ChangeMiner: um *plug-in* visual para auxiliar a manutenção de software

Francisco R. Santos[1], Methanias Colaço Júnior[2, 3], Manoel Mendonça [2]

[1] GRUFE – Grupo de Pesquisa no Desenvolvimento de Ferramentas Computacionais Educacionais
Instituto Federal de Sergipe (IFS) – Lagarto – Brasil

[2] LES – Laboratório de Engenharia de Software
Universidade Federal da Bahia (UFBA) – Salvador – Brasil

[3] NUPIC – Núcleo de Pesquisa e Prática em Inteligência Competitiva
Universidade Federal da Sergipe (UFS) – Itabaiana – Brasil

frchico@gmail.com, mjrse@hotmail.com, manoel.g.mendonca@gmail.com

Abstract. *Software repository mining is an important approach to understand and improve software development and maintenance activities. A way to leverage this approach even further is to use IDEs, plugin-in mined results and models to these environments, to support decision making during software engineering activities. This article discusses the construction of an IDE plug-in to predict module change association on the fly, in order to help programmers to reduce omission errors during software maintenance activities.*

Resumo. *A mineração de repositórios de software tem se apresentado como uma importante alternativa para compreensão e melhoria das atividades de desenvolvimento e manutenção de software. Uma forma de aplicar esta abordagem na prática é explorar sua utilização em IDEs. Plug-ins podem ser construídos para acoplar resultados e modelos minerados a estes ambientes, disponibilizando apoio à decisão on-line. Este artigo discute a construção de um plug-in que prediz associações entre modificações de módulos à medida que estes são acessados, com o objetivo de reduzir erros de omissão durante atividades de manutenção de software.*

1. Introdução

A garantia da qualidade de produtos de software reside na execução de tarefas sistemáticas e de auditorias para verificar como o processo está sendo ou deve ser implementado. Estas tarefas devem ser apoiadas por métricas de software, tratadas e exploradas por Basili et al (1994), IEEE (1998) e Fenton et al (1997), enquadrando-se nos padrões e requisitos de projeto com eficiência e eficácia.

Em qualquer processo de garantia de qualidade, a qualidade do produto está diretamente ligada com o processo de desenvolvimento e produção. Na engenharia de software esta relação é mais próxima, motivando a criação de padrões de maturidade de software como o CMM (*Capability Maturity Model*) e o CMMI (*Capability Maturity Model Integration*) [SEI 2002].

Modelos de maturidade sugerem um repositório integrado de métricas, visando coletar dados consistentes de diferentes projetos [Plaza et. Al. 2003]. Devido à disponibilidade de tais repositórios, pesquisas em sistemas de informação vêm agora sendo intensificadas para melhor explorá-los. Entre as principais metas dessas pesquisas, podemos citar a construção de modelos preditivos para estimação de defeitos em produtos de software [Bergel et. Al 2008]. Dados históricos dos projetos de sistemas são utilizados para o reconhecimento de padrões de modificações, evolução, decadências e identificação de erros em módulos de software.

Sistemas gerenciadores de versão (SCV) estão entre os tipos de repositórios que são frequentemente explorados, pelo fato de armazenarem as modificações que são realizadas durante a construção/melhoria de um sistema de informação. Estas modificações evidenciam o acoplamento e as dependências que módulos de software possuem entre si, os quais são geralmente modificados em grupos. A história pregressa de modificações conjuntas de módulos de software permite a criação de modelos de análise de impacto. Nestes modelos, a taxa de associação entre modificações conjuntas pode ser usada para sugerir quais outros módulos de software podem ser afetados, dado que um deles está sendo modificado.

Estudos anteriores a este trabalho fizeram uso este tipo de abordagem em bases de projetos softwares livres e com SCV abertos [Zimmermann e Weissgerber 2004], havendo poucas evidências que atestem a sua validade em outros contextos. Este fato justifica a experimentação da abordagem de descoberta de associações de modificações de módulos de software em novos contextos. Em particular, é desejável que seja explorada a descoberta de associações em bases de SCV de projetos industriais com processos de desenvolvimento mais rígido e controlado.

Este artigo explora a arquitetura do ChangeMiner, cuja finalidade é a extração de dados de SCVs, para inferir um modelo de predição e acoplá-lo a um IDE (MS Visual Studio). Na abordagem utilizada, as estimativas de associações podem ser solicitadas pelo desenvolvedor, em tempo real, durante a execução de uma manutenção de sistema em um IDE comercial.

