UNIVERSIDADE FEDERAL DO ESTADO DO RIO DE JANEIRO
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA

Relatórios Técnicos
do Departamento de Informática Aplicada
da UNIRIO
n°0005/2009

Mineração da Base de Dados de Defeitos de Software

Fernando de Castro Netto
Márcio de Oliveira Barros
Fernanda Araujo Baião

Departamento de Informática Aplicada

UNIVERSIDADE FEDERAL DO ESTADO DO RIO DE JANEIRO
Av. Pasteur, 458, Urca - CEP 22290-240
RIO DE JANEIRO – BRASIL

Relatórios Técnicos do DIA/UNIRIO, No. 0005/2009
Editor: Prof. Sean W. M. Siqueira Março, 2009

Mineração da Base de Dados de Defeitos de Software

Fernando de Castro Netto, Márcio de Oliveira Barros, Fernanda Araujo Baião

Depto de Informática Aplicada – Universidade Federal do Estado do Rio de Janeiro (UNIRIO)
{fernando.netto, marcio.barros, fernanda.baiao} @uniriotec.br

Abstract. Even if a development team uses the best Software Engineering practices to produce high-quality software, end users may find defects that were not previously identified during the software development life-cycle. These defects must be fixed and new versions of the software incorporating the patches that solve them should be released. So, the project manager must allocate developers to error correction tasks and establish the schedule of these tasks. This work proposes a method to allocate the developers and define the schedule of correction tasks based on data-mining techniques applied in the software defects database. This approach was evaluated using a subset of the Eclipse bug database.

Keywords: software defects, error correction task allocation, data-mining, Bugzilla, Eclipse

Resumo. Mesmo que uma equipe utilize as melhores práticas que a Engenharia de Software oferece para a garantia de qualidade no desenvolvimento de software, é provável que o cliente descubra defeitos no software. Estes defeitos devem ser corrigidos e novas versões do software com as respectivas correções devem ser liberadas. De posse do conjunto de defeitos reportados, o gerente do projeto de software deve alocar os desenvolvedores para as atividades de correção e estabelecer o seqüenciamento destas atividades. Este trabalho propõe uma técnica para alocação e seqüenciamento das atividades de correção de defeitos baseada na mineração da base de dados de defeitos de software. A proposta sugerida foi analisada utilizando os dados históricos dos defeitos reportados para o projeto Eclipse.

Palavras-chave: defeitos de software, alocação das tarefas de correção de defeitos, mineração de dados, Bugzilla, Eclipse

Sumário

1 Introdução

Defeitos de software podem ser definidos como desvios nas especificações encontrados pelos usuários finais após a implantação do sistema em produção (Pressman, 2005). Mesmo que uma equipe utilize as melhores práticas que a Engenharia de Software oferece para a garantia de qualidade no desenvolvimento de software, é provável que o cliente descubra defeitos no software. No contexto deste trabalho, será utilizado o termo bug como sinônimo de defeito de software, enquanto a expressão "bug fix" corresponde à atividade de correção de um defeito de software.

Ferramentas conhecidas como Bug Reporting Tools são freqüentemente utilizadas para facilitar a comunicação entre as partes envolvidas (usuários, membros da equipe de garantia da qualidade, desenvolvedores, gerentes de projetos) no processo de correção de defeitos de software (Anvik et al, 2006) que começa com a identificação do defeito pelo usuário, continua com a implementação da correção pelo desenvolvedor e termina com os testes de validação realizados pela equipe da garantia de qualidade de software (SQA – Software Quality Assurance). Estas ferramentas mantêm uma base de dados de bugs, onde cada registro de bug é composto por um conjunto de atributos que o caracterizam, tais como produto (o software onde o bug foi identificado), componente (módulo da aplicação responsável pela funcionalidade), versão (identificador da versão do software), prioridade (prioridade da correção do bug) e severidade (nível de impacto do bug na aplicação). Além disso, Bug Reporting Tools também são capazes de prover a rastreabilidade do processo de correção de defeitos de software, mantendo as informações e comentários referentes aos registros bugs, desde a identificação ao término do ciclo de vida.

Dado um conjunto de bugs reportados para um determinado software, o fornecedor da aplicação deve analisar estas notificações, e, nos casos onde forem confirmados os desvios nas especificações, deverá providenciar as respectivas correções que deverão ser incorporadas em versões futuras do software. Logo, o gerente de projeto deverá elaborar um plano de trabalho para corrigir os bugs reportados, especificando a matriz de alocação entre desenvolvedores x bug fix e o seqüenciamento destas atividades. Como pré-requisitos para estabelecer este plano, o gerente de projeto precisa analisar a experiência de cada desenvolvedor de forma a estabelecer a melhor alocação das tarefas e estimar a duração das atividades para definição do cronograma. Um planejamento ad-hoc pode contribuir para o aumento do prazo e do custo das correções dos defeitos.

O objetivo principal deste trabalho é propor uma metodologia para auxiliar o gerente de projeto no planejamento das correções de defeitos de software baseada no uso de técnicas de mineração de dados. Descreve-se o processo de construção de um datawarehouse para armazenar os registros de defeitos de software, cujos dados são extraídos das bases de dados de Bug Reporting Tools. Utilizando os recursos de tabelas e gráficos dinâmicos do MS Excel[1], serão demostrados exemplos de análises complexas processadas no datawarehouse. As propostas para alocação de desenvolvedores e estimativa da duração das atividades consistem no uso de técnicas de data-mining para

[1] http://office.microsoft.com/excel

classificação dos bugs e previsão do tempo de correção de novos bugs baseado na análise da série histórica dos tempos de correção de defeitos da mesma classe.

A análise da proposta utilizará como referência os bugs reportados para a comunidade de desenvolvimento do Eclipse[2]. Os dados foram capturados a partir do portal de acesso do software Bugzilla[3] que é utilizado para o tratamento de defeitos de software encontrados nos projetos do Eclipse.

Este trabalho está organizado através das seguintes seções: a seção 2 descreve alguns trabalhos relacionados. A seção 3 apresenta uma breve descrição dos Bug Reporting Tools. A seção 4 descreve o projeto do datawarehouse de defeitos de software. A seção 5 descreve a proposta para aplicação de técnicas de data-mining no datawarehouse de defeitos de software. A seção 6 apresenta um estudo de caso para avaliar a proposta de solução enquanto que a seção 7 apresenta as considerações finais e as conclusões do trabalho.

