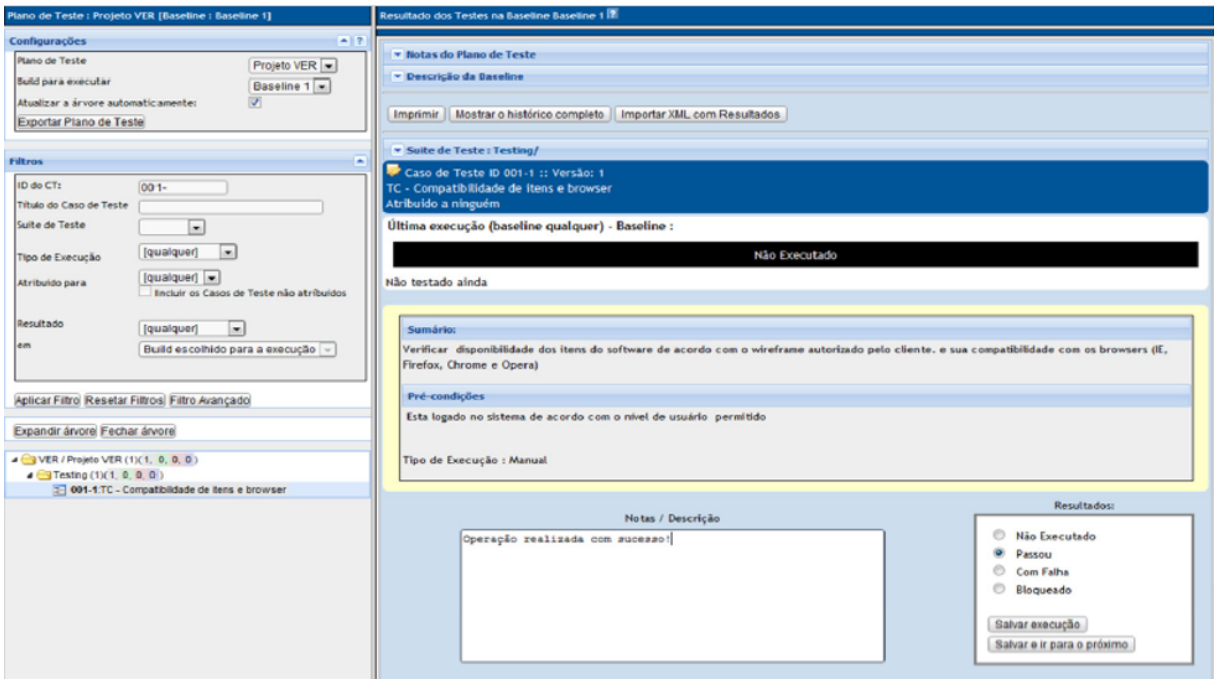
Figura 8 – Procedimento a ser executado.



Fonte: (NETO; OLIVEIRA, 2013).

A execução do processo é acompanhada e registrada facilitando assim futuras correções. Os defeitos encontrados podem ser classificados de acordo com os critérios de organização, definindo os riscos para o projeto e as consequências na qualidade desse produto. Ao serem identificados, uma ação corretiva deve ser realizada a fim de remover os defeitos encontrados adequadamente (NETO; OLIVEIRA, 2013).

Figura 9 – Execução de procedimentos pré-cadastrados.



Fonte: (NETO; OLIVEIRA, 2013).

Ao fim das execuções e correções, analises e relatórios dos resultados são realizados, como mostrado na Figura 9. A ferramenta TestLink gera de forma automática esses relatórios que auxiliam na gestão do projeto, possuindo uma visão mais objetiva na funcionalidade “Relatórios e Métricas” (NETO; OLIVEIRA, 2013).

Esses relatórios podem ser exportados em algum formato de arquivo ou podem ser observados nos diferentes tipos de relatórios presentes na ferramenta como mostra a Figura 10 (NETO; OLIVEIRA, 2013).

Figura 10 – Modelos de relatórios gerados pelo TestLink.

Fonte: (NETO; OLIVEIRA, 2013).

A ferramenta TestLink não é apenas uma ferramenta de gestão de defeitos, uma vez que permite cadastrar casos de teste e acompanha-los, sendo assim é uma ferramenta de gestão de testes como um todo.

3.1.3 Bugzilla

O Bugzilla (Figura 11) é uma ferramenta altamente popular, de aplicação *web*, de código aberto é utilizada principalmente para reporte de defeitos em software. É baseada em PERL (*Practical Extraction and Report Language*) (KSHIRSAGAR; CHANDRE, 2015).

Suas principais características são (TOLEDO; DAIBERT; ARAúJO, 2010):

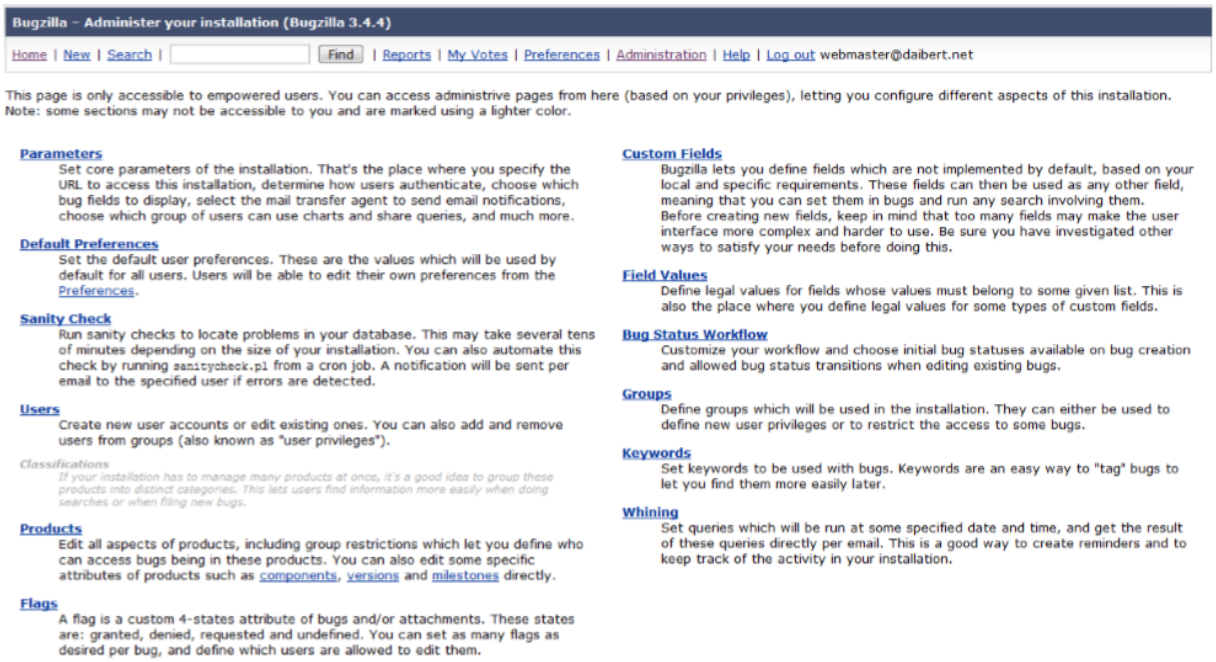
- Execução em um servidor *Web*;
- Suporta vários bancos: MySQL, PostgreSQL e Oracle;
- Instalação complexa, necessário acesso ao *shell* do servidor;
- Vários perfis de acesso: Admin, Usuário e Desenvolvedor;
- *Workflow* com diferentes *status* (*Bug Status Workflow*);
- Amplamente utilizada como plataforma de registro de defeitos.

Figura 11 – Tela inicial do Bugzilla.

Fonte: (TOLEDO; DAIBERT; ARAúJO, 2010).

O Bugzilla possui diversos níveis de perfis, por exemplo, a visão do administrador do sistema. O usuário que possui esse tipo de perfil tem todas as permissões e pode configurar todo o sistema de rastreamento Bugzilla. A tela inicial, após o *Login*, é apresentada na figura abaixo (TOLEDO; DAIBERT; ARAúJO, 2010):

Figura 12 – Tela administrativa do Bugzilla.



Fonte: (TOLEDO; DAIBERT; ARAúJO, 2010).

Como mostrado na Figura 12, são exibidos diversos *links* no topo da tela que são ações que podem ser executadas. São (TOLEDO; DAIBERT; ARAúJO, 2010):

- *Home* - tela principal do sistema, ao clicar volta para a interface inicial;
- *New* – reporta um novo defeito;
- *Search* – local para busca de defeitos reportados;
- *Reports* – lista de todos os defeitos reportados;
- *My Votes* – apresenta os comentários dos usuários para cada defeito;
- *Preferences* – disponibiliza uma interface para configuração personalizada de acordo com cada usuário;
- *Administration* – somente administrador tem acesso a esta interface, é o local responsável por toda configuração e personalização da ferramenta.

No Bugzilla, um usuário comum também pode acessar o sistema e reportar defeitos basta apenas se registrar. Sua visão inicial será a mesma que foi apresentada na Figura 12,

nessa interface o usuário poderá buscar um defeito em (*search*), reportar um novo defeito (*New / File a bug*) fazer *login* no sistema ou criar uma nova conta (*Open a new account*) (TOLEDO; DAIBERT; ARAúJO, 2010).

Para criar uma nova conta, basta clicar no botão "*open a new account*", logo em seguida na nova interface será solicitado o seu e-mail. O Bugzilla então irá enviar uma mensagem para o seu e-mail pedindo para validar o acesso (TOLEDO; DAIBERT; ARAúJO, 2010).

