UNIVERSIDADE FEDERAL FLUMINENSE
INSTITUTO DE COMPUTAÇÃO
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO

Luiz Laerte Nunes da Silva Junior
Thiago Nazareth de Oliveira

Vertical Code Completion

Niterói
2010

Luiz Laerte Nunes da Silva Junior
Thiago Nazareth de Oliveira

Vertical Code Completion

Monografia apresentada ao Departamento de Ciência da Computação da Universidade Federal Fluminense como parte dos requisitos para obtenção do Grau de Bacharel em Ciência da Computação.

Orientadores: Leonardo Murta
Co-orientador: Alexandre Plastino

Niterói
2010

Luiz Laerte Nunes da Silva Junior
Thiago Nazareth de Oliveira

Vertical Code Completion

Monografia apresentada ao Departamento de Ciência da Computação da Universidade Federal Fluminense como parte dos requisitos para obtenção do Grau de Bacharel em Ciência da Computação.

Aprovado em Junho de 2010

BANCA EXAMINADORA

Prof. Leonardo Murta, D.Sc.
Orientador
UFF

Prof. Alexandre Plastino, D.Sc.
Co-Orientador
UFF

Prof. - , M.Sc.
UFF

Prof. - , D.Sc.
UFF

Niterói
2010

RESUMO

VCC VERTICAL CODE COMPLETION AQUI ENTRA O RESUMO

Palavras Chave:
Engenharia de Software, Code Completion, Mineração de Dados, Mineração de Padrões Sequenciais.

ABSTRACT

VCC VERTICAL CODE COMPLETION AQUI ENTRA O ABSTRACT

Keywords:
Software Engineering, Code Completion, Data Mining, Sequence Mining.

LISTA DE ACRÔNIMOS

VCC: *Vertical Code Completion*

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

CAPÍTULO 1 - INTRODUÇÃO

No domínio do desenvolvimento de aplicações comerciais ou não, alcançar o máximo de produtividade, qualidade e eficiência é um dos objetivos fundamentas da Engenharia de Software [8]. Diversas técnicas e ferramentas são criadas com esse propósito, e muitas delas utilizam de alguma forma conhecimento de produtos de software preexistentes. Em muitos casos, esses produtos possuem uma quantidade de linhas de código da ordem de milhões e muito conhecimento importante pode ser extraído desse montante de dados [7].

Dentre as ferramentas que utilizam essa fonte de informações, as de *code completion* se destacam e são adotadas em praticamente todas as IDEs (Integrated Development Environment) utilizadas atualmente [10]. Isso acontece, pois elas incrementam a produtividade dos programadores, e os encorajam a usar nomes de variáveis mais descritivos tornando o código mais legível — com um simples clique ou uma simples combinação de teclas pode-se reescrever uma variável de nome extenso.

Todavia, essas são ferramentas bastante simplificadas que fazem apenas um casamento do que foi digitado com o que está disponível no ambiente de desenvolvimento. Neste contexto, este trabalho visa a utilização dos repositórios de software de uma forma mais ampla, melhorando ainda mais a produtividade dos desenvolvedores de software.

Essa nova proposta de utilização exige uma grande quantidade de informações em um repositório de código fonte, para que ocorra uma efetiva extração de conhecimento. Entretanto, essa necessidade também se torna um empecilho, pois não é fácil filtrar o que realmente importa. Coletar esses dados manualmente é inviável, pois demandaria muito tempo e esforço, e os projetos em geral estão em contínua evolução. Surge então a necessidade de automatizar a extração de informações úteis desse grande volume de dados. Nesse contexto, técnicas de mineração de dados se apresentam como uma forma eficiente de se extrair padrões que se repetem frequentemente [6].

Dessa forma, o objetivo deste trabalho é criar uma ferramenta que se propõe a ir além do *code completion* tradicional [10], sugerindo sequências semânticas de código fonte, extraídas de um repositório através da aplicação de técnicas de mineração de padrões sequenciais. Essas sugestões podem aumentar a produtividade do desenvolvedor assim como evitar o surgimento

de erros, devido ao questionamento natural que o mesmo irá fazer sempre que uma informação pertinente, que não se relaciona com o que estava para ser desenvolvido, for sugerida no decorrer da atividade.