2 Trabalhos Relacionados

Anvik et al (Anvik et al, 2006) analisam o problema de alocação de desenvolvedores para as tarefas de correção de defeitos de software. A proposta apresentada neste trabalho consiste em analisar uma base de dados de defeitos de software usando um algoritmo de machine learning para identificar os tipos de bugs que cada desenvolvedor costuma corrigir. Dessa forma, os desenvolvedores seriam alocados para resolver novos bugs que estão associados a categorias nos quais eles estão habilitados a corrigir. Neste trabalho, cada registro de bug da base de dados de defeitos é tratado como um documento texto, sendo associado a um conjunto de categorias, onde cada categoria representa um desenvolvedor), utilizando técnicas de categorização de textos. Os pesquisadores avaliaram a solução proposta utilizando como referência a base de dados de defeitos dos projetos Eclipse, Firefox[4] e GCC[5], onde os testes registraram níveis de precisão de 57%,64% e 6%, respectivamente.

Weiss et al (Weiss et al, 2007) descrevem uma técnica para prever o esforço necessário para corrigir bugs. Dado um novo bug reportado, o método realiza uma busca na base de dados de bugs e captura os k registros de bugs mais similares ao novo bug reportado que já foram corrigidos. O esforço para correção estimado é calculado a partir da média do tempo de correção dos k bugs similares recuperados. Utilizou-se a técnica de text similarity, onde os campos textuais com a descrições dos defeitos foram comparados para definir quais seriam os k bugs mais semelhantes ao novo bug. A proposta foi avaliada utilizando como referência a base de dados de defeitos de software do projeto JBoss[6], onde os resultados revelaram que com um grande número de bugs registrados é possível fazer previsões de esforço próximas do realizado.

Wang et al (Wang et al, 2008) apresentam uma técnica para verificar se novos bugs reportados podem ser classificados como a registros duplicados de outros bugs previ-

[2] http://www.eclipse.org/

[3] https://bugs.eclipse.org/bugs/

[4] http://br.mozdev.org/

[5] http://gcc.gnu.org/

[6] http://www.jboss.org/

amente cadastrados na base. Cada registro de bugs é tratado como um documento texto que, a partir de técnicas de processamento de linguagem natural (NLP – Natural Language Processing), é mapeado em um vetor de palavras chave. Dado um novo registro de bug, são calculados os índices de similaridade do novo registro com cada um dos demais bugs cadastrados. Caso algum dos resultados calculados for igual ou superior a limite pré-definido, então o novo bug será considerado como registro duplicado. Os experimentos conduzidos utilizando como referência os bugs do Eclipse e Firefox apresentaram níveis de precisão de 67% e 93% respectivamente.

3 Bug Reporting Tools

Grandes projetos de software frequentemente utilzam algum Bug Reporting Tool onde desenvolvedores, membros da equipe de SQA e usuários finais podem reportar e consultar a resolução dos defeitos de um determinado software. Estas ferramentas estabelecem um canal de comunicação entre os diferentes grupos envolvidos no desenvolvimento do software e ajudam a melhorar a qualidade do software, promovendo a percepção dos defeitos que são encontrados no software e, consequemente, as respectivas correções.

BugZilla é um dos Bug Reporting Tools mais utilizados pela indústria de software. Esta aplicação mantem uma base de dados dos registros de bugs para projetos de software e facilita a comunicação e o controle entre os membros da equipe de qualidade de software, os desenvolvedores e os usuários finais. Permite que qualquer indivíduo com um login e senha reporte um novo registro de bug ou faça comentários sobre um registro de bug exsistente.

BugZilla mantém a rastreabilidade do processo de correção de um defeito através do ciclo de vida do bug, desde o momento da identificação até o término com a resolução atribuída, incluindo a alocação do desenvolvedor e a construção de pacote que inclui o bug-fix. Quando um usuário final ou testado encontra um defeito no software, cria-se um novo registro no BugZilla. Periodicamente, membros da equipe de SQA recuperam os novos registros criados e analisam cada um para verificar se os problemas notificados podem ser classificados como defeitos do software. Os defeitos que passarem por este crivo devem ser corrigidos por algum desenvolvedor ao passo que as respectivas correções deverão ser validadas por membros da equipe de SQA. Se a solução implementada for validada, então o bug-fix deverá ser incorporado em alguma versão futura do software.

O processo descrito acima é representado no BugZilla como uma máquina de estados, conforme apresentado na Figura 1. Inicialmente, todo registro é associado ao estado NEW. Quando um desenvolvedor é alocado para tratar o defeito reportado, o registro muda para estado ASSIGNED. Depois que desenvolvedor providenciou a solução, o registro migra para o estado RESOLVED. Depois que a solução é validada pela equipe de SQA, o registro migra para o estágio VERIFIED e, consequemente, para CLOSED quando o bug-fix é disponibilizado para o usuário final. Se a solução não for satisfatória, o registro retorna para o estágio ASSIGNED. Por fim, um bug tratado pode sempre retornar como REOPENED mais tarde por um usuário final insatisfeito ou por algum membro da equipe de SQA. Conforme observado na Figura 1, existem 5 tipos de resolução de bugs: FIXED (quando a correção implica na alteração do código-fonte), DUPLICATE (trata-se de um registro duplicado de outro bug), WORKSFORME (não

foi possível reproduzir o bug), INVALID (não se trata de um defeito de software) e WONTFIX (o problema não será corrigido).