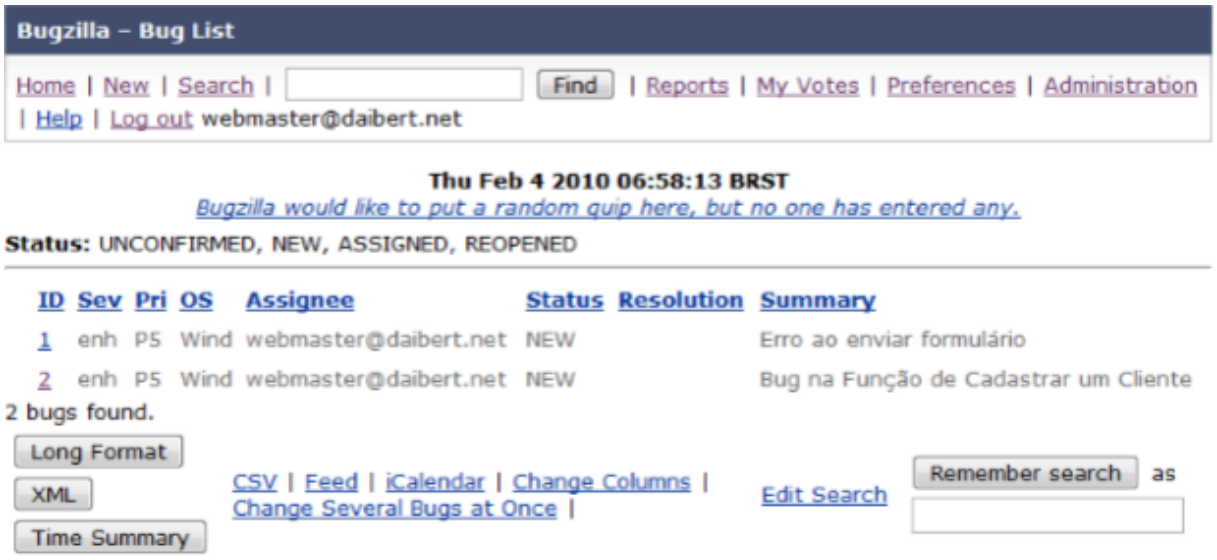
Com o seu cadastro ativo, esse novo usuário poderá acessar o sistema e reportar novos defeitos. Ao solicitar a aberturar de um novo *report*, o usuário será encaminhado para uma nova interface apresentada na Figura 13 abaixo (TOLEDO; DAIBERT; ARAúJO, 2010).

Figura 13 – Tela de *report* de defeito.

Fonte: (TOLEDO; DAIBERT; ARAúJO, 2010).

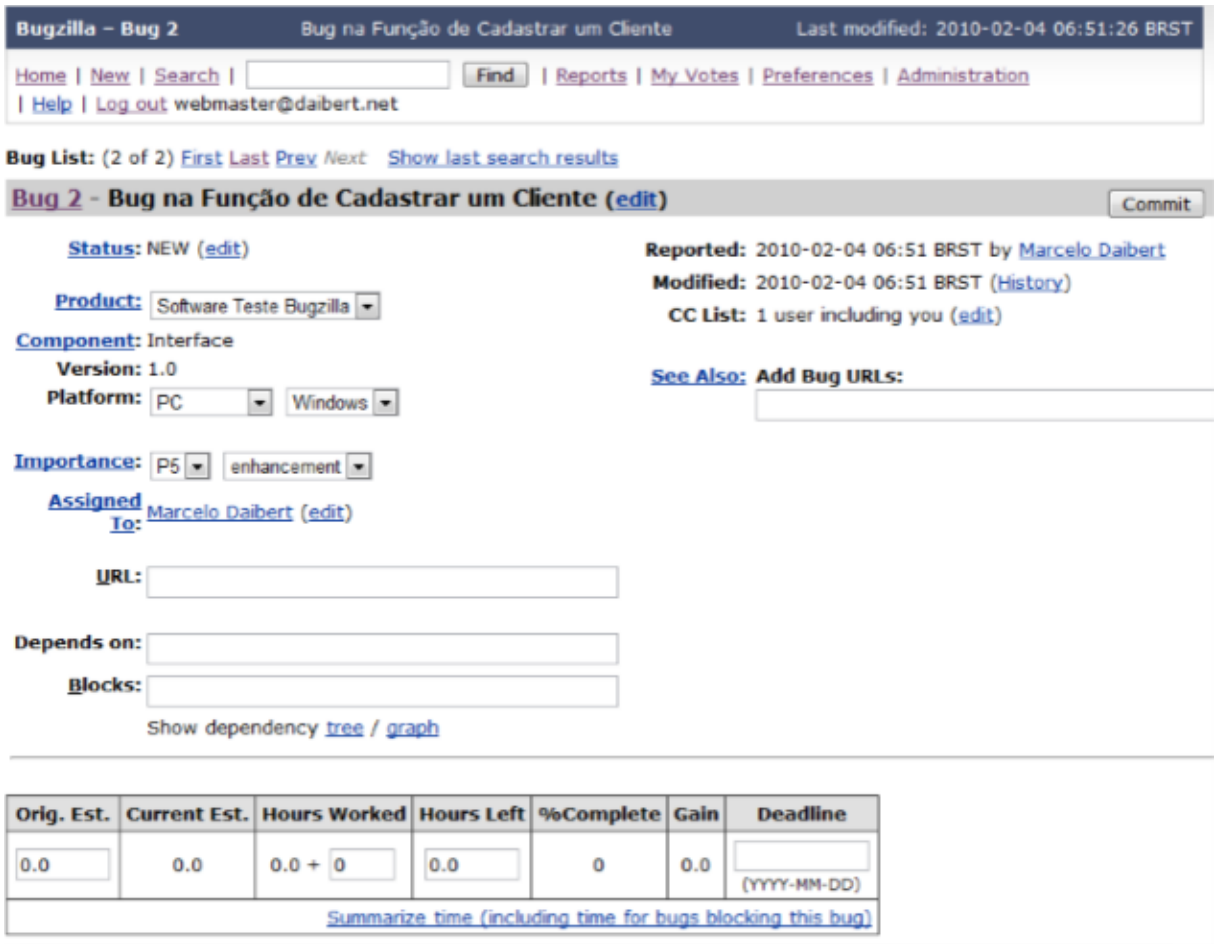
O desenvolvedor será responsável pela coleta dos dados, terá uma visão dos defeitos reportados como demonstrado na Figura 14. Nessa tela, será apresentada uma lista de defeitos e terá uma opção de pesquisa. Ao selecionar um defeito ele terá uma visão mais detalhada (Figura 15), com opções de visualizar o *status* do defeito além de corrigi-lo (TOLEDO; DAIBERT; ARAúJO, 2010).

Figura 14 – Visão dos defeitos reportados.



Fonte: (TOLEDO; DAIBERT; ARAúJO, 2010).

Figura 15 – Visão detalhada do defeito.



Fonte: (TOLEDO; DAIBERT; ARAúJO, 2010).

3.1.4 In*Bug

É uma ferramenta *web* que facilita a inspeção, navegação e compreensão dos relatórios de defeitos. A ferramenta In*Bug fornece uma visão geral de todos os dados do defeito de uma forma detalhada, interativa e complementar (SASSO; LANZA, 2014).

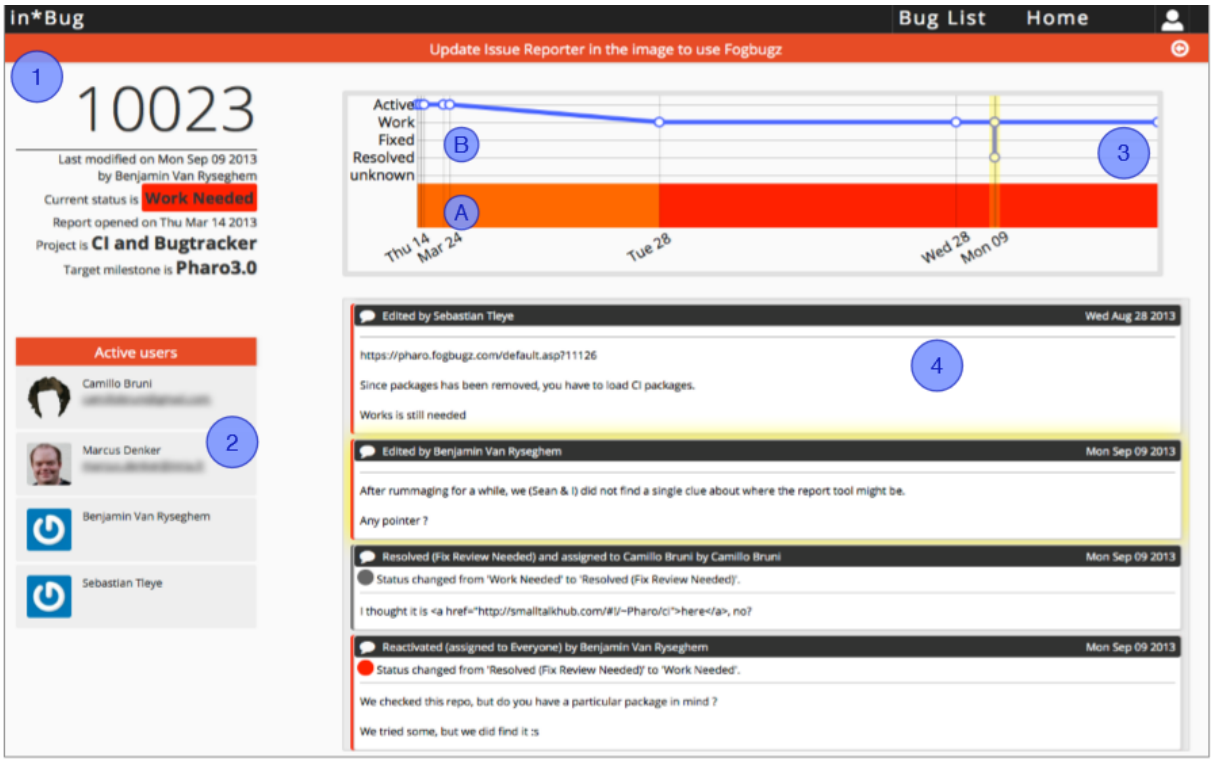
Figura 16 – Tela principal do In*Bug.

Fonte: (SASSO; LANZA, 2014).

Essa ferramenta possui diversas funcionalidades como mostrado na Figura 16. São elas (SASSO; LANZA, 2014):

- *Bug lifetime panel* (1): esta visão mostra os relatórios contidos no repositório de defeito, mostrando duração e status além de mostrar os eventos do defeito;
- *Project selection panel* (2): neste painel, o usuário pode escolher os projetos cujos defeitos mais lhe interessa. O tamanho da tag indica o número de defeitos reportados para o projeto;
- *Details panel* (3): nesse painel fornece toda informação comunicada sobre o relatório de defeito, relatando todos os detalhes do seu tempo de vida, como data de abertura, status, data da última modificação, etc;
- *Filter and options panel* (4): nesse painel os defeitos são classificados, existem três critérios padrão de classificação: questões do projeto, data de abertura e data que o defeito foi resolvido.

Figura 17 – Tela de detalhes do defeito.



Fonte: (SASSO; LANZA, 2014).