O restante deste trabalho está organizado da seguinte forma.

O Capítulo 2 apresenta uma introdução à área de mineração de dados, descrevendo e exemplificando as suas tarefas: extração de regras de associação, classificação, clusterização e extração de padrões sequenciais, sendo esta última a técnica usada neste trabalho para extrair sugestões de código. Além disso, apresentamos alguns trabalhos que aplicam mineração de dados na área de Engenharia de Software.

O Capítulo 3 apresenta uma visão geral da abordagem proposta neste trabalho, descrevendo em alto nível os passos tomados para se chegar a solução proposta.

O Capítulo 4 discute os detalhes de implementação da abordagem proposta, descrevendo técnicas, ferramentas e tecnologias utilizadas.

O Capítulo 5 apresenta os resultados experimentais obtidos, descrevendo o planejamento do experimento bem como a avaliação e discussão dos resultados.

Finalmente, o Capítulo 6 apresenta a conclusão deste trabalho, relatando as suas contribuições, limitações e possíveis trabalhos futuros.

CAPÍTULO 2 - REVISÃO DA LITERATURA

Neste capítulo, são abordados alguns conceitos e técnicas de Mineração de Dados, além da aplicação dessas técnicas em problemas de Engenharia de Software, citando trabalhos desenvolvidos nessa área.

2.1 MINERAÇÃO DE DADOS

A evolução computacional das últimas décadas, ocasionada pela evolução tecnológica, permitiu um grande aumento no poder de processamento e na capacidade de armazenamento de dados a baixo custo, inserindo no mercado, novas tecnologias de transmissão e disponibilização de dados [4]. Isso permitiu que empresas e centros de pesquisa acumulassem grandes quantidades de dados históricos a partir dos anos 70 e 80[6].

Porém, essa enorme quantidade de dados não refletia uma grande riqueza de conhecimento, pois não era analisada com ferramentas adequadas, já que um volume tão extenso de informações ultrapassa a habilidade humana de compreensão. Consequentemente, importantes decisões eram frequentemente tomadas somente por intuição, simplesmente pela falta dessas ferramentas que poderiam extrair conhecimentos valiosos dos repositórios de dados [6].

Isso motivou diversos estudos a partir do início dos anos 90, que resultaram no surgimento do campo de pesquisa de Mineração de Dados, área que se refere ao processo de descoberta de novas informações e conhecimento, no formato de regras e padrões, a partir de grandes bases de dados [12]. A partir daí, foram desenvolvidas ferramentas para analisar e descobrir importantes padrões de dados, contribuindo para diversas áreas, tais como [6]: pesquisas médicas, negócios estratégicos, biologia molecular, entre outras.

Dentre as principais tarefas em Mineração de Dados, destacam-se [5]: classificação, extração de regras de associação, clusterização, padrões em séries temporais e extração de padrões seqüências. Em geral, essas tarefas podem ser classificadas em duas categorias: mineração preditiva e mineração descritiva [6].

Na mineração preditiva, deseja-se prever o valor desconhecido de um determinado atributo, a partir da análise histórica dos dados armazenados na base. Nessa categoria se enquadram as

tarefas de classificação e padrões em séries temporais. Na mineração descritiva, padrões e regras descrevem características importantes dos dados com os quais se está trabalhando. Mineração de regras de associação, clusterização e mineração de padrões sequenciais fazem parte dessa categoria.

A seguir serão descritas as principais tarefas de mineração de dados.

2.1.1 CLASSIFICAÇÃO

A tarefa de classificação tem por objetivo identificar, entre um conjunto pré-definido de classes, aquela à qual pertence um elemento a partir de seus atributos. Para inferir a qual classe esse elemento pertence, é necessária uma base de treinamento.