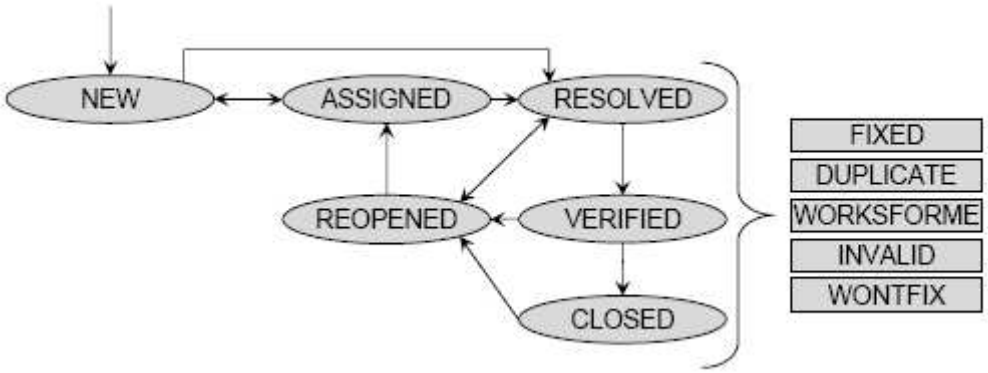

Figura 1: ciclo de vida de um registro de bug

4 Datewarehouse de Defeitos de Software

Os Bug Reporting Tools de grandes projetos de software geralmente mantêm um volume muito grande de registros de defeitos de software que são armazenados em modelos de dados relacionais específicos de cada aplicação. Estas ferramentas permitem a exportação dos dados para arquivos de diferentes formatos, tais como CSV, XML, HTML, etc. Em alguns casos, estes dados também podem ser capturados via Web Services. Logo, para cada tipo de Bug Reporting Tool, torna-se necessária a criação de um programa de Extração, Transformação e Carga (ETL – Extract Transform Load) dos dados específicos.

Contudo, conjunto de atributos e os respectivos domínios de valores são muito semelhantes. Sendo assim, este trabalho propõe a construção de um modelo multidimensional genérico baseado no esquema estrela (Kimball, 1997) para armazenar os registros de defeitos de software, conforme apresentado na Figura 2.

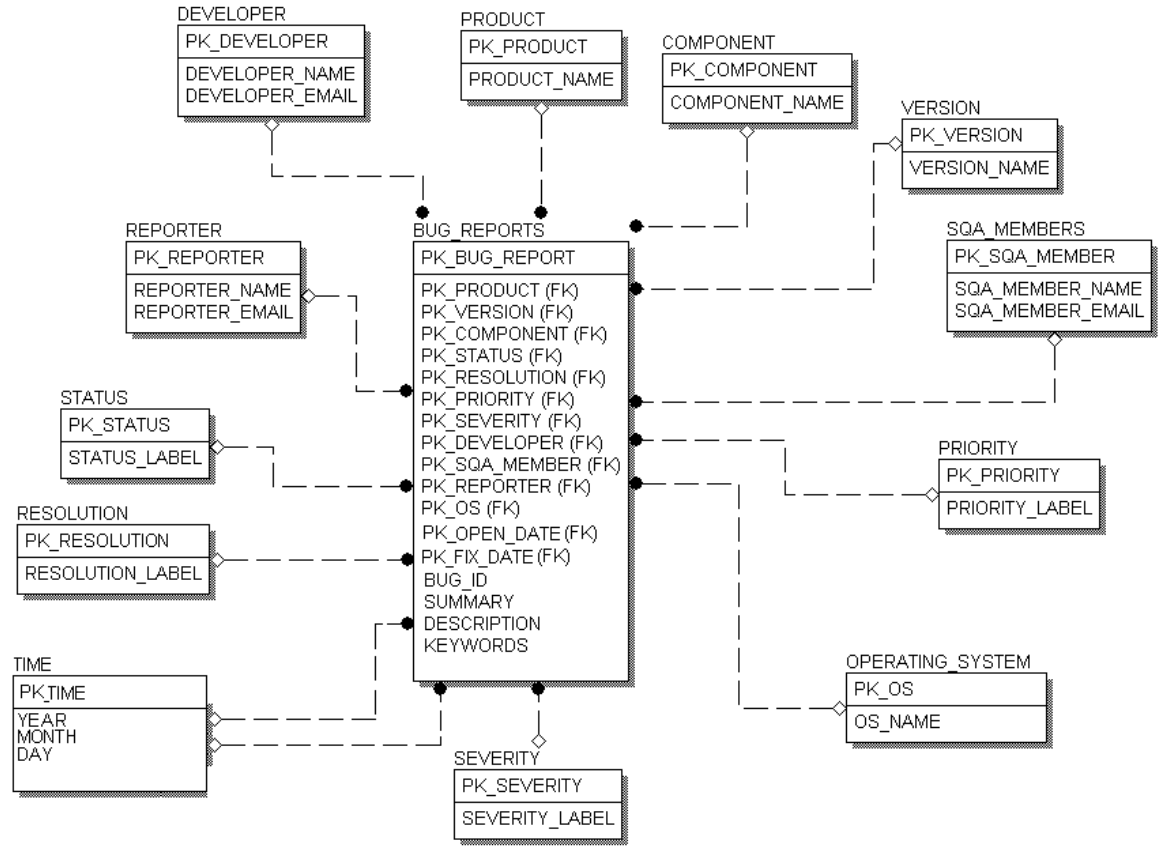

Figura 2: esquema multidimensional para registros de defeitos de software

No modelo apresentado, há uma tabela de fatos cercada por 12 tabelas de dimensão cujas chaves primárias são surrogate keys (Kimball & Becker, 2006). A tabela DEVELOPER mantém o cadastro de desenvolvedores, enquanto que as tabelas SQA_MEMBERS e REPORTER mantêm os cadastros de membros da equipe de SQA e de usuários respectivamente. A tabela PRODUCT mantém a lista de projetos de software. As tabelas COMPONENT e VERSION armazenam as listas de componentes e identificadores de versões dos softwares. A tabela STATUS mantém o domínio de estados que registro de bug pode assumir (NEW, ASSIGNED, RESOLVED, REOPENED, VERIFIED, CLOSED), ao passo que as tabelas PRIORITY e SEVERITY mantêm os domínios de prioridade de correção (P1, P2, P3, P4, P5) e impacto do bug no sistema (MINOR, TRIVIAL, NORMAL, BLOCKER, CRITICAL, MAJOR), respectivamente. As listas das possíveis resoluções (FIXED, DUPLICATE, WORKSFORME, INVALID, WONTFIX) e os sistemas operacionais são mantidas respectivamente nas tabelas RESOLUTION e OPERATING_SYSTEM. Todos os fatos armazenados na tabela BUG_REPORTS são não aditivos: BUG_ID (identificador do registro de bug no sistema de origem), SUMMARY (resumo da descrição do bug), DESCRIPTION (descrição completa do problema, incluindo comentários dos usuários, desenvolvedores e testadores), KEYWORDS (palavras chaves registradas pelo usuário no momento do cadastramento do bug).