A Figura 17 apresenta de uma forma mais detalhada como é um relatório de defeito no In*Bug. Esse relatório apresenta alguns painéis como (SASSO; LANZA, 2014):

- *Bug Report Metadata* (1): onde mostra os dados importantes de um defeito: o id, a data da ultima modificação, o *status* atual, a data de abertura e fechamento, a qual projeto pertence e o principal erro;
- *User List* (2): exibe as pessoas envolvidas nesse defeito;
- *Bug Report Life Visualization* (3): mostra a vida do relatório de defeitos, mostrando todo o percurso desde a abertura ate o fechamento;
- *Event Interactive View* (4): é uma lista de todos os eventos que representa esse relatório, mostrando todos os dados utilizados nos eventos.

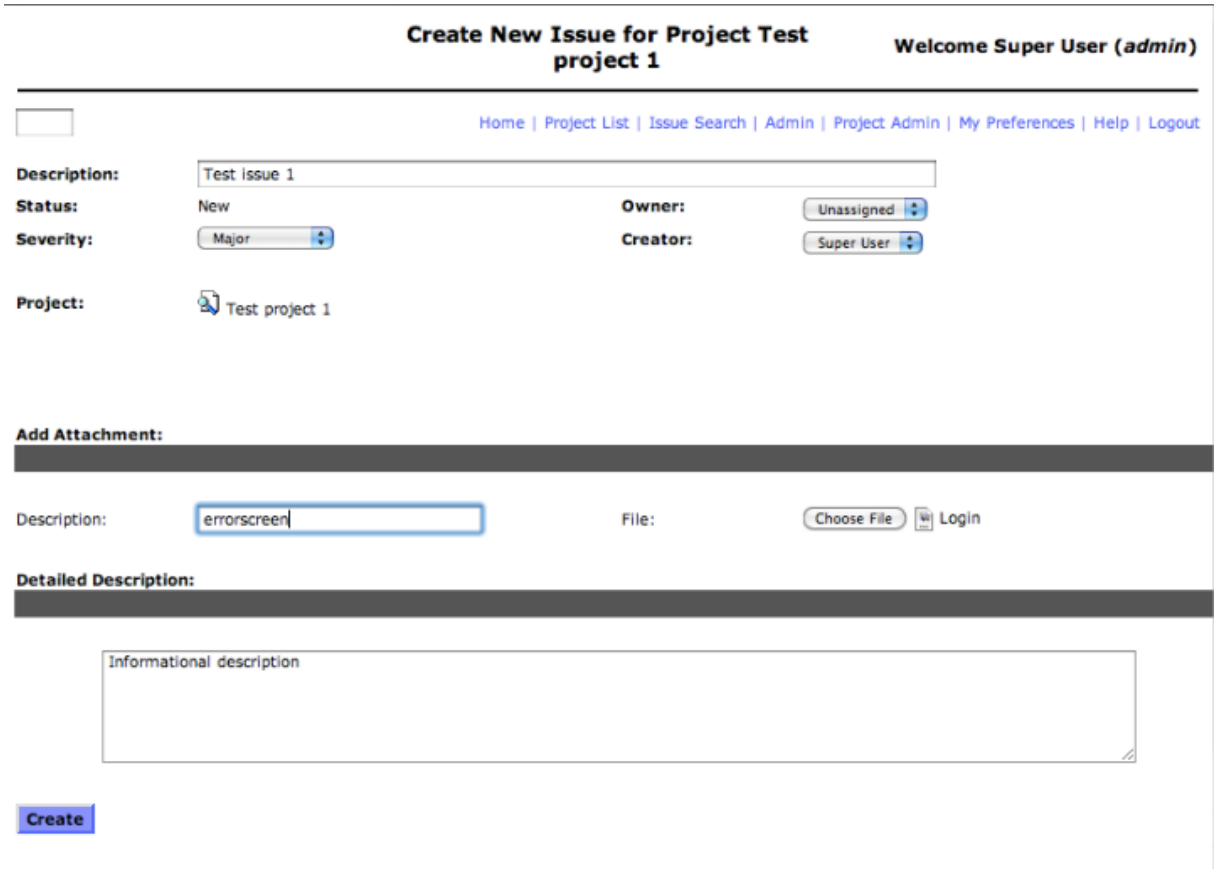
Concluindo a ferramenta In*Bug busca detalhar de uma forma mais interativa e compreensiva como um relatório de defeitos deve se comportar. Sua intenção é trazer maior visualização dos dados aos usuários facilitando assim a avaliação da evolução do projeto e a correção do defeito. Essa abordagem poderia ser aplicada a qualquer sistema de rastreamento de defeito (SASSO; LANZA, 2014).

3.1.5 iTracker

iTracker é uma ferramenta de gerenciamento de defeitos (Figura 18) de código aberto licenciado pela LGPL (*Lesser General Public License*). Foi projetado por Jason Carroll em 2002 sendo construído na linguagem Java (CARROLL, 2002).

A principal diferença dentre as suas características é a independência de plataforma, pois é uma aplicação J2EE (*Java2 Platform Enterprise Edition*) além de ter um banco de dados completamente independente (SERRANO; CIORDIA, 2005).

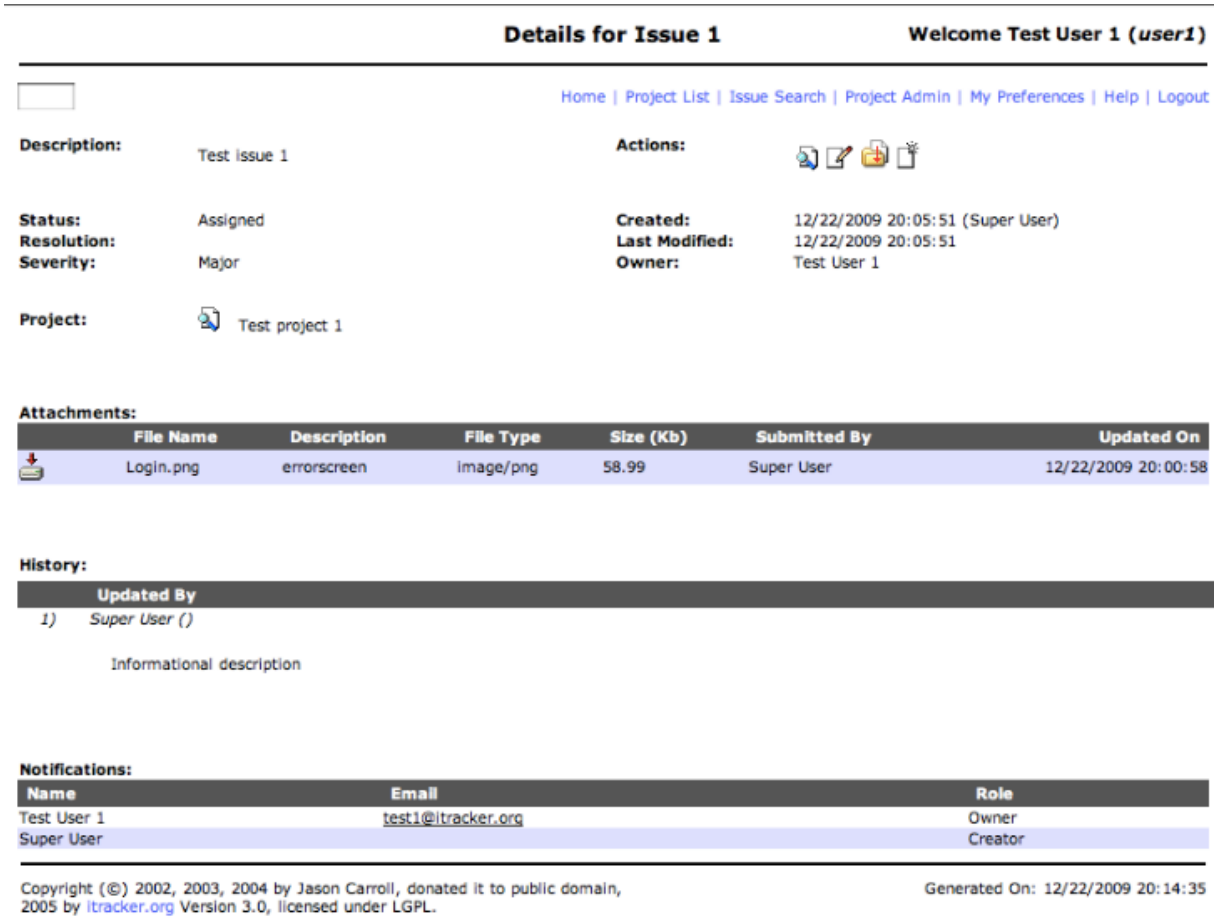
Figura 18 – Tela de criação de defeitos.



Fonte: (CARROLL, 2002).

É uma ferramenta semelhante ao Bugzilla, possui interface simples, porém objetiva. É rápido, customizável, modular e com diversas soluções (CARROLL, 2002).

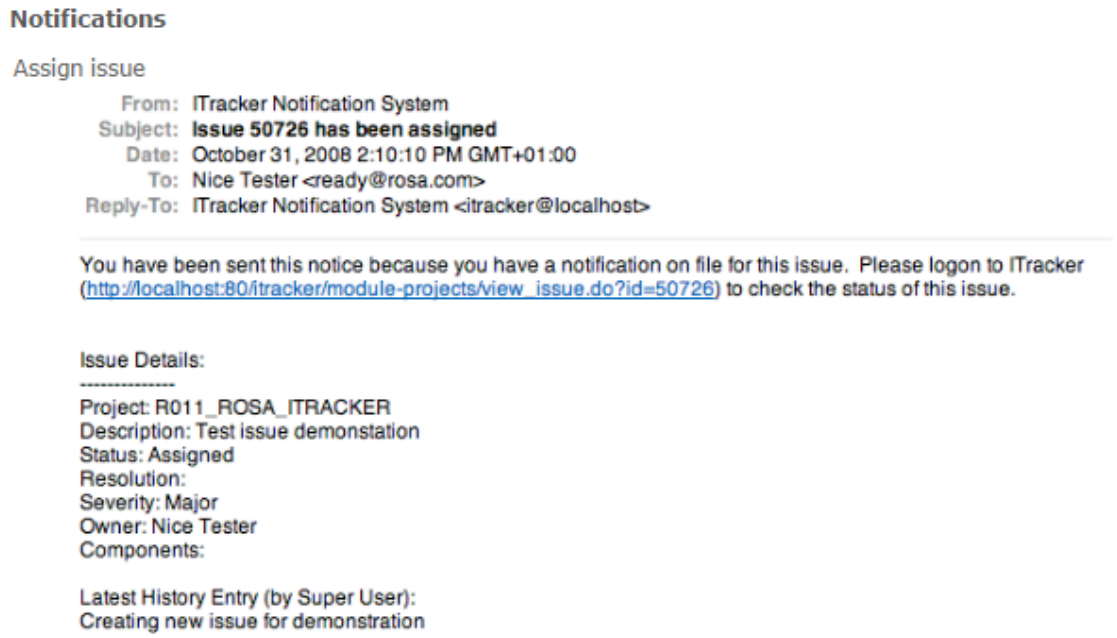
Figura 19 – Tela de visualização dos defeitos.