Um sistema de um banco, que tem por objetivo inferir a classe à qual o cliente pertence, indicando se o mesmo será ou não um bom pagador, com base nos dados de clientes antigos e nas características do elemento que está sendo classificado, é um exemplo de utilização da tarefa de classificação.

2.1.2 REGRAS DE ASSOCIAÇÃO

Uma regra de associação representa um padrão de relacionamento entre itens de dados de um domínio de aplicação, que ocorre com uma determinada frequência. Essas regras são extraídas a partir de uma base de dados organizada em transações, que são formadas por um conjunto de itens desse domínio. Um exemplo genérico de regra de associação que poderia ser extraído de uma base de dados de dados qualquer de vendas é: “clientes que compram o produto A geralmente compram o produto B”.

2.1.3 CLUSTERIZAÇÃO

A tarefa de clusterização é usada para agrupar elementos de uma base de dados através de seus atributos ou características, de forma que elementos similares fiquem no mesmo cluster e elementos não similares entre si fiquem em clusters distintos.

Essa técnica é muito utilizada em sistemas de grandes operadoras de cartão de crédito, separando os clientes em grupos de forma que aqueles que apresentam o mesmo comportamento de consumo fiquem no mesmo grupo. A separação desses clientes por grupo pode ser usada para fazer algum tipo de marketing apropriado ao grupo ou na detecção de fraudes, no caso de um cliente apresentar um comportamento diferente do esperado para o seu perfil.

2.1.4 PADRÕES EM SÉRIES TEMPORAIS

Uma série temporal é uma seqüência de valores mensurados em intervalos iguais de tempo [6]. O principal objetivo da análise de padrões em séries temporais é realizar previsões futuras baseando-se no histórico dos dados.

Cotação diária do dólar, faturamento anual de uma empresa e evolução do índice da bolsa de valores são exemplos de séries temporais.

2.1.5 PADRÕES SEQUENCIAIS

Nesta seção, detalharemos com maior profundidade a extração de padrões sequenciais, visto que essa será aplicada em nosso trabalho.

Existem muitas aplicações envolvendo dados sequenciais, e a ordem com que esses dados aparecem é muito importante para análise e entendimento de alguns padrões. Exemplos típicos incluem sequências de compras de um cliente, sequências biológicas e sequências de eventos na ciência e na engenharia. Padrões sequenciais representam sequências de eventos ordenados, que aparecem com significativa frequência em uma base de dados. Um exemplo de padrão seqüencial é: "clientes que compram uma câmera digital Canon comumente compram uma impressora HP colorida dentro de um mês".

Sequências são listas ordenadas de eventos. Uma sequência **s** é representada por $< e_1 e_2 e_3 ... e_n >$, onde e_j, $1 \leq j \leq$n, é dito um evento ou elemento da sequência **s** e e_1 ocorre antes de e_2, que ocorre antes de e_3 e assim sucessivamente. Por sua vez, um evento ou elemento da sequência é representado por **e** = $(i_1 i_2 i_3 ... i_m)$, onde i_k, $1 \leq k \leq$n, é um item do domínio da aplicação. O tamanho da sequência é determinado pelo seu número de itens.

Podemos dizer que um evento em uma base de dados de uma loja de vendas é uma compra feita por um cliente, e os itens do domínio da aplicação são os produtos que pertencem à essa compra.

Além disso, outras definições são importantes. Uma sequência pode ser parte de outra sequência maior. Nesse caso, a sequência $\alpha = < a_1 a_2 ... a_n >$ é chamada de subsequência de outra sequência $\beta = < b_1 b_2 ... b_m >$, e β é uma supersequência de α, denotado como $\alpha \subseteq \beta$, se existirem inteiros $1 \leq j_1 < j_2 < ... < j_n \leq$ m tais que $a_1 \subseteq b_{j1}$, $a_2 \subseteq b_{j2}, ..., a_1 \subseteq b_{jn}$. Por exemplo, se $\alpha = < (ab), (d) >$ e $\beta = < (abc), (de) >$, onde a, b, c, d e e são itens, então α é uma subsequência de β e β é uma supersequência de α.