5 Mineração de Dados

Diante de uma base de dados com um grande volume de registros de defeitos reportados, surgem as seguintes perguntas: "O que fazer com todos os dados armazenados?", "Como utilizar a base de dados em benefício da organização?". Para lidar com estas questões a Descoberta do Conhecimento em Base de Dados (knowledge Discovery in Databases – KDD) propõe um processo de extração de informações implicitamente contidas em uma base de dados, para prover subsídios para os tomadores de decisões acerca de alocação de recursos humanos para resolução de bugs.

De acordo com Goldschmidt e Passos (Goldschmidt & Passos, 2005), o processo de descoberta de conhecimento em base de dados é dividido em 3 etapas: Pré–Processamento, Mineração de Dados e Pós-Processamento. O Pré-Processamento consiste das tarefas de captura e armazenamento dos dados. A Mineração de Dados, que é o tópico central deste trabalho, tem como objetivo a busca efetiva de conhecimentos úteis no contexto da aplicação, que neste caso é o tratamento de defeitos reportados para a comunidade de software. A fase de Pós-Processamento nem sempre é necessária e tem como finalidade prover a avaliação do conhecimento descoberto. Para ilustrar como ocorre a descoberta do conhecimento a partir da extração de dados elementares, a Figura 3 apresenta a pirâmide que evidencia as diferenças entre dado, informação e conhecimento no contexto do trabalho.

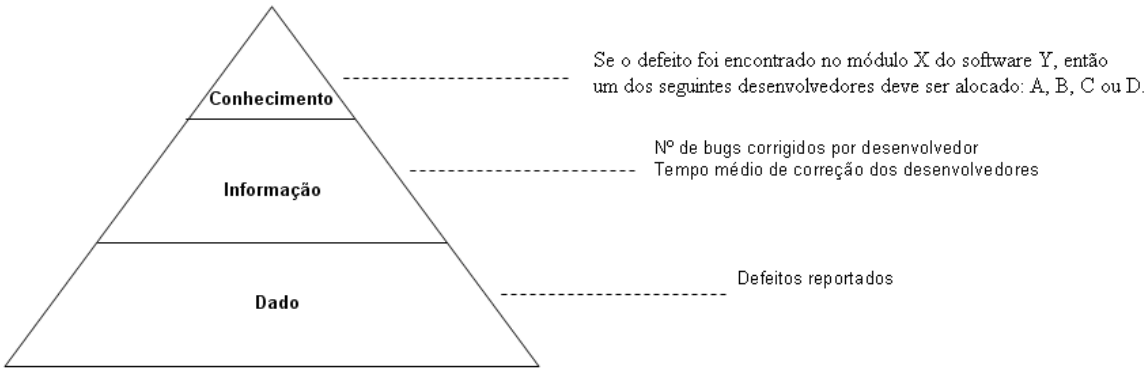

Figura 3: hierarquia entre Dado, Informação e Conhecimento

Classificação é o processo de encontrar um modelo que descreva classes diferentes de dados (Elmasri & Navathe, 2005). Uma possível classificação para os registros de bugs poderia ser o conjunto dos seguintes atributos: PRODUCT, COMPONENT e OPERATING_SYSTEM. Dessa forma, poderíamos aplicar uma técnica de classificação capaz de gerar na forma de uma árvore de decisão regras para recomendar desenvolvedores para cada classe de bugs, conforme ilustrado na Figura 4.

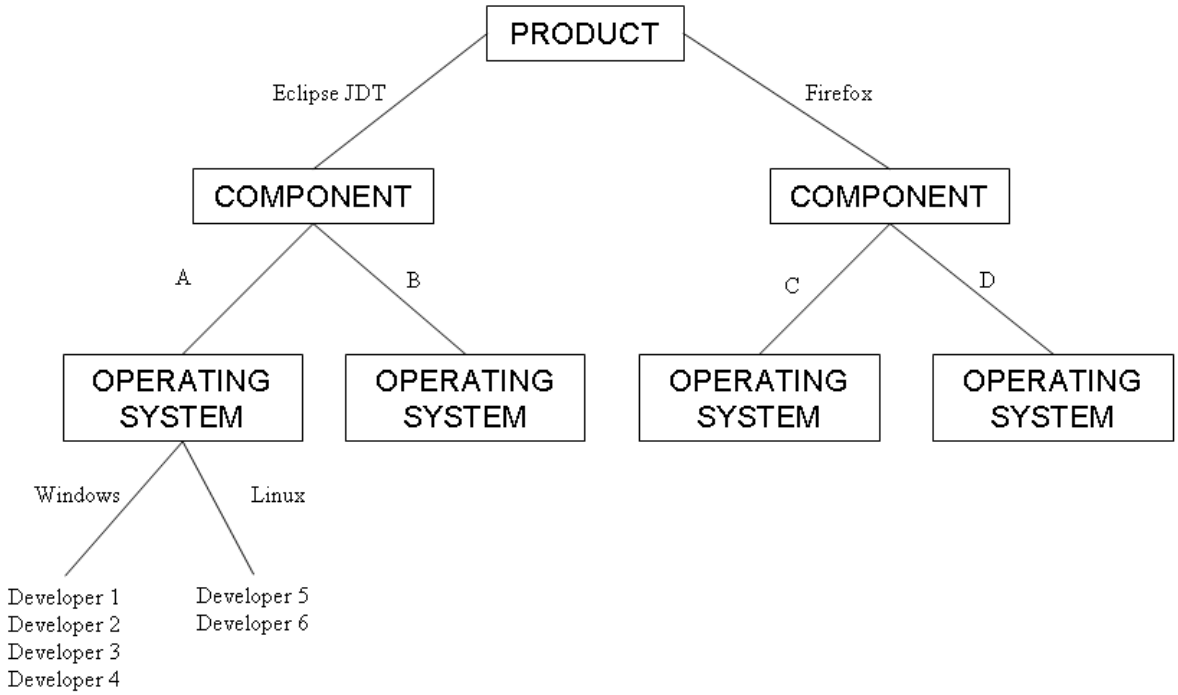

Figura 4: processo de classificação baseado em árvore de decisão

É possível agrupar os registros de bugs usando técnicas de agrupamento (clustering), cujo objetivo é colocar os registros de bugs em grupos, de tal forma que registros de um grupo sejam similares aos demais registros do mesmo grupo e diferentes daqueles dos demais grupos. Utilizando técnicas de Information Retrieval (Feldman et al, 1998) para extrair informações das descrições textuais de cada registro de bug, calcula-se o índice de similaridade para cada par de registros conforme especificado em Wang et al (Wang et al, 2008), disponibilizando o resultado na matriz de similaridades apresentada na Tabela 1.