Fonte: (CARROLL, 2002).

Os defeitos reportados ficam em uma tela de visualização (Figura 19) para que os desenvolvedores possam pesquisar e assim resolvê-los. Defeitos que por sua vez podem ser enviados por mensagens via e-mail, como mostra a Figura 20 (SERRANO; CIORDIA, 2005).

Figura 20 – Notificação pelo e-mail.



Notifications

Assign issue

From: ITracker Notification System
Subject: **Issue 50726 has been assigned**
Date: October 31, 2008 2:10:10 PM GMT+01:00
To: Nice Tester <ready@rosa.com>
Reply-To: ITracker Notification System <itracker@localhost>

You have been sent this notice because you have a notification on file for this issue. Please logon to ITracker (http://localhost:80/itracker/module-projects/view_issue.do?id=50726) to check the status of this issue.

Issue Details:

Project: R011_ROSA_ITRACKER
Description: Test issue demonstation
Status: Assigned
Resolution:
Severity: Major
Owner: Nice Tester
Components:

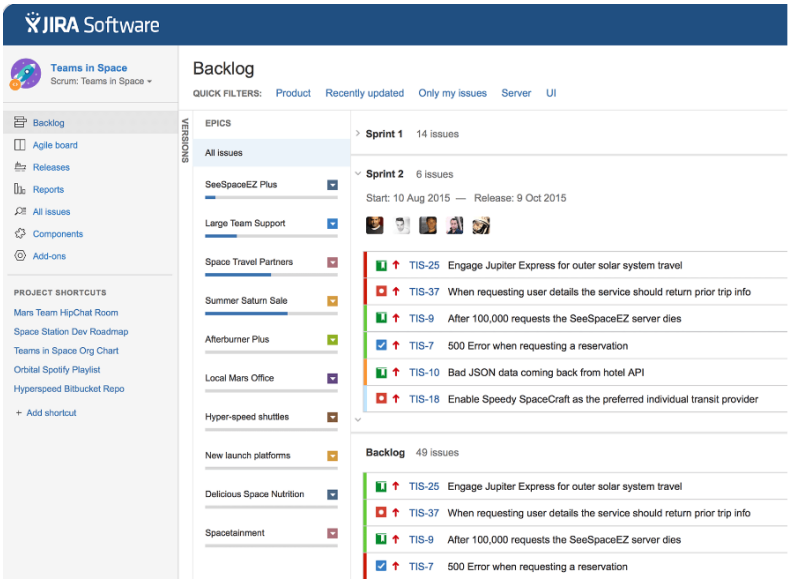
Latest History Entry (by Super User):
Creating new issue for demonstration

Fonte: (CARROLL, 2002).

3.1.6 JIRA

JIRA (Figura 21) é uma ferramenta *web*, construída para gerenciamento de projetos de equipes ágeis (*agile teams*), tem como objetivo planejar, controlar e gerenciar projetos de softwares, com características para acompanhar e reportar defeitos de um projeto (JIRA, 2004).

Figura 21 – JIRA.



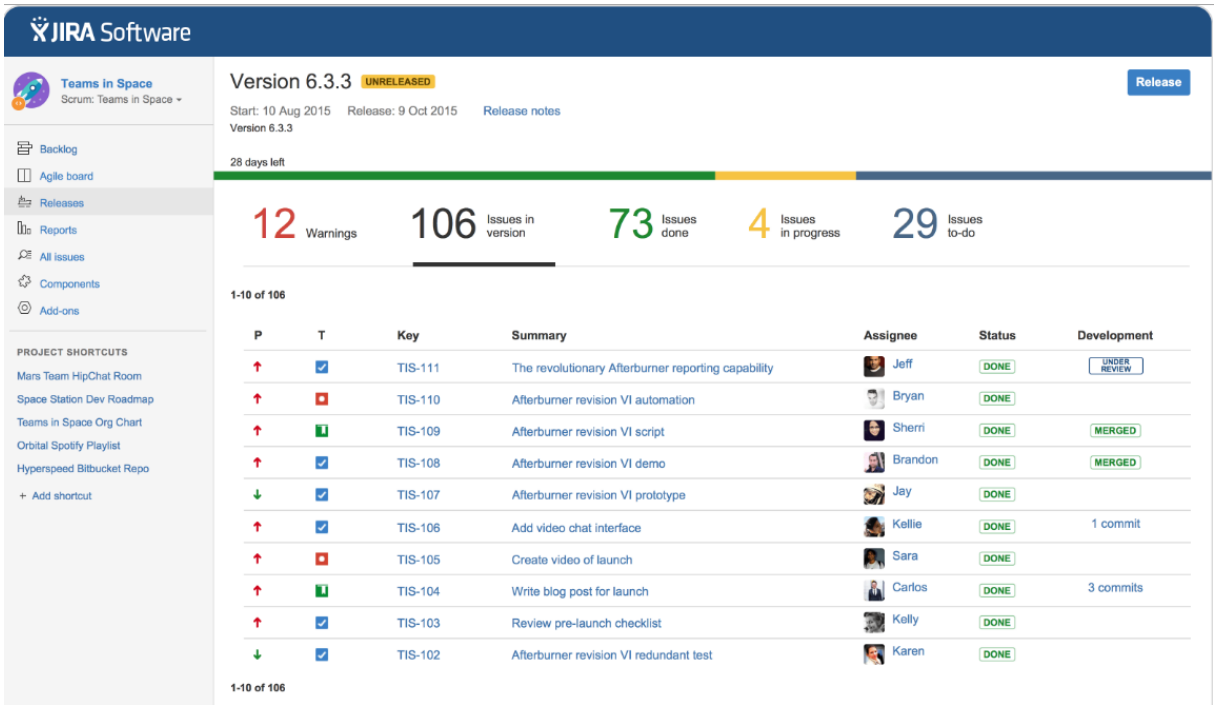
Fonte: (JIRA, 2004).

Dentre suas características se destaca principalmente em (JIRA, 2004):

- Gestão de defeitos: possui funcionalidades personalizáveis para detalhar e acompanhar os defeitos;
- Gestão de projetos: permite a criação de projetos por meio de fluxos de trabalho (*workflows*);
- Integração com outras ferramentas de desenvolvimento;
- Acesso via *web* pelos principais *browsers*, além de alertas por e-mail e centenas de ferramentas de auxilio (*plug-ins*) (módulo de extensão).

Além de ser uma ferramenta ágil e simples, JIRA possui relatórios de gestão que facilita o desempenho da equipe de desenvolvimento em tempo real, com base em dados visual que podem ser usados como mostra a Figura 22.

Figura 22 – Relatórios do JIRA.



Fonte: (JIRA, 2004).

3.2 Comparativo de ferramentas *bug tracker*

Sistemas de rastreamento de defeitos são utilizados por vários desenvolvedores e usuários, no qual os desenvolvedores usam das informações obtidas pelos relatórios para poderem identificar os defeitos. Existem diversos sistemas de rastreamento de defeitos, porém aquele sistema que apresenta ser eficiente no processo de desenvolvimento e com uma boa qualidade se torna um sistema valioso (KSHIRSAGAR; CHANDRE, 2015).

Em busca dessa qualidade de sistema alguns pesquisadores Yusop, Grundy e Vasa (2015) realizaram um estudo qualitativo para avaliação de ferramentas de gestão de defeitos com diferentes profissionais selecionados por grupos: desenvolvedores de software e testadores de software com diferentes níveis de experiência. Dos entrevistados, 60,7% são desenvolvedores de software e 39,3% são testadores de software, na Tabela 3 a seguir mostra o nível de experiência de alguns participantes (YUSOP; GRUNDY; VASA, 2015):

Tabela 3 – Distribuição do nível de experiência dos representantes.

Nível de experiência	**Possuem treinamentos com Interação Humano-Computador**			
	Testador		**Desenvolvedor**	
	Sim	Não	Sim	Não
Menos de 1 ano	0	3	2	2
Entre 1 a 3 anos	1	2	0	8
Entre 3 a 5 anos	0	3	1	3
Mais de 5 anos	3	10	3	15
Total de representantes	**22**		**34**	

Fonte: (YUSOP; GRUNDY; VASA, 2015).

Entre os entrevistados, dentre as ferramentas de código aberto de relatórios de defeitos, o Bugzilla foi a ferramenta mais utilizada seguido pela ferramenta JIRA. Mantis foi uma das ferramentas menos utilizadas, como mostrado na Tabela 4 (YUSOP; GRUNDY; VASA, 2015):

Tabela 4 – Ferramentas de relatórios de defeitos utilizados pelos representantes.

Ferramentas de relatórios de *bugs*	**Testador**	**Desenvolvedor**
Bugzilla	15	11
Mantis	1	0

Fonte: (YUSOP; GRUNDY; VASA, 2015).

A maioria dos entrevistados não estavam satisfeitos com as suas ferramentas. Para

alguns entrevistados os relatórios de defeitos eram bastante complexos principalmente para usuários não técnicos, como apresentado no Bugzilla e nada declarado no Mantis (YUSOP; GRUNDY; VASA, 2015).

Pode-se notar também que diversas ferramentas não suportam anexar arquivos, como arquivos de multimídia ou documentos ODF (*OpenDocument Format*), como forma de ilustrar e enriquecer a apresentação do defeito. Outros entrevistados queixaram-se que essas ferramentas não são capazes de depurar os defeitos de maneira automática (YUSOP; GRUNDY; VASA, 2015).

Além dessas ferramentas, outros sistemas de relatórios de defeitos foram utilizados na pesquisa como: JIRA, Redmine, Pivotal Track, Web Helpdesk, Trac, Target Process, Clearcase e FogBuz (YUSOP; GRUNDY; VASA, 2015).

Em outro artigo, Serrano e Ciordia (2005) compara as diferenças entre ferramentas *bug tracker*, entre o Bugzilla e iTracker como mostrado na Tabela 5 a seguir:

Tabela 5 – Comparação Bugzilla e iTracker.