A métrica de avaliação dos padrões sequenciais é o suporte. Um banco de dados de

sequência, S, é composto de um conjunto de tuplas, $< SID, s >$, onde SID é o identificador da sequência e s é a sequência. O suporte de uma sequência α num banco de dados de sequência, S, é o número de tuplas no banco de dados que contem α. Esse suporte pode ser absoluto, representado apenas por um número inteiro, ou relativo, informando o percentual de vezes que a sequência ocorreu. Dessa forma, uma sequência de eventos é dita frequente se a quantidade de vezes que essa sequência ocorrer for superior ao suporte mínimo informado pelo usuário, formando assim um padrão sequencial.

2.2 MINERAÇÃO DE DADOS NA ENGENHARIA DE SOFTWARE

Tarefas de mineração de dados têm sido muito utilizadas em engenharia de software, principalmente quando aplicadas em repositórios de dados para se obter informações sobre a evolução de software ao longo do tempo, com o objetivo de aumentar a qualidade do processo de desenvolvimento de software [1, 2, 11, 13, 14].

Em [11], utilizou-se um algoritmo classificador, a partir do aprendizado em uma base de treinamento, para extrair relações que indicam que arquivos de código fonte, em um sistema legado, são relevantes uns aos outros no contexto da manutenção de software. Tais relações poderem revelar interconexões complexas entre os arquivos de código fonte do sistema, podendo sua vez, ser úteis na compreensão deles. Assim, o algoritmo classificador, após receber dois arquivos de código fonte, retorna verdadeiro ou falso, indicando se são relevantes entre si.

Em [13], aplica-se mineração de dados para se encontrar relações de dependências entre arquivos do código fonte, até mesmo dependências difíceis de se determinar, tais como aquelas existentes entre códigos fonte de linguagens diferentes. Em especial, é utilizada a técnica de extração de regras de associação para determinar padrões de mudanças entre arquivos do código fonte para ajudar os desenvolvedores em tarefas de modificação.

Em [2], utilizam-se técnicas de mineração de dados num repositório UML versionado para se extrair regras de associação que possam identificar elementos do modelo UML que foram modificados juntos no passado e que provavelmente precisarão ser modificados juntos no futuro.

Em [14], aplicam-se técnicas de mineração de dados em repositórios versionados de código fonte para se extrair regras de associação do tipo: “programadores que alteraram a função A também alteraram as funções B e C”. Tais regras são extraídas com o objetivo de guiar desenvolvedores de software na alteração do código fonte.

Em [1], um algoritmo de clusterização foi utilizado num sistema de controle de versões

para identificar classes semanticamente relacionadas. A partir do gráfico gerado pelo algoritmo, pôde-se analisar que mudanças em classes de certo cluster eram frequentemente envolvidas com mudanças em classes de outro cluster.

CAPÍTULO 3 - VERTICAL CODE COMPLETION

Nesse capítulo será detalhada a abordagem proposta nessa dissertação. O mesmo foi dividido em duas seções, que representam duas fases distintas no processo de uso do plugin VCC, construído como resultado desse trabalho.

A primeira fase é a de preparação e mineração dos dados, onde são extraídos todos os padrões que serão sugeridos ao programador que estiver utilizando a ferramenta. Nesta fase o código fonte é analisado e organizado, para permitir que a mineração de dados desse código seja efetuada e em seguida, seus resultados sejam armazenados em uma estrutura adequada.

Será visto na subseção 3.1.1 esse processo de análise de código fonte e em seguida a estratégia de mineração de dados será descrita na seção 3.1.2. Finalizando essa primeira seção, a estrutura de armazenamento dos padrões obtidos será detalhada na seção 3.1.3.

A segunda fase desse projeto está detalhada na seção 3.2. Nessa etapa, o código que está sendo produzido em tempo real pelo usuário do VCC será analisado com o intuito de encontrar padrões frequentes que foram obtidos na primeira fase. Em seguida, esses padrões são classificados através de métricas e sugeridos para o usuário.