	Bug 1	Bug 2	Bug 3	Bug 4	Bug 5	Bug 6
Bug 1	1,00	(0,21)	(0,74)	(0,89)	0,53	0,72
Bug 2	(0,21)	1,00	0,73	0,72	0,02	(0,07)
Bug 3	(0,74)	0,73	1,00	0,98	(0,14)	(0,32)
Bug 4	(0,89)	0,72	0,98	1,00	0,21	0,11
Bug 5	0,53	0,02	(0,14)	0,21	1,00	0,65
Bug 6	0,72	(0,07)	(0,32)	0,11	0,65	1,00

Tabela 1: matriz de similaridades entre bugs

Analisando a Tabela 1, vemos que os registros Bug 1, Bug 5 e Bug 6 pertencem ao mesmo grupo, assim como os registros Bug 2, Bug 3 e Bug 4 também são agrupados em outro grupo. Dado que o Bug 6 é um novo registro cadastrado, para estimar o prazo de correção pode-se utilizar como referência a média dos tempos registrados para correções de bugs similares, isto é, a média dos tempos de correção registrados para os bugs 1 e 5.

6 Estudo de Caso

Eclipse é uma comunidade de software livre cujos projetos visam a construção de plataformas abertas para desenvolvimento de aplicações, incluindo ferramentas para construção, implantação e gerenciamento de software, permeando todo ciclo de vida. O portal do BugZilla da comunidade de software livre Eclipse mantém todos os registros de defeitos reportados pelos usuários. Cada registro de defeito pode ser visualizado em diversos formatos, incluindo HTML, XML, CSV e TXT. O processo de extração dos registros dos bugs consistiu na criação de um utilitário para fazer o download dos registros do BugZilla no formato XML. No decorrer do trabalho, o utilitário criado capturou mais de 200.000 registros de bugs, contemplando a série histórica de outubro/2001 a maio/2008.

A análise dos dados foi realizada utilizando as ferramentas de tabelas e gráficos dinâmicos do Microsoft Excel. Do total de registros de defeitos do Eclipse capturados, 171.938 foram resolvidos até a presente data enquanto que 28.226 ainda aguardavam as respectivas correções. O resultado desta análise é apresentado na Tabela 2.

Situação	Status	Total
RESOLVIDO		**171.938**
	CLOSED	34.187
	RESOLVED	111.116
	VERIFIED	26.635
EM ANDAMENTO		**28.226**
	ASSIGNED	5.573
	NEW	22.101
	REOPENED	552
Total geral		**200.164**

Tabela 2: análise dos bugs reportados para a comunidade Eclipse

Sabendo que os problemas foram reportados pelos usuários dos softwares da comunidade Eclipse, é provável que quanto maior o número de registros para um determinado produto maior será a quantidade de usuários da aplicação. A partir desta hipótese, o principal software da comunidade seria o Eclipse Platform, que é o ambiente de integração para as demais aplicações desenvolvidas na comunidade. De fato, esta conclusão é pertinente, pois se trata de um pré-requisito para a execução das demais ferramentas, sendo utilizado por todos os usuários de qualquer aplicação da comunidade. A segunda ferramenta mais utilizada pelos usuários é o Eclipse Java Development Tool (JDT) que é largamente utilizado na indústria de software por progamadores Java. A Figura 5 exibe o gráfico das 15 aplicações mais utilizadas da comunidade. Esta lista de aplicações contempla cerca de 85% do total de problemas reportados.

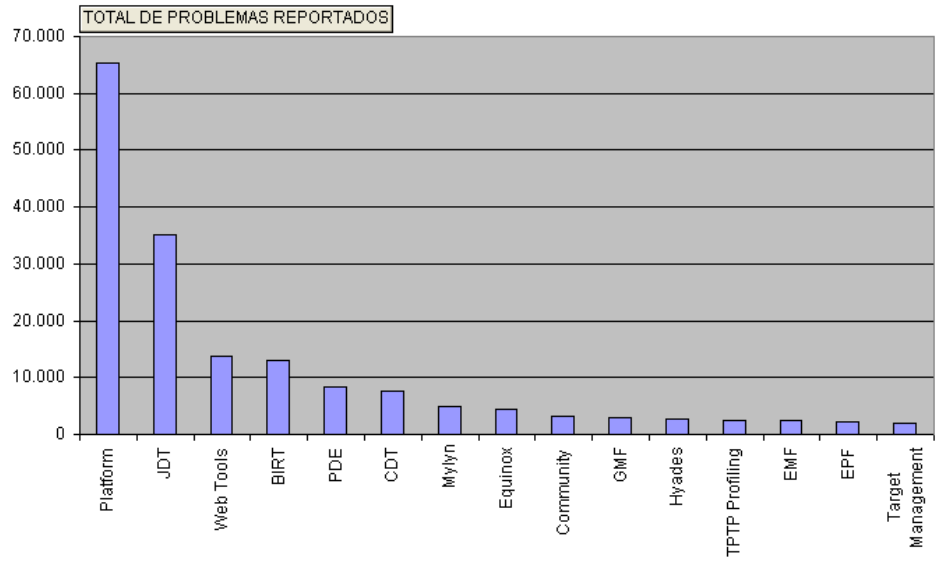

Figura 5. Software x Nº de Bugs Reportados

Outra informação que pode ser adquirida a partir da base de dados é o acompanhamento das liberações de versão ao longo do período da coleta de dados, observando as migrações de versões. A Figura 6 mostra a quantidade de bugs reportados para diferentes versões do software Eclipse Platform durante o período de coleta de dados. Note que nos primeiros anos, 2001 e 2002, a versão mais utilizada era a 2.0. Em 2003 houve o lançamento da versão 3.0, tornando-se a versão mais utilizada nesta época. O grande número de bugs reportados obrigou que versões subseqüentes fossem lançadas com correções. O gráfico nos permite concluir que a 3.4 é a versão do Eclipse Platform mais utilizada atualmente.