Características	Bugzilla	iTracker
Plataforma é independente?	Principalmente para Linux, Windows com Cygwin	Sim, construído em J2EE
Independente de banco de dados?	Não, só funciona com o MySQL	Sim
Quantas vezes os profissionais utilizam?	Muitas vezes	Ás vezes
Como é sua personalização?	Baixo: Apenas para apresentação	Alto: Permite que os campos de relatórios sejam definidos pelo usuário
Número de usuários limitados?	Não	Não
Qual é o custo do ciclo de vida?	Custo médio	Custo médio-baixo
Permite equipes distribuídas?	Sim, Interface Web	Sim, Interface Web

Fonte: (SERRANO; CIORDIA, 2005).

Como pode-se ver, a grande diferença entres as ferramentas é o fato do iTracker ser construído em J2EE o que o torna independente de plataforma, além disso funciona em diferentes bancos de dados, enquanto Bugzilla é popularmente mais usual entre os profissionais. Apesar destas diferenças as ferramentas se assemelham (SERRANO; CIORDIA, 2005).

Em um outro trabalho realizado por Abaee e Guru (2010), algumas ferramentas *bug*

tracker: Bug-Track, RR-Tracker, BUGtrack e NetResults Trackers foram comparadas por meio de alguns critérios. Como por exemplo, se eram baseadas na *web*, sua flexibilidade com banco de dados, se possuíam relatórios personalizáveis, se importava e exportava dados, usabilidade, entre outras características.

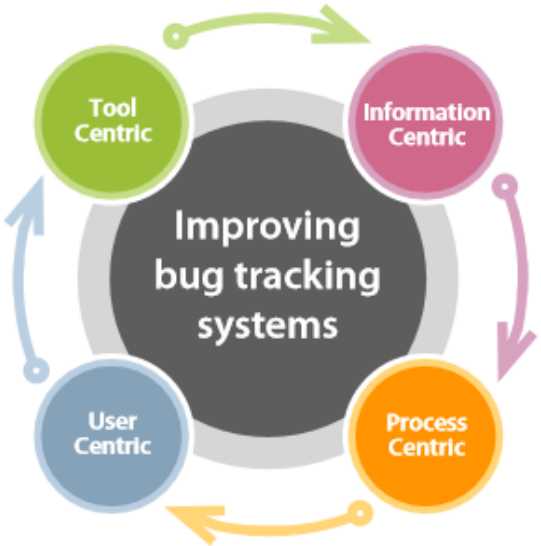
3.3 Melhorias em sistemas de rastreamento de defeitos

Sistemas de rastreamento de defeitos são essenciais para qualquer desenvolvedor de software, pois as informações obtidas são de extrema importância para identificação da causa de um defeito. Porém pesquisas mostram que na maioria dos casos usuários responsáveis por reportar os defeitos (repórteres) omitem itens importantes que força os desenvolvedores a solicitar novas respostas gastando tempo e produtividade (ZIMMERMANN et al., 2009).

Pode-se dizer que o motivo para ocorrer esse problema é que os sistemas de rastreamento de defeitos não dão suporte para os repórteres fornecerem as informações necessárias. São apenas sistemas que armazenam em grande quantidade os dados reportados (ZIMMERMANN et al., 2009).

Essas informações de dados que são armazenadas nos sistemas, inicialmente ajudariam os desenvolvedores a resolver os defeitos encontrados, porém nem sempre estão completas o que representa um problema. Por esse motivo algumas sugestões de melhorias para sistema de rastreamento de defeito serão apresentadas na Figura 23 a seguir (ZIMMERMANN et al., 2009):

Figura 23 – Melhorias para sistemas de rastreamento de defeitos.



Fonte: (ZIMMERMANN et al., 2009).

Como se pode ver na Figura 23, algumas ferramentas de auxílio poderiam ser incorporadas nos sistemas de rastreamento de defeitos, sendo elas:

- *Tool Centric*: essas ferramentas poderão ajudar na redução do volume de informações disponíveis. O sistema poderia ser configurado para localizar automaticamente os defeitos e ser reproduzidos por ferramentas de captura/reprodução de imagens como *screenshots* ou vídeos;
- *Information Centric*: são ferramentas que melhorariam a informação fornecida pelos repórteres podendo facilmente incorporar os sistemas de rastreamento de defeitos fornecendo *feedback* em tempo real dos dados reportados;
- *Process Centric*: essas ferramentas tem foco na administração e estão relacionadas a correção de defeitos, pois poderiam determinar qual desenvolvedor resolveria tal defeito assim automatizando as tarefas;
- *User Centric*: essas ferramentas educariam os repórteres para que soubessem fornecer e coletar as informações corretas. Assim, os desenvolvedores poderiam se beneficiar para resolver os defeitos.

3.3.1 Melhorias em relatórios de defeitos

Sistema de rastreamento de defeito muitas vezes exige muito dos usuários que não estão acostumados com as práticas de desenvolvimento. Por esse motivo, causam frustrações aos desenvolvedores pela falta de qualidade que os relatórios de defeitos são apresentados (JUST; PREMRAJ; ZIMMERMANN, 2008).

Apesar destes relatórios apresentarem interfaces aperfeiçoadas, podem causar problemas na gestão de defeitos, pois muitas vezes os relatórios contém informações difíceis de serem interpretadas e extraídas pelos desenvolvedores. Isso ocorre pelo fato de que os elementos necessários que podem influenciar um defeito são consideravelmente altos e o armazenamento desses defeitos em forma de texto deixa mais complicado para extrair informações (SASSO, 2014).

Segundo Just, Premraj e Zimmermann (2008) uma pesquisa foi feita com desenvolvedores para saber o que influenciava na qualidade dos relatórios de defeitos e quais os problemas que mais ocorriam e os mais difíceis para corrigir. Os resultados encontrados foram:

- Problemas em reproduzir e utilizar os dados obtidos por falta de informação correta;
- Duplicações de defeitos, não é considerado algo problemático mas as vezes podem atrapalhar os desenvolvedores;
- A falta de conhecimento dos repórteres, o que ocasionava em informações incorretas.

Com esses resultados, pode-se notar que a necessidade de melhoria em relatórios de defeitos seria crucial para melhorar os sistemas de rastreamento de defeitos. Porém a diversidade de defeitos encontrados em relatórios é enorme e os defeitos de usabilidade também geram problemas que segundo Yusop, Grundy e Vasa (2015) são problemas inconvenientes para auxiliar na gestão de defeitos, devido a:

- Relatórios de defeitos genéricos;
- Sem suporte para apoiar qualquer anexo de arquivo, especialmente multimídia;
- A falta de ferramentas integradas para coletar informações automaticamente;
- Dificuldade para classificar adequadamente os defeitos, sendo relacionados a usabilidade ou desempenho.

Sasso (2014) afirma que automatizar os passos para uma criação de um relatórios de defeitos mais interativo e mais sólido tende a melhorar a relação dos desenvolvedores com o uso de sistema de rastreamento de defeito. Uma visão de melhoria para relatórios de defeitos seria as seguinte:

- Melhorar a forma como os relatórios de defeitos são coletados, usando técnicas de extração automatizadas;
- Melhorar o processo de compreensão dos relatórios, fornecendo visualizações interativas para auxiliar no processo de fixação do problema;
- Reduzir o tempo gasto na reprodução de um defeito;
- Fornecer uma plataforma interativa para os desenvolvedores poderem resolver os defeito.

Já para Yusop, Grundy e Vasa (2015) melhorar os relatórios de defeitos com defeitos de usabilidade seriam necessários:

- Formulários para notificação de defeitos mais simples, contendo dicas ou qualquer outro tipo de auxilio técnico para os repórteres;
- Poder anexar arquivos multimídias, como vídeos, *screenshots* entre outros documentos;
- Ferramentas de depuração automática dos defeitos, por exemplo, integrar ferramentas de captura para os relatórios, assim qualquer ação dos repórteres poderiam ser gravadas;
- Utilizar mais palavras-chaves e termos específicos para os diversos tipos de defeitos, para facilitar a classificação dos defeitos e assim facilitar a sua correção.

Conclui-se que os relatórios de defeitos atuais não estão conseguindo fornecer as informações necessárias para os desenvolvedores. Sem estas informações os desenvolvedores não podem resolver os defeitos de maneira ágil e por isso acredita-se que essas melhorias nos relatórios poderiam ajudar os sistemas de rastreamento de defeitos (ZIMMERMANN et al., 2009).

3.3.2 Duplicações de defeitos

Como sabemos, sistemas de rastreamento de defeitos são utilizados por diversos usuários em tempo ilimitado, isso de certo modo pode gerar alguns problemas para os relatórios de defeitos, por exemplo, a duplicação de um defeito já relatado anteriormente (THUNG; KOCHHAR; LO, 2014).

Segundo Just, Premraj e Zimmermann (2008) defeitos relatados com frequência, por mais de uma vez, nem sempre são problemáticos, pois essas informações as vezes podem ser um adicional importante do defeito relatado.

Por esse motivo a necessidade de ter uma ferramenta de apoio aos sistemas de rastreamento para a localização e prevenção dos defeitos duplicados é bastante importante, pois facilitaria o desenvolvedor a identificar se a duplicação é desfavorável ou favorável (THUNG; KOCHHAR; LO, 2014).

Vários pesquisadores propuseram técnicas para a resolução desses problemas. Essas técnicas são baseadas em algumas abordagens: sem supervisão e supervisionados, como apresentado a seguir:

- Abordagem sem supervisão é uma técnica de recuperação de dados onde são identificados os relatórios duplicados por meio do modelo de espaço vetorial onde são consultados os termos com mas similaridade e assim comparados com os novos relatórios (THUNG; KOCHHAR; LO, 2014);
- Abordagem supervisionado é uma técnica na qual, após um novo relato de defeito em um relatório, uma lista de defeitos semelhantes é ordenada para prever se houve duplicação usando semelhança do texto, características entre outros fatores (THUNG; KOCHHAR; LO, 2014).

Essas abordagens foram estudadas e avaliadas em vários relatórios e demonstrou-se bastante eficaz. Mas nenhuma dessas técnicas são integradas a sistemas de rastreamento de defeitos. Porém a ferramenta DupFinder busca resolver esse problema e foi feita para ser implementada junto ao sistema de rastreamento de defeito (THUNG; KOCHHAR; LO, 2014).

DupFinder é uma ferramenta que calcula a semelhança entre os textos descritos nos relatórios de defeitos anteriores com os defeitos que serão reportados. Os principais campos de análise dos relatórios que essa ferramenta utiliza são os campos textuais: resumo do defeito e descrição (THUNG; KOCHHAR; LO, 2014).

Existem três etapas de processamento para os textos reportados, que são: *tokenization*, remoção de palavras de parada e palavras derivadas (KSHIRSAGAR; CHANDRE, 2015).