3.1 OBTENÇÃO DE PADRÕES FREQUENTES DE CODIFICAÇÃO DE SOFTWARE.

3.1.1 ANÁLISE DO CÓDIGO FONTE

Para construir um software que possa, através de uma fonte de conhecimento pré-existente, sugerir padrões frequentes de código fonte, se faz necessário a existência de uma base de dados armazenada de forma coesa e estruturada para execução de consultas. Nesse trabalho isso não é uma realidade inicial, visto que o código de uma aplicação é armazenado em formato texto, sem obedecer a padrões rígidos de estruturação.

Felizmente, cada linguagem de programação obedece a um conjunto de regras de formatação, que são necessárias para a compilação adequada em linguagem de máquina do código produzido.

Desta forma, embora não seja possível fornecer diretamente arquivos de código como entrada para um software padrão de mineração sequencial de dados, os padrões da linguagem de programação podem ser utilizados para se extrair as informações pertinentes do código fonte. Essas informações devem então ser organizadas de uma maneira que obedeça aos padrões de entrada do programa que executará a mineração de padrões sequenciais.

Como será visto no capítulo 4, existem ferramentas que podem auxiliar na interpretação do código fonte, evitando que um parser precise ser construído para cada linguagem.

3.1.2 MINERAÇÃO DO CÓDIGO FONTE

Existem diversos algoritmos para realizar mineração de padrões sequenciais, entretanto, há poucas divergências entre a organização da entrada e da saída desses algoritmos. Como visto na seção 2.2.1.5, sequências de eventos relacionados são utilizadas como entrada e então os padrões sequenciais são extraídos. Todavia, em cada domínio de aplicação da mineração de padrões sequenciais, eventos, sequências e os itens que compõem cada evento possuem significados distintos [6].

No projeto VCC, o objetivo é encontrar padrões sequenciais na codificação de métodos, blocos ou procedimentos criados pelo usuário. Como estas nomenclaturas dependem da linguagem de programação em que o projeto está sendo aplicado, nesta dissertação será considerado a sugestão de padrões na construção de métodos.

Sendo assim, cada declaração de método representa o início de uma sequência de eventos, que pode conter algum padrão sequencial frequente no seu interior. Essa sequência de eventos se encontra no corpo do método, e no escopo desse trabalho, ao contrário de algumas convenções frequentemente utilizadas [6], cada evento é atômico, ou seja, não pode ser dividido em diferentes itens.

Esses eventos atômicos são as chamadas a outros métodos do projeto de software que está sendo construído. Dessa forma, os padrões sequenciais minerados são listas de chamadas de método, que obedecem a uma determinada sequência e se repetem frequentemente em diferentes corpos de métodos.

Após definir essas convenções, a mineração de padrões sequenciais pode então ser realizada, mas para que os resultados desse processo sejam proveitosos, os dois conceitos que são apresentados a seguir são essenciais.

3.1.2.1 SUPORTE E CONFIANÇA

O suporte e a confiança dos padrões frequentes que são gerados, são elementos chaves para a mineração sequencial utilizada nesse projeto.

O suporte já foi definido anteriormente no capítulo de revisão da literatura, e sabe-se que é a quantidade de vezes que um determinado padrão se repete na base de dados. Portanto, é possível definir um suporte mínimo para a obtenção desses padrões, filtrando os padrões mais frequentes.

A confiança por outro lado não é um conceito utilizado na mineração sequencial de dados, mas é muito importante nas regras de associação e representa uma nova métrica de avaliação que traz uma maior riqueza para a apresentação de regras mineradas.

Para definir a confiança em padrões sequenciais, primeiramente será definida a confiança em regras de associação. Consequentemente, é necessário que o suporte de uma regra de associação seja definido.

O suporte de um item A, representa a porcentagem de transações da base de dados que contém esse conjunto, e pode ser nomeado como Sup(A). Esse item pode compor um conjunto de itens (A,B) por exemplo, e formar uma regra $(A \rightarrow B)$. O suporte dessa regra será igual à porcentagem de transações que possui A e B [5].