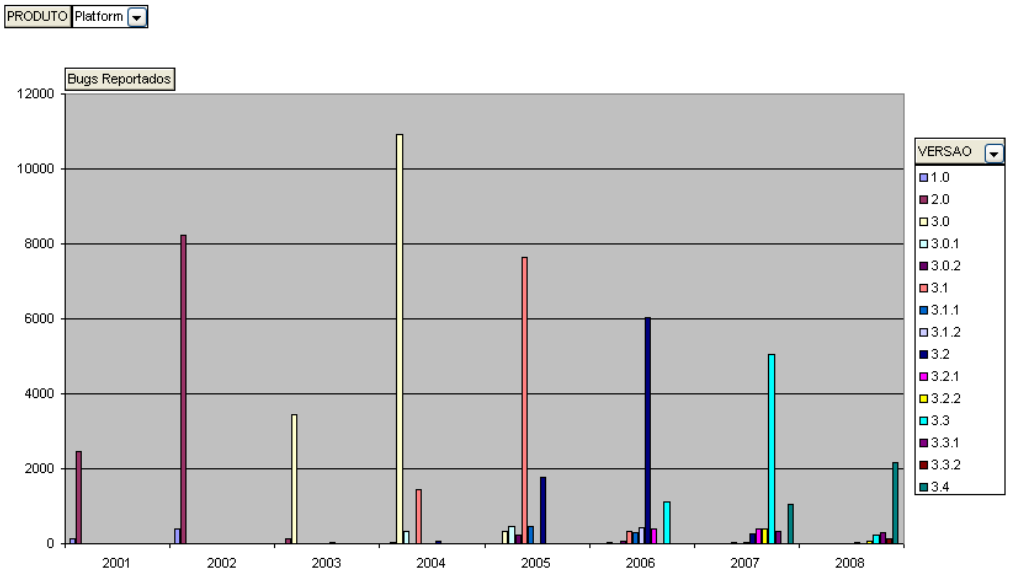

Figura 6. Ano x Nº de Bugs Reportados para o Eclipse Platform

Também é possível criar um ranking de desenvolvedores por bugs corrigidos. Foram reconhecidos 1.378 membros que corrigiram ao menos algum defeito reportado.

Desses, 258 (ou 19% do total de programadores) realizaram 80% do total de correções aplicadas. A figura 7 ilustra o gráfico do ranking de desenvolvedores por número de defeitos corrigidos.

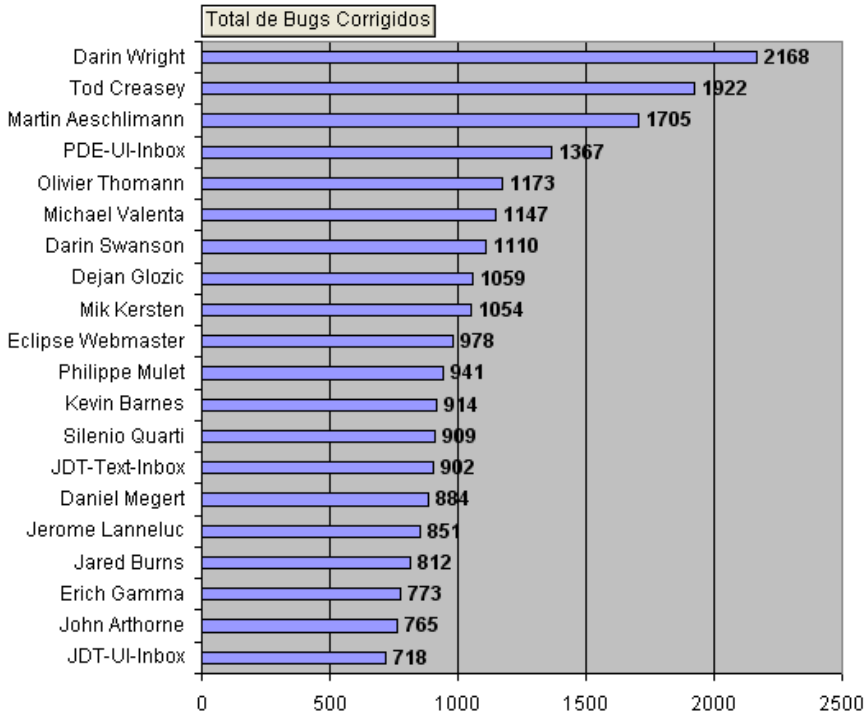

Figura 7. ranking de desenvolvedores por total de bugs corrigidos

Neste trabalho, aplicou-se a técnica de classificação para definição dos desenvolvedores para cada classe de defeito definida (PRODUTO, COMPONENTE, SISTEMA OPERACIONAL) utilizando o software Weka[7]. A instância utilizada nos testes considerou o conjunto de bugs reportados durante o ano de 2007 para os softwares Eclipse Platform (Platform) e Eclipse Java Development Tools (JDT). Foram analisados 10.569 registros de bugs e, como resultado, gerou-se uma árvore de decisão capaz de mapear para cada classe uma recomendação de desenvolvedor para correção. Seguem, a seguir, os resultados da classificação processada no Weka.

Number of Leaves : 110
Size of the tree : 119
Time taken to build model: 0.31 seconds

=== Stratified cross-validation ===		
=== Summary ===		
Correctly Classified Instances	4.970	47,02%
Incorrectly Classified Instances	5.599	52,98%

[7] http://www.cs.waikato.ac.nz/ml/weka/

Kappa statistic	0,4524	
Mean absolute error	0,0116	
Root mean squared error	0,0764	
Relative absolute error	0,6707	
Root relative squared error	0,8220	
Total Number of Instances	10.569	