- *Tokenization*: é uma etapa onde são removidos os símbolos especiais como, por exemplo, espaços, sinais de pontuação e etc.;
- Remoção de palavras de parada: nesse caso, as palavras de parada, aquelas irrelevantes, são removidas como “um”, “uma”, “o”, “ou” e etc.;

- Remoção de palavras derivadas: as palavras derivadas são reduzidas a sua forma raiz, como por exemplo, “correndo” para “corre”.

O cenário de uso dessa ferramenta é muito simples, por exemplo, um usuário insere um texto de descrição do defeito e o seu resumo nos campos no relatório de defeito. Logo em seguida, um *script* é executado e enviado para o servidor que retorna os conteúdos em uma tabela com os possíveis relatórios duplicados, essa resposta é notificada ao usuário mesmo, sendo positiva ou negativa (THUNG; KOCHHAR; LO, 2014).

3.3.3 Sugestões para melhoria da correção dos defeitos

As diversas retribuições de defeitos presente em sistemas de rastreamento de defeitos ocasionadas devido as tomadas de decisões feitas pelos desenvolvedores é um grande desafio, e por esse motivo algumas sugestões para melhorar a correção dos defeitos são realizadas como se pode ver a seguir (BORTIS; HOEK, 2011):

- Conhecimento desenvolvedor: ter um conhecimento geral do projeto e de toda a equipe envolvida é essencial para realizar uma boa correção dos defeitos;
- *Feedback* instantâneo: a cada tarefa realizada uma resposta deve ser imediatamente apresentada ao sistema de rastreamento de defeito para poder tomar decisões;
- Histórias dos defeitos: é importante saber as informações históricas dos defeitos, pois isso facilita para o desenvolvedor na correção;
- Relações de defeitos: em grandes projetos, vários defeitos se relacionam entre si e devem ser atribuídos para o mesmo desenvolvedor. Essa relação facilita na correção;
- Facilidade de entrada: o processo de inserção dos defeitos e suas informações na criação dos relatórios serem feitas de uma forma mais ágil e prática;
- Público e Privado: dar opções para os desenvolvedores apresentarem suas informações para diferentes perfis com relação ao seu nível de privacidade, sendo assim, guardando dados confidenciais.

Essas práticas poderiam ser utilizadas para apoiar e coordenar os desenvolvedores de qualquer tipo de sistema de rastreamento de defeito.

4 Materiais e métodos

Este trabalho trata-se de uma pesquisa exploratória por meio de revisão de literatura uma vez que busca expandir o conhecimento a respeito da gestão de defeitos a partir de pesquisas na literatura presente principalmente nas bases científicas e revistas da área de Engenharia de Software (VERGARA, 2000).

Para realizar essa pesquisa e direcionar a seleção dos artigos considerados no trabalho foi utilizado o método apresentado por Costa (2014). Estes serão descritos a seguir.

A busca de trabalhos relacionados ao tema foi realizada em 2016 nas bases de consultas da CAPES. As bases IEEE Xplore, Science Direct e ACM Digital Library foram as principais escolhidas por serem repositórios de referência de trabalhos da área de ciências exatas, principalmente na área de tecnologia. Foi consultada também a revista Engenharia de Software Magazine já que o tema pertence a área de Engenharia de Software.

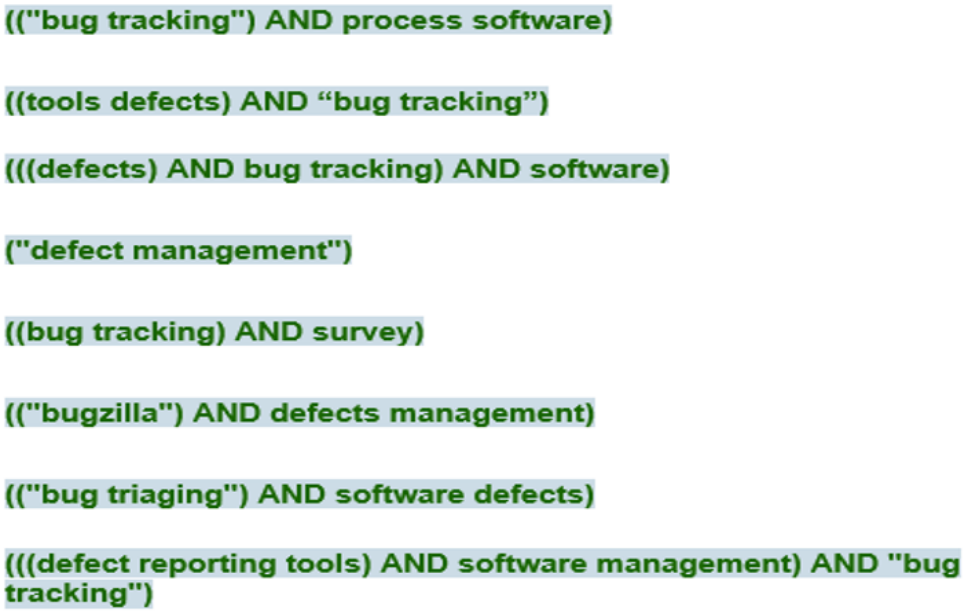
Alguns critérios foram definidos com o intuito de melhorar as pesquisas relacionadas aos artigos, como por exemplo:

- Artigos com no máximo 8 anos de publicação, período no qual intensificou-se as pesquisas sobre gerenciamento de defeitos;
- Artigos com idiomas em português ou inglês;
- Artigos científicos publicados.

As palavras de buscas utilizadas nas pesquisas foram definidas através de uma pré-analise feita de artigos inicialmente pesquisados com os temas: *software defects*, *bug triaging*, *defect management*, *bug tracking* e *defect management approach*.

Destes artigos, foram selecionados aqueles que mais atendiam a abordagem do tema, e por sua vez *strings* (cadeia de caracteres) de busca foram montadas por meio das palavras-chaves retiradas destes artigos. A Figura 24 apresenta os termos utilizados nas *strings* de busca.

Figura 24 – *Strings* de busca.



Fonte: Elaborado pelo autor.

Estas *strings* não foram utilizadas nas pesquisas das revistas de engenharia de software. Pois as revistas não possuem um algoritmo de busca avançado como apresentado nas bases da CAPES. Por esse motivo apenas nas bases da CAPES utilizou-se as *strings*.

Como apresentado na Figura 24, as *strings* foram utilizadas nas bases da CAPES, essas bases por sua vez possuem algoritmos de busca bastante sofisticados, o que proporcionou uma pesquisa mais especifica. De forma lógica as *strings* foram montadas de acordo com a preferência de busca, onde palavras compostas por aspas eram pesquisadas por inteiro nos trabalhos. O algoritmo seguia a ordem das operações aritméticas para realizar suas pesquisas.

A partir dessas pesquisas ocorreram os filtros de pesquisa, e os métodos utilizados para selecionar os artigos são mostrados a seguir:

1. Leitura dos títulos de cada artigo encontrado, verificando se estava de acordo com o que condiz o assunto;
2. Após os artigos serem selecionados pelo título, uma leitura do resumo foi realizada e aqueles relevantes foram analisados na próxima etapa;
3. Os artigos que obtiveram sucesso nas duas primeiras condições passam por outra seleção, nessa etapa foi feito uma leitura da Introdução e Conclusão de cada artigo relevante, se este estivesse de acordo com o assunto era selecionado.

Todos os artigos selecionados foram lidos e avaliados por meio de notas, sendo assim categorizados de acordo com o nível de importância para o tema, como detalhado na Tabela 6 a seguir.

Tabela 6 – Critério de notas.

Notas	Critérios para definição da nota
Notas 1 e 2	- Apresenta pouco conteúdo sobre as ferramentas de gerenciamento de defeitos; - Pouca abordagem sobre gerenciamento de defeitos, tem o foco em detalhar outros assuntos da área.
Nota 3	- Abordagem mais centralizada a gestão de defeitos; - Apresenta algumas metodologias ou técnicas de gerenciamento de defeitos; - Apresenta algumas ferramentas de gerenciamento de defeitos, detalhando algumas caractéristicas além de melhorias e sugestões.
Notas 4 e 5	- Cita ferramentas de gerenciamento de defeitos com alto índice de informação; - Abordagem do artigo é focada em gerenciamento de defeitos, buscando apresentar de melhor forma o conteúdo e suas caracterítiscas; - Apresenta comparações de ferramentas de gerenciamento de defeitos; - Apresenta metodologias e técnicas de gerenciamentos de defeitos com mais detalhes.

Fonte: Elaborado pelo autor.

Em resultados na seção 5.1 será apresentado como foi feito esse tipo de análise e algumas questões serão detalhadas.

5 Análise dos resultados

Nesse capítulo serão apresentados os resultados obtidos por meio da aplicação dos métodos realizados nesse trabalho, além de uma análise geral dos conceitos estudados durante a pesquisa.

5.1 Aplicação dos métodos

Após a definição da pesquisa mencionada na seção 4, foi possível colocar em prática os métodos a serem utilizados. A partir dessas pesquisas realizadas nas bases de dados da CAPES e nas revistas Engenharia de Software Magazine, obteve-se vários artigos por meio das *strings* de busca utilizadas.

Os artigos encontrados foram filtrados por diversos critérios de avaliação, seguindo passo a passo todo o método mencionado na seção 4, apenas os artigos mais relevantes foram selecionados. Na Tabela 7 pode-se ver os resultados obtidos.

Tabela 7 – Resultados da pesquisa nas bases de dados CAPES.

Fonte/Palavras-Chaves	Palavra 1	Palavra 2	Palavra 3	Palavra 4	Palavra 5	Palavra 6	Palavra 7	Palavra 8	Total
ACM (Total)	18	19	102	5	143	4	22	26	339
ACM (Relevantes)	3	3	3	3	3	2	3	2	22
ACM (Selecionados)	3	2	1	2	0	1	1	1	11
IEEE (Total)	89	5	30	65	8	8	1	1	207
IEEE (Relevantes)	15	3	6	12	1	2	1	1	41
IEEE (Selecionados)	7	1	2	5	1	1	1	0	18
Science Direct (Total/Selecionados)	1	0	0	0	0	0	0	0	1
Total	107	24	132	70	151	12	23	27	546
Total (Relevantes)	18	6	9	15	4	4	4	3	63
Total (Selecionados)	10	3	3	7	1	2	2	1	30

Fonte: Elaborado pelo autor.