Para descrição da ferramenta, o restante do artigo é organizado da seguinte forma: a próxima seção discute trabalhos relacionados. Na seção 3, são descritos os princípios usados para prover a infra-estrutura básica da solução. A seção 4 descreve o funcionamento e arquitetura geral da ferramenta. Finalmente, a seção 5 discute aprimoramentos e oportunidades de pesquisas futuras.

2. Trabalhos relacionados

Alguns trabalhos já exploraram a utilização de regras de associação para predição de mudanças de software. Ying et al. (2004) apresenta uma abordagem de mineração de associação em modificações de arquivos, sem oferecer a mineração em tempo real (*on-the-fly*). Zimmermann et al. (2005) desenvolveu uma abordagem para a mineração *on-the-fly* para predizer relações entre módulos ao nível de classes e de métodos, na IDE *Eclipse*. Estes dois trabalhos treinaram seus modelos e testaram suas acuracidades com softwares e repositórios de código aberto. Ambos apresentam as associações através de uma interface textual.

A execução deste projeto segue uma abordagem semelhante a que é proposta por Zimmermann et al na concepção da ferramenta ROSE [Zimmermann e Weissgerber

2004]. A ROSE minera as mudanças de um módulo de software, sugerindo novas alterações a serem feitas nas entidades ou alertando para mudanças que ainda precisam ser feitas.

No trabalho anteriormente mencionado, tanto o SCV quanto a base de dados minerada provieram de sistemas de código aberto. Além disso, não foi utilizado um repositório projetado para a análise de dados históricos [Colaço 2004].

Em detrimento a essa estrutura, nossa solução possui seguintes características: (a) ela utiliza um Data Warehouse (DW) para armazenar os dados do SCV [Colaço et al 2009B]; (b) realização de testes em ambiente industrial com softwares de código fechado e (c) a avaliação do modelo e da ferramenta construída foi feita em um estudo experimental que compara os resultados de precisão e cobertura [Rijsbergen 1979] das regras de associação produzidas.

3. Princípios utilizados

3.1. Extração de dados de repositórios de códigos

Repositórios de versões são amplamente utilizados nas equipes de desenvolvimento de software, pois oferecem uma estrutura de controle da evolução do sistema, além de serem requisitos em modelos de maturidade tais como o CMMI e o MPS-BR [Pronus 2008].

Modelos de maturidade sugerem armazenamento histórico, o qual é suprido em parte pelos SCVs, à medida que os mesmos guardam as alterações feitas nos módulos de um software, além de oferecerem dados tais como programador responsável, data e hora da alteração, número da revisão e comentário de modificação.

Extrair esses dados é um requisito básico para criação de modelos de associação de modificações de módulos de software. Neste trabalho, para extração, foi utilizado o algoritmo proposto por Zimmermann e Weissgerber (2004), em conjunto com um algoritmo de identificação dos sistemas envolvidos na alteração. São considerados associados aqueles módulos modificados em uma mesma transação, sendo definida por uma janela de tempo (neste caso cinco minutos), o nome do autor, o comentário e o caminho da alteração. Com a identificação dos módulos alterados juntos, torna-se possível fazer a busca de associações descrita a seguir.

3.2. Regras de associação

Identificados os módulos que participaram de uma transação, utiliza-se um algoritmo para descoberta de regras de associação do pacote de *Business Intelligence* da Microsoft, o qual disponibiliza um algoritmo pertencente à família APRIORI [Agrawal e Srikant 1994], eficiente para achar relações entre os itens de uma transação.

Como saída, o algoritmo produz regras com a associação entre os módulos de software, informando o nível de suporte e confiança para as mesmas. Neste contexto, o suporte é o número de transações em que o antecedente aparece e a confiança da regra é o percentual de transações do suporte nas quais as entidades antecedente e consequente aparecem. Um exemplo de alerta ao programador pode ser visto na Figura 3, traduzindo-se assim: *quem alterou ngVendasCC.cs, também alterou dbAcaoMkt.cs e fcAcaoMkt.cs, com confianças de 93,85 e 93,39, respectivamente.*

4. ChangeMiner

O *plug-in* que construímos chama-se ChangeMiner. Ele é um *plug-in* gratuito para o Visual Studio, capaz de minerar um *Data Mart* que pode ser alimentado pelos repositórios *Visual Source Safe* (VSS) ou *Subversion* (SVN). No momento da alteração de código, baseado no histórico pregresso de alteração conjunta de módulos, o *plug-in* reporta ao desenvolvedor os próximos pontos prováveis (módulos) de manutenção tanto de maneira textual quanto gráfica, objetivando aumentar a eficiência da manutenção e diminuir a inserção de erros por omissão de código.