Tabela 3: detalhamento do cross-validation

=== Classifier model (full training set) ===

J48 pruned tree

COMPONENTE = UI
| PRODUTO = Platform
| | SISTEMA_OPERACIONAL = Windows Vista: Tod Creasey (183.0/100.0)
| | SISTEMA_OPERACIONAL = All: Boris Bokowski (574.0/461.0)
| | SISTEMA_OPERACIONAL = Mac OS X: Kim Horne (128.0/100.0)
| | SISTEMA_OPERACIONAL = Windows XP: Tod Creasey (1454.0/1180.0)
| | SISTEMA_OPERACIONAL = Unix All: Tod Creasey (0.0)
| | SISTEMA_OPERACIONAL = Linux-GTK: Tod Creasey (39.0/32.0)
| | SISTEMA_OPERACIONAL = Linux: Paul Webster (141.0/113.0)
| | SISTEMA_OPERACIONAL = Windows 2003 Server: Kim Horne (2.0/1.0)
| | SISTEMA_OPERACIONAL = Windows 2000: Tod Creasey (62.0/46.0)
| | SISTEMA_OPERACIONAL = Windows All: Kim Horne (10.0/7.0)
| | SISTEMA_OPERACIONAL = AIX Motif: Kim Horne (1.0)
| | SISTEMA_OPERACIONAL = Mac OS X - Cocoa: Tod Creasey (0.0)
| | SISTEMA_OPERACIONAL = Windows Vista-WPF: Tod Creasey (6.0/3.0)
| | SISTEMA_OPERACIONAL = Solaris: Karice McIntyre (1.0)
| | SISTEMA_OPERACIONAL = Windows NT: Eric Moffatt (61.0/32.0)
| | SISTEMA_OPERACIONAL = Linux-Motif: Paul Webster (1.0)
| | SISTEMA_OPERACIONAL = HP-UX: Tod Creasey (0.0)
| | SISTEMA_OPERACIONAL = Solaris-GTK: Andrew Niefer (4.0/2.0)
| | SISTEMA_OPERACIONAL = Windows 98: Tod Creasey (0.0)
| | SISTEMA_OPERACIONAL = Solaris-Motif: Tod Creasey (0.0)
| | SISTEMA_OPERACIONAL = Windows CE: Tod Creasey (0.0)
| | SISTEMA_OPERACIONAL = QNX-Photon: Tod Creasey (0.0)
| PRODUTO = JDT: JDT-UI-Inbox (1118.0/588.0)
COMPONENTE = Compare: Platform-Compare-Inbox (188.0/75.0)
COMPONENTE = SWT
| SISTEMA_OPERACIONAL = Windows Vista: Steve Northover (69.0/43.0)
| SISTEMA_OPERACIONAL = All: Grant Gayed (104.0/86.0)
| SISTEMA_OPERACIONAL = Mac OS X: Silenio Quarti (211.0/124.0)
| SISTEMA_OPERACIONAL = Windows XP: Steve Northover (539.0/351.0)
| SISTEMA_OPERACIONAL = Unix All: Silenio Quarti (2.0/1.0)
| SISTEMA_OPERACIONAL = Linux-GTK: Bogdan Gheorghe (189.0/96.0)
| SISTEMA_OPERACIONAL = Linux: Bogdan Gheorghe (154.0/94.0)
| SISTEMA_OPERACIONAL = Windows 2003 Server: Steve Northover (4.0/2.0)
| SISTEMA_OPERACIONAL = Windows 2000: Steve Northover (36.0/17.0)
| SISTEMA_OPERACIONAL = Windows All: Steve Northover (17.0/9.0)
| SISTEMA_OPERACIONAL = AIX Motif: Bogdan Gheorghe (8.0/4.0)

| SISTEMA_OPERACIONAL = Mac OS X - Cocoa: Kevin Barnes (29.0/12.0)
| SISTEMA_OPERACIONAL = Windows Vista-WPF: Kevin Barnes (34.0/7.0)
| SISTEMA_OPERACIONAL = Solaris: Bogdan Gheorghe (5.0/3.0)
| SISTEMA_OPERACIONAL = Windows NT: Platform-SWT-Inbox (1.0)
| SISTEMA_OPERACIONAL = Linux-Motif: Bogdan Gheorghe (10.0/6.0)
| SISTEMA_OPERACIONAL = HP-UX: Bogdan Gheorghe (5.0/2.0)
| SISTEMA_OPERACIONAL = Solaris-GTK: Bogdan Gheorghe (11.0/4.0)
| SISTEMA_OPERACIONAL = Windows 98: Steve Northover (0.0)
| SISTEMA_OPERACIONAL = Solaris-Motif: Bogdan Gheorghe (2.0)
| SISTEMA_OPERACIONAL = Windows CE: Silenio Quarti (3.0/2.0)
| SISTEMA_OPERACIONAL = QNX-Photon: Rodney Dowdall (2.0)
COMPONENTE = User Assistance: platform-ua-inbox (476.0/66.0)
COMPONENTE = Ant: Platform-Ant-Inbox (147.0/48.0)
COMPONENTE = Core: JDT-Core-Inbox (958.0/745.0)
COMPONENTE = IDE
| SISTEMA_OPERACIONAL = Windows Vista: Platform IDE Inbox (7.0/2.0)
| SISTEMA_OPERACIONAL = All: Tod Creasey (23.0/10.0)
| SISTEMA_OPERACIONAL = Mac OS X: Tod Creasey (10.0/7.0)
| SISTEMA_OPERACIONAL = Windows XP: Tod Creasey (129.0/86.0)
| SISTEMA_OPERACIONAL = Unix All: Tod Creasey (1.0)
| SISTEMA_OPERACIONAL = Linux-GTK: Platform IDE Inbox (5.0/2.0)
| SISTEMA_OPERACIONAL = Linux: Platform IDE Inbox (31.0/13.0)
| SISTEMA_OPERACIONAL = Windows 2003 Server: Paul Webster (4.0/3.0)
| SISTEMA_OPERACIONAL = Windows 2000: Platform IDE Inbox (13.0/6.0)
| SISTEMA_OPERACIONAL = Windows All: Paul Webster (1.0)
| SISTEMA_OPERACIONAL = AIX Motif: Platform IDE Inbox (0.0)
| SISTEMA_OPERACIONAL = Mac OS X - Cocoa: Platform IDE Inbox (0.0)
| SISTEMA_OPERACIONAL = Windows Vista-WPF: Platform IDE Inbox (0.0)
| SISTEMA_OPERACIONAL = Solaris: Kim Horne (2.0/1.0)
| SISTEMA_OPERACIONAL = Windows NT: Platform IDE Inbox (0.0)
| SISTEMA_OPERACIONAL = Linux-Motif: Platform IDE Inbox (0.0)
| SISTEMA_OPERACIONAL = HP-UX: Tod Creasey (1.0)
| SISTEMA_OPERACIONAL = Solaris-GTK: Platform IDE Inbox (0.0)
| SISTEMA_OPERACIONAL = Windows 98: Platform IDE Inbox (0.0)
| SISTEMA_OPERACIONAL = Solaris-Motif: Platform IDE Inbox (0.0)
| SISTEMA_OPERACIONAL = Windows CE: Platform IDE Inbox (0.0)
| SISTEMA_OPERACIONAL = QNX-Photon: Platform IDE Inbox (0.0)
COMPONENTE = Text
| PRODUTO = Platform: Platform-Text-Inbox (310.0/16.0)
| PRODUTO = JDT: JDT-Text-Inbox (333.0/50.0)
COMPONENTE = Runtime: platform-runtime-inbox (167.0/17.0)
COMPONENTE = Debug
| PRODUTO = Platform: Platform-Debug-Inbox (583.0/249.0)
| PRODUTO = JDT: JDT-Debug-Inbox (494.0/187.0)
COMPONENTE = Doc
| PRODUTO = Platform: Platform-Doc-Inbox (39.0/12.0)
| PRODUTO = JDT: JDT-Doc-Inbox (31.0/7.0)
COMPONENTE = Search: Platform-Search-Inbox (49.0/12.0)
COMPONENTE = Team: Platform Team Inbox (232.0/88.0)
COMPONENTE = CVS: platform-cvs-inbox (285.0/70.0)
COMPONENTE = Update: Platform-Update-Inbox (279.0/14.0)
COMPONENTE = APT: Walter Harley (75.0/37.0)