Como mostrado na Tabela 7, foi realizado uma busca em cada base de dados da CAPES utilizando cada *string* de busca (Tabela 8), a tabela apresenta o número total de todos os artigos encontrados em suas respectivas bases, assim como, os artigos relevantes e os artigos selecionados de cada uma.

O uso das *strings* foi realizado em todas as bases no período de oito anos e como pode-se notar resultou em um número razoável de artigos, totalizando em 546.

Tabela 8 – *Strings* de busca.

Palavras	*Strings* de busca
Palavra 1	(bug tracking) and process software
Palavra 2	(tools defects) and “bug tracking”
Palavra 3	(((defects) and bug tracking) and software)
Palavra 4	(“defect management”)
Palavra 5	(bug tracking) and survey
Palavra 6	(“bugzilla”) and defects management
Palavra 7	(bug triaging) and software defects
Palavra 8	((defect reporting tools) and software management) and “bug tracking”

Fonte: Elaborado pelo autor.

Após a realização da pesquisa em cada base houve uma filtragem dos artigos em busca dos melhores qualificados e aderentes ao assunto, aplicando os métodos de filtro especificados na seção 4.

O primeiro método a ser aplicado foi a leitura dos títulos na qual os artigos foram reorganizados para logo em seguida aplicar um segundo filtro, onde os resumos foram lidos e avaliados de acordo com o seu conteúdo escolhendo aqueles que foram mais relevantes, o que resultou na escolha de 63 artigos.

Com esses artigos relevantes um terceiro filtro foi utilizado, onde uma leitura da introdução e conclusão de cada artigo foi realizada e avaliada, se estes por sua vez estivessem qualificados adequadamente ao assunto desse trabalho seriam selecionados. Após a filtragem foram finalmente selecionados 30 artigos.

Depois que os artigos foram selecionados uma outra avaliação foi realizada com base em notas de 1 a 5 dadas a cada artigo a respeito do nível de relação e importância do mesmo para a pesquisa. O grau de importância de cada nota é relacionado a qual contribuição e relação cada artigo teve para este trabalho, como mostrado na Tabela 9, onde 5 significa o maior grau de importância e relação com o trabalho apresentado.

Tabela 9 – Avaliação de qualidade dos artigos.

Fonte/Notas	Nota 1	Nota 2	Nota 3	Nota 4	Nota 5	Total
ACM	0	4	5	1	1	11
IEEE	0	7	6	2	3	18
Science Direct	0	0	1	0	0	1
Total	0	11	12	3	4	19

Fonte: Elaborado pelo autor.

Observando a tabela pode-se notar que em geral os trabalhos apresentaram pontuações satisfatórias, uma vez que a maioria destes obtiveram suas notas acima da média.

É importante ressaltar que os artigos que obtiveram notas abaixo de 3 não podem ser considerados de baixa qualidade, apenas não apresentaram um bom conteúdo para o propósito deste trabalho. Quanto aos demais que conseguiram notas superiores a 3 foram efetivamente utilizados nesta pesquisa, totalizando um resultado de 19 artigos.

Nas revistas de Engenharia de Software Magazine os resultados obtidos foram com base em uma busca de termos ou palavras-chaves relacionadas ao assunto do trabalho sendo elas: *software defects*, *bug triaging*, *defect management*, *bug tracking* e *defect management approach*. Não utilizou-se as *strings* de busca (Tabela 8) pelo fato de não possuírem um algoritmo de busca sofisticado como nas bases da CAPES.

5.2 Análise das pesquisas

Nesse trabalho foi possível perceber que as empresas estão cada vez mais na busca por qualidade de software. Consumidores de produtos de software hoje em dia esperam sempre um alto nível de qualidade quando adquirem seus produtos. Isso de fato é um grande problema para a engenharia de software, pois garantir alta qualidade requer um desempenho enorme ao softwares.

Com a complexidade que os softwares vêm adquirindo com o passar dos anos, torna-se difícil de controlar os defeitos, para isso uma boa gestão dos defeitos podem minimizar as ocorrências de tais fatores e assim conseguir melhor qualidade do produto.

A qualidade de um software pode ser definida como qualquer requisito necessário que o software deverá obter para que o produto atenda as necessidades do consumidor, ou seja, minimizar os defeitos em um projeto diminui os custos e garante qualidade ao software.

Para qualquer software o necessário é reduzir seus custos e manter o nível elevado de qualidade, como em qualquer outro tipo de produto. Porém reduzir esses custos com a produção do software é uma tarefa bastante complicada e testes de software é um processo indispensável em qualquer projeto.

A engenharia de software sempre buscou soluções para minimizar o tempo e o custo de um projeto, e com auxilio da gestão de defeitos é possível ganhar produtividade, reduzindo o tempo gasto na correção dos defeitos encontrados no software.

As empresas eram acostumadas a utilizar planilhas manuais para gerenciar seus dados e reportar os defeitos encontrados no software. Método considerado robusto e cansativo, o que levou a engenharia de software a desenvolver ferramentas de apoio para a gestão dos defeitos, na qual foi possível relacionar de forma simultânea testadores e desenvolvedores.

Porém métodos tradicionais para reportar defeitos ainda são utilizados, mesmo com o auxilio das ferramentas os desenvolvedores responsáveis pela execução e correção dos defeitos dependem ainda das informações fornecidas pelos usuários, informações que as vezes podem ser omitidas ou reportadas de maneira incorreta, ocasionado perda de tempo no

processo de desenvolvimento.

Pelo fato das informações serem reportadas de forma manual é normal que ocorram falhas, pois é algo humano. Devido essas condições algumas melhorias para os sistemas de rastreamento de defeitos em busca de uma melhor gestão dos defeitos seriam considerados como, por exemplo, a obtenção dos dados de forma automática, reduzindo assim a perda de tempo e produtividade para a equipe de desenvolvimento. Com essas melhorias o volume das informações seriam reduzidos, além da melhoria de qualidade das informações reportadas.

Para a gestão de defeitos existem diversas ferramentas, porém este trabalho levantou algumas delas: Mantis, TestLink, Bugzilla, In*Bug, iTracker e JIRA.

O Mantis é um rastreador de defeitos muito útil na gestão, é de código aberto, desenvolvido em PHP. Seus usuários são capazes de gerir seus projetos e colaborar com a equipe de desenvolvimento por meio de relatórios. Mantis é uma ferramenta que pode ser utilizada para identificar, registrar, reportar e acompanhar os defeitos encontrado em todo o processo de desenvolvimento do software.

TestLink é uma ferramenta de código aberto para gestão de testes como um todo. Além de gerenciar defeitos, o TestLink permite cadastrar planos e casos de teste e ainda controlar a sua execução. Com o TestLink é possível que as equipes de testes trabalhem de forma simultânea mesmo em locais distintos. Por ser uma ferramenta *web*, possui diversos níveis de acesso, além disso analistas de testes podem gerar as especificações de testes que outras equipes poderão executar. Outra característica interessante da ferramenta TestLink é o controle de execuções, que gera uma base de dados dos testes nas quais a aplicação foi submetida. Essa ferramenta pode ser integrada a outras ferramentas de gestão de defeitos, possibilitando cadastrar os defeitos e associa-los aos casos de teste.

Outra ferramenta bastante utilizada por desenvolvedores é o Bugzilla, essa ferramenta é um sistema de rastreamento de defeito que permite usuários criarem perfis de acesso e cadastrar defeitos, é uma ferramenta de código aberto. Bugzilla foi projetado para ajudar a gerenciar o desenvolvimento de software, fornecendo melhorias na comunicação da equipe. O Bugzilla busca sempre ajudar sua equipe de desenvolvimento a se organizar e comunicar de forma eficaz. Possui diversas características que favoreceu várias empresas, tornando-se bastante popular entre os sistemas de rastreamento. Essa ferramenta contém uma avançada pesquisa de busca, notifica os usuários por meio de mensagens via e-mail relatando qualquer mudança ocorrida no projeto, possui relatórios de defeitos, suporta vários formatos de listas de defeitos como CSV, XML entre outros.

Dentre as ferramentas de gestão de defeitos, podemos destacar também o software JIRA. JIRA é um sistema de rastreamento de defeito que permite o monitoramento e acompanhamento dos projetos. É uma ferramenta projetada em JAVA e suporta diversos bancos de dados. O JIRA permite a criação de vários projetos e cada projeto com diferentes tipos de tarefas, pode-se criar diversos *workflows* para controlar o fluxo de trabalho de cada tarefa. Esses *workflows* definem o fluxo de execução e as mudanças que cada tarefa executa em cada estado.

Além dessas ferramentas pode-se citar mais duas outras: iTracker e o In*Bug. iTracker é outra ferramenta de gerenciamento de defeitos, é de código aberto e foi construído pela linguagem JAVA. Destaca-se por ser uma ferramenta simples, fácil de usar, personalizável, fácil de integrar, rápida e escalonável para qualquer tipo de projeto.

Já o In*Bug, é uma ferramenta de gerenciamento de defeitos com o foco em relatórios de defeitos. Essa ferramenta facilita a compreensão dos defeitos reportados por meio de seus relatórios com intuito de facilitar a vida dos desenvolvedores. Trazendo mais interatividade e compreensão dos defeitos nos relatórios.

Uma crítica comum em sistemas de rastreamento de defeitos é a forma na qual seus relatórios são apresentados. Na maioria dos sistemas de rastreamento esses relatórios se tornam complexos, para alguns usuários isso é bastante prejudicial para o desenvolvimento do projeto.

Os relatórios de defeitos devem ser explicativos, simples e interativos. Os desenvolvedores devem utilizar esses relatórios para poder extrair de forma rápida e resumida toda a informação necessária para corrigir os defeitos, ganhando tempo. De uma forma em geral os sistemas de rastreamento trazem certas vantagens para a gestão de defeitos, pois os relatórios servem de base para guardar as informações, sendo assim um histórico de dados para os desenvolvedores.

Para uma equipe de desenvolvimento que contém usuários com diversos perfis hierárquicos, a utilização de ferramentas de rastreamento torna-se um ponto chave para garantir a qualidade de seu software. Em um projeto sem nenhuma ferramenta de rastreamento de defeito, é difícil de controlar e manter todas as informações históricas do seus defeitos e até mesmo a forma que foram resolvidos. Sem esse histórico até mesmo um dos defeitos mais simples poderá reaparecer, pois não irá possuir nenhuma informação de como ele surgiu, ou até mesmo de como corrigi-lo.

Por não haver históricos de dados, defeitos recursivos surgem em projetos. Esses defeitos são definidos como recursivos pois eles voltam a vida mesmo que se tenha certeza de que já foram resolvidos. Em projetos que não adotam as ferramentas de rastreamento isso é bastante comum.