4.1. Pré-Requisitos

São pré-requisitos do ChangeMiner: (a) Microsoft SQL Server 2005 com Analysis Services; (b) Microsoft .NET Framework 3.0 ou superior e (c) Uso do SCV VSS 6.0a (ou superior) ou SVN 1.4 (ou superior).

4.2. Arquitetura da ferramenta

A arquitetura proposta para a ferramenta (ver Figura 1) é composta de quatro camadas básicas e integradas definidas como: (a) *Data Source*, para representar os repositórios que fornecerem os dados a serem trabalhos; (b) Ferramentas de ETL, apresentada na seção 4.2.1, realiza a coleta, o tratamento e unificação os dados; (c) Ambiente de DW, anteriormente publicado em [Colaço Jr. et al, 2009A], com o objetivo de agrupar os dados corporativos extraídos pelas diversas ferramentas de ETL e (d) Ambiente de Análise e construção de softwares, seção 4.2.2, para representar as ferramentas que podem fazer uso dos dados agrupados e armazenados no DW, seja através de consultas, de ferramentas OLAP, painéis de gerenciamento (*dashboards*) ou, até mesmo, dentro de um ambiente de desenvolvimento.

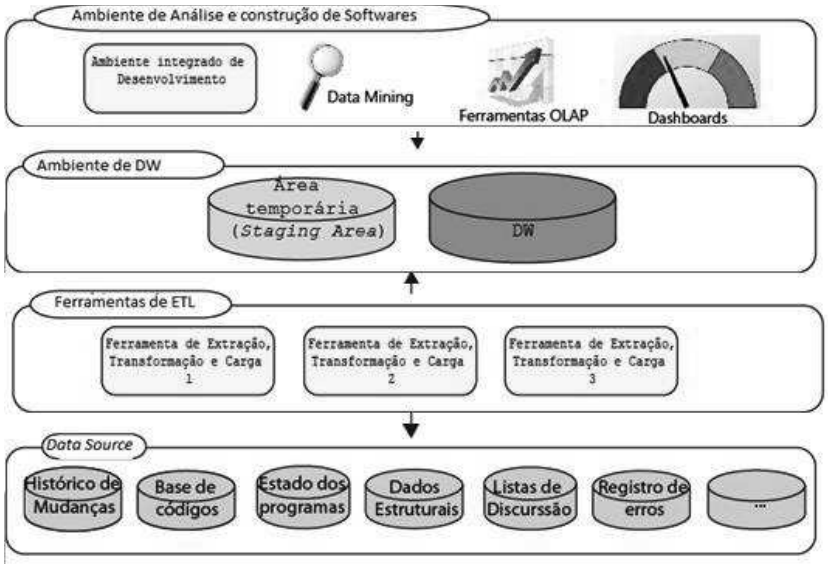

Figura 1. Arquitetura do ChangeMiner

4.2.1.Ferramenta de ETL

A ferramenta possui duas estruturas lógicas para extração e armazenamento dos dados: (1) Uma para camada de extração, transformação e carga dos dados (ETL) e (2) o *Data Mart* que recebe os dados já tratados, oriundos da camada ETL.

Para o modelo de ETL, foram definidas duas tabelas que recebem os dados importados dos SCVs (vide Figura 2). A tabela *VSSUser* armazena as informações dos usuários do repositório e a tabela *VssItem* guarda cada item de uma transação. A carga para o *Data Mart* pode ser realizada de duas maneiras: automática e manual, sendo a automática a configuração padrão, a qual é inicializada após a configuração de um agendamento (*job*) no Sistema Gerenciador de Banco de Dados (SGBD) para invocar um procedimento de carga. Já a manual é realizada para forçar uma carga dos SCVs.