COMPONENTE = Resources: Platform-Resources-Inbox (212.0/75.0)
COMPONENTE = Releng
| SISTEMA_OPERACIONAL = Windows Vista: Kit Lo (1.0)
| SISTEMA_OPERACIONAL = All: Platform-Releng-Inbox (45.0/23.0)
| SISTEMA_OPERACIONAL = Mac OS X: Platform-Releng-Inbox (4.0)
| SISTEMA_OPERACIONAL = Windows XP: Kim Moir (173.0/75.0)
| SISTEMA_OPERACIONAL = Unix All: Kim Moir (0.0)
| SISTEMA_OPERACIONAL = Linux-GTK: Platform-Releng-Inbox (4.0/1.0)
| SISTEMA_OPERACIONAL = Linux: Platform-Releng-Inbox (19.0/8.0)
| SISTEMA_OPERACIONAL = Windows 2003 Server: Kim Moir (0.0)
| SISTEMA_OPERACIONAL = Windows 2000: Kim Moir (4.0/2.0)
| SISTEMA_OPERACIONAL = Windows All: Kim Moir (0.0)
| SISTEMA_OPERACIONAL = AIX Motif: Kim Moir (0.0)
| SISTEMA_OPERACIONAL = Mac OS X - Cocoa: Kim Moir (0.0)
| SISTEMA_OPERACIONAL = Windows Vista-WPF: Kim Moir (0.0)
| SISTEMA_OPERACIONAL = Solaris: Kim Moir (0.0)
| SISTEMA_OPERACIONAL = Windows NT: Kim Moir (0.0)
| SISTEMA_OPERACIONAL = Linux-Motif: Kim Moir (0.0)
| SISTEMA_OPERACIONAL = HP-UX: Platform-Releng-Inbox (2.0)
| SISTEMA_OPERACIONAL = Solaris-GTK: Kim Moir (0.0)
| SISTEMA_OPERACIONAL = Windows 98: Kim Moir (0.0)
| SISTEMA_OPERACIONAL = Solaris-Motif: Platform-Releng-Inbox (1.0)
| SISTEMA_OPERACIONAL = Windows CE: Kim Moir (0.0)
| SISTEMA_OPERACIONAL = QNX-Photon: Kim Moir (0.0)
COMPONENTE = Website: Michael D. Elder (2.0/1.0)
COMPONENTE = WebDAV: Platform-WebDAV-Inbox (4.0)
COMPONENTE = Incubator: Tom Schindl (4.0/2.0)
COMPONENTE = Scripting: Kim Horne (1.0)

7 Considerações Finais

Os resultados apresentados neste trabalho ainda não são conclusivos. Contudo, o trabalho indica que a mineração da base de dados de defeitos de software tem potencial para prover informações pertinentes para a tomada de decisão nas organizações que desenvolvem software.

A continuidade deste trabalho sugere a validação dos resultados obtidos com a aplicação da técnica de classificação, isto é, deve-se comparar as recomendações de alocação com as alocações realizadas, verificando o índice de assertividade. Recomenda-se a aplicação técnicas de text mining baseadas nos métodos de processamento de linguagem natural e construção de vetores indexados por palavras-chave. Os registros de bugs devem ser agrupados de acordo com critérios de similaridades entre os respectivos vetores indexados por palavras-chave. O prazo de correção para cada grupo de bug pode ser estimado usando técnicas de regressão.

Referências Bibliográficas

Pressman, S., Roger (2005), **Software Engineering - A Practitioner's Approach**, 6 ed., New York, McGraw-Hill.

Anvik, J.; Hiew, L.; Murphy, G. (2006), **Who Should Fix This Bug?**, Proceedings of ICSE.

Weiss, Cathrin; Premraj, Rahul; Zimmermann, Thomas; Zeller, Andreas (2007), **How Long will it Take to Fix This Bug?**, 29th International Conference on Software Engineering Workshops (ICSEW'07).

Wang, Xiaoyin; Zhang, Lu; Xie, Tao; Anvik, John; Sun, Jiasu (2008), **An Approach to Detecting Duplicate Bug Reports using Natural Language and Execution Information**, Proceedings of the 30th international conference on Software engineering.

Kimball, Ralph (1997), **A Dimensional Modeling Manifesto**, DBMS.

Kimball, Ralph; Becker, Bob (2006), **Design Tip #81 Fact Table Surrogate Key**, Kimball Design Tips.

Passos, Lopes, Emanuel; Goldschmidt, Ronaldo (2005), **Data Mining - um Guia Prático**, 1 ed., São Paulo, Campus.

Elmasri, Ramez; Navathe, B., Shamkant (2005), **Fundamentals of database systems**, 4 ed., Addison Wesley.

Feldman, Ronen; Fresko, Moshe; Kinar, Yakkov ; Lindell, Yehuda; Liphstat , Orly; Rajman , Martin; Schler , Yonatan; Zamir, Oren (1998), **Text Mining at the Term Level**, Proceedings of the Second European Symposium on Principles of Data Mining and Knowledge Discovery.