Evaldo de Oliveira da Silva – 01/06/2011
Leitura 9. Zimmermann, T., Weisgerber, P., Diehl, S., Zeller, A. Mining Version Histories to Guide Software Changes. International Conference on Software Engineering (ICSE), p.563-572. 2004.

Este artigo propõe a aplicação de mineração de dados utilizando histórico de versões. Com a mineração de dados é possível elaborar questões como: "Desenvolvedores que modificaram estas funcionalidades também modificam...", ou "Dado uma coleção de mudanças existentes, determinadas regras sugerem mais mudanças em outras parte do código". A mineração de dados pode prevenir erros, ou apresentar itens de configuração de um mesmo sistema que também podem ser alterados. O artigo descreve também o protótipo para mineração de dados chamado *Rose*, que conseguiu sugerir 26% a mais de arquivos para serem modificados, e 15% de funções e variáveis que deveriam ser modificadas ou inseridas.

A arquitetura construída para o *Rose* é dividida em dois ambientes, o *Rose Server* que lê os arquivos versionados por meio de grupos de mudanças com base em transações. A mineração das transações é realizada utilizando regras que descrevem implicações entre entidades de software. Ou seja, se algum objeto está modificado, então o método que o utiliza está modificado também.

O artigo apresenta a introdução formal das definições de mudanças, transações, e entidades afetadas, generalizando os conceitos geralmente encontrados em arquivos versionados. A notação de mudança é mapeada como $\int : P \rightarrow P$, e quando aplicada transforma um produto $p \in P$ dentro de um produto modificado, ou seja, $p' = \int(p) \in P$. Neste caso, P é a coleção de todos os produtos, já a coleção de todas as mudanças é representada como $C = P \rightarrow P$. Além disso, as mudanças podem ser compostas usando uma operação de composição o: $C \times C \rightarrow C$. Isto é útil para denotar transações consistindo de múltiplas mudanças em múltiplos locais. Por exemplo, a transação $\Delta_{1,2}$ entre duas versões $p_1, p_2 \in P$, composta por n mudanças individuais, tais como, $\int_1 ..... \int_n$, é representada como $\Delta_{1,2} = \int_1 o \int_2 o .... \int_n$ com $\Delta_{1,2}(p_1) = (\int_1 o \int_2 o .... \int_n)(p_1) = \int_1 (\int_2 (...\int_n(p_1))) = p_2$. O mapeamento de entidades permite a recuperação de todas as entidades afetadas por uma mudança ou transação, por meio da seguinte expressão: $entidades(\Delta) = entidades(\int_1) U ... entidades(\int_n)$.

O servidor da ferramenta *Rose* recupera as mudanças e transações dos arquivos versionados da seguinte forma: i-) inferindo transações; ii-) por meio de ramos e junções (ou *merges*); e iii-) recuperação de entidades. A primeira opção identifica modificações subsequentes de um mesmo desenvolvedor e com o mesmo motivo de mudança. A segunda considera as modificações realizadas em ramos e que são resultantes de junções. A terceira opção realiza a associação das mudanças ocorridas nas entidades a um intervalo de linhas.

Algumas regras estabelecidas para a mineração de dados permitem fazer associações entre as entidades modificadas sugerindo alterações em uma entidade que possui referência ao código alterado na outra entidade. É importante destacar, que as regras nem sempre dizem a verdade, e tem como base a interpretação das transações identificadas após as modificações.

As regras de mineração de dados são aplicadas em tempo de desenvolvimento por meio do *Rose Client*, o qual pode sugerir possíveis mudanças nos códigos. A ferramenta *Rose* criada como protótipo para mineração de dados versionados usa o algoritmo *Appriori* para computar a associação de regras ao ambiente de desenvolvimento.

O artigo apresenta ainda vários exemplos de regras adotadas no desenvolvimento e compilação de códigos desenvolvidos em C e que usam o compilador GCC, e aplicações desenvolvidas em Pynthon e Postgresql. Além disso, algumas medições sobre a eficiência da mineração de dados foi realizada, concluindo que a ferramenta *Rose* pode ser bastante útil na sugestão de mudanças. Porém, os autores destacam que alguns pontos da ferramenta podem ser melhorados, como analisar e conhecer mais amplamente o histórico de modificações a fim de sugerir mudanças de acordo com a realidade de cada aplicação.