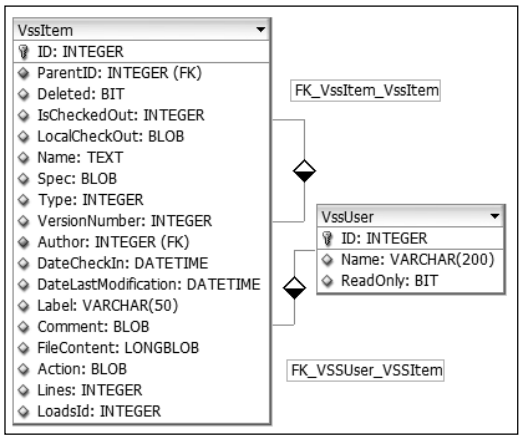

Figura 2. Tabelas básicas do ETL

O *framework* de extração dos dados foi construindo utilizando a linguagem C# e permite a integração com outras bibliotecas e ferramentas de ETL através da implementação da interface *ExtractFromSourceControl.* A interface garante o funcionamento do módulo extrator com diversos SCVs, delegando as tarefas de obtenção dos dados a uma extensão especializada para um repositório SCV específico.

O presente trabalho customizou o módulo extrator para o VSS e SVN através das implementações *ExtractVSS* e *ExtractSVN*. Ou seja, o módulo realiza a conexão com o repositório e inicia uma pesquisa em profundidade, buscando e extraindo os itens dos diretórios que atendem a um intervalo de data/hora previamente definidos (através de parâmetros). Os artefatos que atendem à condição são enviados para as tabelas do ETL.

Ao término do procedimento de importação dos dados dos repositórios, são executados as *store procedures* no banco de dados para seleção do que pode ou não pode ser enviado ao DW. Os procedimentos analisam as transações de modificações, utilizando o algoritmo proposto por Zimmermann e Weissgerber (2004), com uma janela de tempo de cinco minutos, em conjunto com um algoritmo de identificação dos responsáveis pela alteração, sistemas e classes modificadas.

Já a integração com outros tipos de programas, à exemplo de coleta métricas de engenharia de software, é dada através da implementação dos eventos (*delegates* em C#). Com isso, a API do ChangeMiner torna-se um framework para novas ferramentas de Engenharia de Software.

4.2.2.Módulo de Apresentação

Um botão no IDE permite a descoberta, em tempo real (*on the fly),* de associações existentes entre a entidade sendo editada e as outras entidades do sistema (Figura 3).

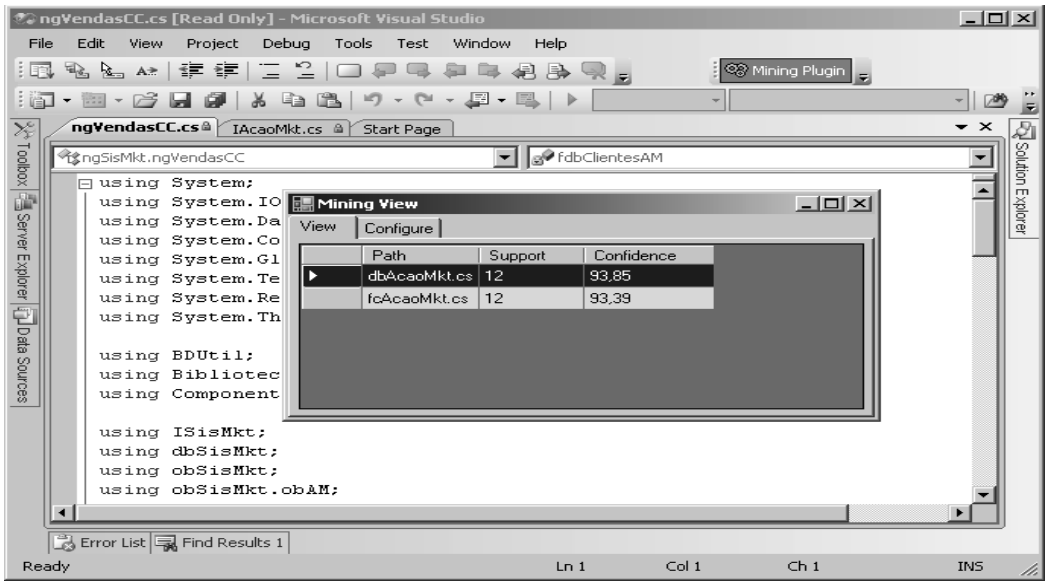

Figura 3. Execução do *Plug-in*

O resultado de consulta ao modelo é exibido de duas formas: grafo e tabular. Na forma tabular (vide Figura 3), são apresentadas ao programador as dez primeiras entidades que alcançam uma confiança de 75% de associação com a entidade sendo alterada, em ordem decrescente de suporte e confiança. Já na forma de grafo, conforme Figura 4, a raiz representa a entidade em alteração, ligada às suas possíveis dependências, respeitando-se a confiança e ordenação citadas anteriormente. Esta funcionalidade ainda está em evolução e na sua versão final será possível navegar na árvore exibindo as associações relacionadas a cada um dos itens.