Projetos de softwares que utilizam de sistemas de rastreamento podem facilmente identificar quando foi a última vez que ocorreu um defeito, o que foi feito e porque não deveria estar acontecendo novamente.

Por isso é muito importante haver relatórios de defeitos integrados aos sistemas. De certo modo armazenar esses defeitos criando um histórico de dados aumenta a produtividade de desenvolvimento da equipe, ou seja, será mais fácil de controlar e corrigir esses defeitos.

6 Considerações finais

Este trabalho teve o propósito de realizar um levantamento bibliográfico amplo, em bases de dados científicas pertinentes a área que pudesse gerar um arcabouço de informações relevantes para profissionais da área, alunos e professores, apresentando os conceitos fundamentais de gerenciamento de defeitos, além de mostrar as principais características de algumas ferramentas de gerenciamento de defeitos.

As pesquisas realizadas no trabalho foram extraídas das bases de artigos científicos da CAPES, utilizando-se somente artigos selecionados por meio de métodos de pesquisas como mostrado no capítulo 4. Na maioria dos artigos selecionados não foram abordados as ferramentas de gestão de defeitos, o que é abordado mais em revistas de engenharia de softwares. Os artigos por sua vez abordam sobre os conceitos de gestão de defeitos, sistemas de rastreamento, melhorias em sistemas de rastreamento, duplicação de defeitos, relatórios de defeitos, entre outros. Conceituando a importância de gerenciamento de defeitos na engenharia de software.

Por meio dessas pesquisas, pode-se notar que os sistemas de rastreamento de defeitos estão assumindo um papel cada vez mais importante no processo de desenvolvimento de software. Equipes de desenvolvimento buscam sempre atingir qualidade em seus produtos, com o uso dessas ferramentas de gestão.

Uma crítica em se tratando de sistemas de rastreamento é a falta de um processo automatizado para coleta das informações, além disso outros fatores de melhorias em sistemas de rastreamento poderiam surgir, como por exemplo, relatórios de defeitos que as vezes são complexos o que dificulta a correção dos desenvolvedores, melhor interação das ferramentas com os desenvolvedores, fornecer opção de anexo de informações multimídia em relatórios de defeitos entre outras funcionalidades como apresentado na seção 3.3.

Este trabalho proporciona diversas opções para trabalhos futuros, uma opção seria a implementação de um sistema de rastreamento de defeitos com base em todo o conceito de gerenciamento de defeitos, focando principalmente nos pontos de melhoria em busca de uma ferramenta melhor e mais viável do que as ferramentas apresentadas.

Um outro trabalho possível, seria o estudo de outras ferramentas de gestão de defeitos não mencionadas, porém apresentando uma comparação mais prática das ferramentas estudadas.

Referências

ABAEE, G.; GURU, D. S. Enhancement of bug tracking tools; the debugger. In: *Software Technology and Engineering (ICSTE), 2010 2nd International Conference on*. [S.l.: s.n.], 2010. v. 1, p. V1–165–V1–169. Citado nas páginas 14, 15, 17 e 39.

AKINWALE, O.; DASCALU, S.; KARAM, M. Duotracker: Tool support for software defect data collection and analysis. In: *Software Engineering Advances, International Conference on*. [S.l.: s.n.], 2006. p. 22–22. Citado nas páginas 14 e 21.

ALENEZI, M.; BANITAAN, S. Bug reports prioritization: Which features and classifier to use? In: *Machine Learning and Applications (ICMLA), 2013 12th International Conference on*. [S.l.: s.n.], 2013. v. 2, p. 112–116. Citado nas páginas 17, 18 e 19.

BARTIé, A. *Garantia da qualidade de software*. [S.l.: s.n.], 2002. Citado nas páginas 11 e 14.

BASILI, V. *Models and Metrics for Software Management and Engineering*. [S.l.]: Proceedings of ASME Century International Computer Technology Conference (invited paper), 1980. Citado na página 15.

BAYSAL, O.; HOLMES, R.; GODFREY, M. W. Revisiting bug triage and resolution practices. In: *User Evaluation for Software Engineering Researchers (USER), 2012*. [S.l.: s.n.], 2012. p. 29–30. Citado na página 19.

BORTIS, G.; HOEK, A. van der. Teambugs: A collaborative bug tracking tool. In: *Proceedings of the 4th International Workshop on Cooperative and Human Aspects of Software Engineering*. New York, NY, USA: ACM, 2011. (CHASE '11), p. 69–71. ISBN 978-1-4503-0576-1. Citado nas páginas 19 e 44.

CAETANO, C. Gestão de defeitos: Qualidade de software. *Engenharia de Software Magazine*, Edição Especial, p. 60–67, 2007. ISSN 1983127-7. Citado nas páginas 15, 16, 21, 22, 23 e 24.

CARROLL, J. *iTracker*. 2002. Disponível em: <http://itracker.sourceforge.net/>. Citado nas páginas 34, 35 e 36.

COSTA, T. T. *Projeto de jogos móveis para idosos: um estudo com foco na motivação para jogar*. 2014. 25-28 p. Pontifícia Universidade Católica de Minas Gerais. Citado na página 45.

IEEE Guide to Classification for Software Anomalies. *IEEE Std 1044.1-1995*, p. i–, 1996. Citado na página 15.

JIRA, A. 2004. Disponível em: <https://www.atlassian.com/software/jira/>. Citado nas páginas 36 e 37.

JUST, S.; PREMRAJ, R.; ZIMMERMANN, T. Towards the next generation of bug tracking systems. In: *2008 IEEE Symposium on Visual Languages and Human-Centric Computing*. [S.l.: s.n.], 2008. p. 82–85. ISSN 1943-6092. Citado nas páginas 11, 12, 41 e 43.

KORHONEN, K.; SALO, O. Exploring quality metrics to support defect management process in a multi-site organization - a case study. In: *2008 19th International Symposium on Software Reliability Engineering (ISSRE)*. [S.l.: s.n.], 2008. p. 213–218. ISSN 1071-9458. Citado na página 15.

KSHIRSAGAR, A. P.; CHANDRE, P. R. Issue tracking system with duplicate issue detection. In: *Proceedings of the Sixth International Conference on Computer and Communication Technology 2015*. New York, NY, USA: ACM, 2015. (ICCCT '15), p. 41–45. ISBN 978-1-4503-3552-2. Citado nas páginas 21, 28, 37 e 43.

LINTULA, H.; KOPONEN, T.; HOTTI, V. Exploring the maintenance process through the defect management in the open source projects - four case studies. In: *Software Engineering Advances, International Conference on*. [S.l.: s.n.], 2006. p. 53–53. Citado nas páginas 19 e 20.

MAIMON, O.; ROKACH, L. *Data Mining and Knowledge Discovery Handbook*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2005. ISBN 0387244352, 9780387244358. Citado na página 15.

MATOS, R. de F.; SILVA, L. D. da; SILVA, E. de Oliveira da. Introdução ao mantis: Desenvolvimento +gestão com agilidade. *Engenharia de Software Magazine*, v. 20, p. 41–47, 2010. ISSN 1983127-7. Citado na página 23.

NETO, O. N. B.; OLIVEIRA, S. R. B. Utilizando ferramentas de software livre para implementação de testes a partir do processo de verificação constante no modelo de referência do mps.br. *Computer on the Beach*, p. 11–20, 2013. Citado nas páginas 25, 26 e 27.

PRESSMAN, R. S. *Engenharia de software*. 6. ed. [S.l.]: Rio de Janeiro, 2002. Citado na página 14.

SANDUSKY, R. J.; GASSER, L. Negotiation and the coordination of information and activity in distributed software problem management. In: *Proceedings of the 2005 International ACM SIGGROUP Conference on Supporting Group Work*. New York, NY, USA: ACM, 2005. (GROUP '05), p. 187–196. ISBN 1-59593-223-2. Citado na página 11.

SASSO, T. D. Managing software defects. In: *Software Maintenance and Evolution (ICSME), 2014 IEEE International Conference on*. [S.l.: s.n.], 2014. p. 669–669. ISSN 1063-6773. Citado nas páginas 41 e 42.

SASSO, T. D.; LANZA, M. In*bug: Visual analytics of bug repositories. In: *Software Maintenance, Reengineering and Reverse Engineering (CSMR-WCRE), 2014 Software Evolution Week - IEEE Conference on*. [S.l.: s.n.], 2014. p. 415–419. Citado nas páginas 17, 32 e 33.

SERRANO, N.; CIORDIA, I. Bugzilla, itracker, and other bug trackers. *IEEE Software*, v. 22, n. 2, p. 11–13, March 2005. ISSN 0740-7459. Citado nas páginas 21, 34, 35 e 39.

TESTLINK. 2005. Disponível em: <http://testlink.sourceforge.net/docs/docs/features.php>. Citado na página 24.

THUNG, F.; KOCHHAR, P. S.; LO, D. Dupfinder: Integrated tool support for duplicate bug report detection. In: *Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering*. New York, NY, USA: ACM, 2014. (ASE '14), p. 871–874. ISBN 978-1-4503-3013-8. Citado nas páginas 43 e 44.

TOLEDO, J. V.; DAIBERT, M. S.; ARAúJO, M. A. P. Gerenciamento de defeitos em projetos de software: Mps.br. *Engenharia de Software Magazine*, v. 22, p. 52–60, 2010. ISSN 1983127-7. Citado nas páginas 28, 29, 30 e 31.

VERGARA, S. C. *Projetos e relatórios de pesquisa em administração*. 3. ed. [S.l.]: São Paulo, 2000. ISBN 85-224-2623-6. Citado na página 45.

WEERD, I. van de; KATCHOW, R. On the integration of software product management with software defect management in distributed environments. In: *Software Engineering Conference in Russia (CEE-SECR), 2009 5th Central and Eastern European*. [S.l.: s.n.], 2009. p. 167–172. Citado na página 11.

YUSOP, N. S. M.; GRUNDY, J.; VASA, R. Reporting usability defects: Limitations of open source defect repositories and suggestions for improvement. In: *Proceedings of the ASWEC 2015 24th Australasian Software Engineering Conference*. New York, NY, USA: ACM, 2015. (ASWEC ' 15 Vol. II), p. 38–43. ISBN 978-1-4503-3796-0. Citado nas páginas 16, 17, 38, 39, 41 e 42.

ZIMMERMANN, T. et al. Improving bug tracking systems. In: *Software Engineering - Companion Volume, 2009. ICSE-Companion 2009. 31st International Conference on*. [S.l.: s.n.], 2009. p. 247–250. Citado nas páginas 17, 40 e 42.