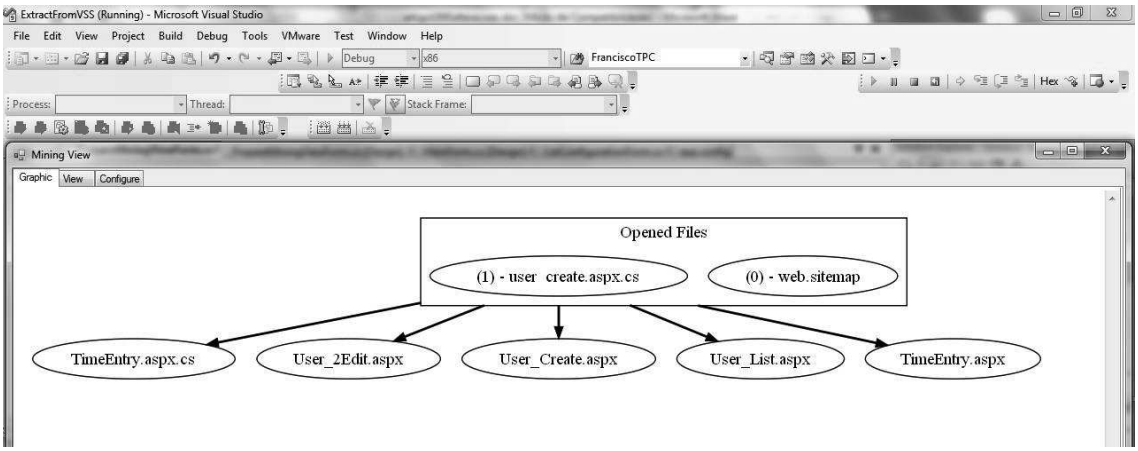

Figura 4. Resultados exibidos usando Grafo

4.3. Resultados Experimentais

Um experimento em ambiente industrial foi realizado para avaliar o plug-in e a acurácia do modelo [Colaço Jr. et al., 2009B]. O parceiro escolhido foi a CIAL, um fabricante e distribuidor de produtos Coca-Cola para os estados de Sergipe e Alagoas, que possui vários sistemas de grande porte desenvolvidos na plataforma Microsoft .NET.

Após oitos anos de uso do VSS pela corporação, foram avaliadas as bases VSS de 18 grandes sistemas, perfazendo 256.804 transações em 4.096 arquivos. Destas

transações, após a limpeza e seleção dos dados, foram identificadas 153.288 transações caracterizadas como *labels* ou *tags*, 69.592 mudanças de diretórios e 33.924 transações realmente válidas para treinamento do modelo de mineração.

Os resultados médios de cobertura e precisão foram de 76% e 58%, respectivamente. Em outras palavras, dada uma mudança, a ferramenta acerta 58% das sugestões das próximas alterações, chegando a alcançar a precisão de mais de 70% em diversos sistemas, conforme pode ser observado na tabela 1. No quesito prevenção de erros, a ferramenta conseguiu um resultado médio de 90% de precisão [Colaço et al 2009B].

Tabela 1. Dados minerados do ambiente industrial (C = Cobertura Média; P = Precisão Média)

Seq	Projeto: Descrição	C	P
1	CadAlmox: Utilitário para migração do sistema de almoxarifado	0.83	0.56
2	Cadastros: Sistema de Cadastros	0.79	0.61
3	Cliente: Gerenciamento dos Clientes	0.87	0.73
4	Contabil: Sistema contábil	0.65	0.53
5	DesenvolProjetos: Ferramentas de desenvolvimento	0.89	0.62
6	Fiscal: Sistema Fiscal	0.59	0.71
7	FrameWork: Framework	0.73	0.61
8	GUF: Sistema de Cosméticos	0.91	0.54
9	Legados: Legados	0.67	0.71

Seq	Projeto: Descrição	C	P
10	Logistica: Sistema de Logística	0.74	0.51
11	SECV: Sistema de Administrativo	0.90	0.48
12	SisAlmox: Sistema de almoxarifado	0.86	0.50
13	SisAP: Sistema de pesquisa	0.57	0.46
14	SisComo: Sistema de comodato	0.68	0.51
15	SisCusteio: Sistema de custos	0.67	0.50
16	SisManutencao: Utilitário para configuração do portal (WEB)	0.89	0.67
17	SisMkt: Sistema de Marketing	0.83	0.68
18	SisPortal: Portal WEB	0.65	0.56

Média Geral:	0.76	0.58

Uma amostra das regras geradas foi validada pelos programadores, os quais se beneficiaram da possibilidade de ver as dependências de forma visual (Figura 4) e, através de suas experiências, confirmaram a acuracidade das principais dependências apresentadas.

5. Conclusão e trabalhos futuros

Este artigo apresentou um *plug-in* que explora a mineração de regras de associação para melhoria do processo de manutenção de software. O desenvolvimento da ferramenta mostrou que é viável manter uma base de dados histórica do processo de manutenção de código, bem como explorar o acoplamento de técnicas de mineração de dados aos ambientes de desenvolvimento de software utilizados na indústria.

Assim, nós acreditamos que esse trabalho pode estimular a indústria a investir no uso de mineração de dados para auxiliar as tarefas de manutenção, uma vez que foi possível mostrar viabilidade técnica para alcançar altos valores de precisão e cobertura em ambientes industriais.

No que toca as dificuldades encontradas durante a execução deste trabalho, destacam-se: (a) a ausência de documentação que descrevesse como eram estruturadas as bases de dados utilizadas pelos repositórios de códigos, principalmente do VSS; e (b)

a descrição das mensagens de erros das APIs elaboradas pelos fabricantes dos produtos, isto é, dos VSS e SVN.

Para concluir, como trabalhos futuros, teremos que: (1) realizar avaliação utilizando níveis mais finos de granularidade e relacionamento entre sistemas; (2) analisar utilidade do plug-in através do uso controlado pelos programadores e do preenchimento de um *survey* pelos mesmos; (3) explorar diversas técnicas de mineração de dados para descoberta de conhecimento no DW construído; (4) integrar com sistemas de análise de códigos bem como sistemas de gerenciamento de erros e (5) realizar a integração com novos repositórios, principalmente os distribuídos como o GIT.

6. Referências

Agrawal, R.; Srikant, R. Fast algorithms for mining association rules. In *Proceedings of the 20th Very Large Data Bases Conference*, p. 487–499, 1994.

Carneiro, G. D.; Magnavita, R.; Spinola, E.; Spinola, F; Mendonça, M. G. An Eclipse-Based Visualization Tool for Software Comprehension, *XXII Brazilian Symposium on Software Engineering,* 2008.

Colaço Jr., M.; Mendonça, M. G.; Rodrigues, F. Data Warehousing in an Industrial Software Development Environment. In: *33rd Annual IEEE/NASA Software Engineering Workshop*, 2009A.

Colaço Jr., M.; Mendonça, M. G.; Rodrigues, F. Mining Software Change History in an Industrial Environment. *XXIII Brazilian Symposium on Software Engineering*, p. 54-61, 2009B.

Alexandre Bergel and Stéphane Ducasse and Jannik Laval and Romain Peirs. Enhanced Dependency Structure Matrix for Moose. In FAMOOSr, 2nd Workshop on FAMIX and Moose in Reengineering, 2008

Gall, H.; Jazayeri, M.; Krajewski, J. CVS release history data for detecting logical couplings. *In Proc. International Workshop on Principles of Software Evolution*, p. 13–23, 2003.

Pronus. O que é Gerência de Configuração. Disponível em: <http://pronus.eng.br/artigos_tutoriais/gerencia_configuracao/gerencia_configuracao.php?pagNum=0 >, 2008.

Rijsbergen, C. J. *Information Retrieval*. 2 ed. Butterworths, London, 1979.

Ying, A. T.; Murphy, G. C.; NG, R.; Chu-Carroll, M. C. Predicting source code changes by mining change history. *IEEE Transactions on software engineering*, v. 30, n. 9, p. 574-586, 2004.

Zimmermann, T.; Weissgerber, P. Preprocessing CVS data for fine-grained analysis. In: *International Workshop on Mining Software Repositories*, 2004.

Zimmermann, T.; Zeller, A.; Weissgerber, P.; Diehl, S. Mining version histories to guide software changes. *IEEE Transactions on Software Engineering*, v. 31, n. 6, p. 429–445, 2005.