Já a confiança de uma regra de associação A $\rightarrow$ B, representa a porcentagens de transações que contém A e B, dentre todas as transações que contém A, ou seja, Conf$(A \rightarrow B)$ = Sup$(A \cup B)$ / Sup(A) [5].

É possível então definir a confiança em mineração de padrões sequenciais, enxergando-a como uma derivação da existente em regras de associação. Dado um padrão sequencial X, formado por $\{A,B,C,D\}$, pode-se chamá-lo de uma supersequência de Y, formado por $\{A,B\}$, que é uma subsequência de X.

A confiança de um padrão sequencial será então, a porcentagem de transações que possui a sequência X, dentre todas as transações que possuem a subsequência Y.

Definindo a sequencia que contém todos os elementos como **SuperSeq** e a subsequência de **SubSeq**, a confiança de **SuperSeq** é calculada como:

Confiança$_{\text{SuperSeq}}$ = Suporte$_{\text{SubSeq}}$ / Suporte$_{\text{SuperSeq}}$

Esse conceito pode ser exemplificado da seguinte maneira:

Dado que a sequência X possui suporte de 28%, e a sequência Y um suporte de 35%, a

confiança de X em função de Y é de 80%.

Com isso, a seguinte afirmação pode ser empregada pelo VCC:

Usuários que chamam os métodos A e B em sequência, também chamam, com 80% de confiança, os métodos C e D.

3.1.2.2 GERAÇÃO DE ÁRVORE DE CHAMADAS

Para que os padrões obtidos na fase de mineração sequencial possam ser utilizados na sugestão de código fonte, uma estrutura adequada deve ser empregada para o armazenamento dos mesmos.

Nesse trabalho, é utilizada uma árvore de profundidade e largura variável para armazenar os padrões frequentes obtidos. É importante ressaltar que a estrutura utilizada na criação da árvore permite que a busca por uma sequência de código tenha complexidade assintótica[9] O(n). Isso acontece porque todos os elementos presentes na árvore em níveis mais profundos, também estão representados no segundo nível.

Apesar de parecer um desperdício proposital de espaço de armazenamento para obter um melhor desempenho na busca por elementos da árvore, a presença de todos os elementos frequentes no segundo nível da árvore é na verdade um comportamento inerente à mineração de padrões sequenciais.

Isso acontece devido ao princípio que diz que se uma sequência é frequente, ou seja, possui suporte superior ao suporte mínimo, todas as suas subsequências também serão frequentes. Entretanto, o suporte e a confiança das subsequências não são obrigatoriamente iguais aos das sequências que as contém, portanto esses padrões devem ser armazenados independentemente.

Após definir a estrutura de armazenamento como uma árvore, é importante que seja decidido o que cada nó irá armazenar. Considerando cada nó como o fim de um padrão sequencial frequente, nos mesmos será armazenado o suporte desse padrão. Entretanto, a confiança de um padrão não pode ser vista como um único valor.

A confiança de um padrão sequencial, depende da subsequência que está sendo considerada. Desta forma, o tamanho do padrão sequencial minerado determinará quantos valores de confiança o mesmo terá. Dada uma sequência de tamanho igual a três, X = $< C, D, B >$, as seguintes confianças são definidas:

- Confiança de X em relação a uma sequência vazia. O valor desta confiança é o mesmo do suporte do padrão sequencial;

- Confiança de X em relação à sequência $< C >$. O valor desta confiança será o suporte de X, dividido pelo suporte de $< C >$;
- Confiança de X em relação à sequência $< C, D >$. O valor desta confiança será o suporte de X, dividido pelo suporte de $< C, D >$;

Desta forma, uma gama de sugestões pode ser fornecida ao usuário utilizador do VCC. Dado que o método C foi codificado, pode-se sugerir o método D, com suporte s_1 e confiança c_1, e também a sequência $< D, B >$, com suporte s_2 e confiança c_2.

A figura abaixo ilustra uma possível árvore de padrões sequenciais minerados. Nela podem-se observar as convenções adotadas nesse trabalho, que foram citadas acima.

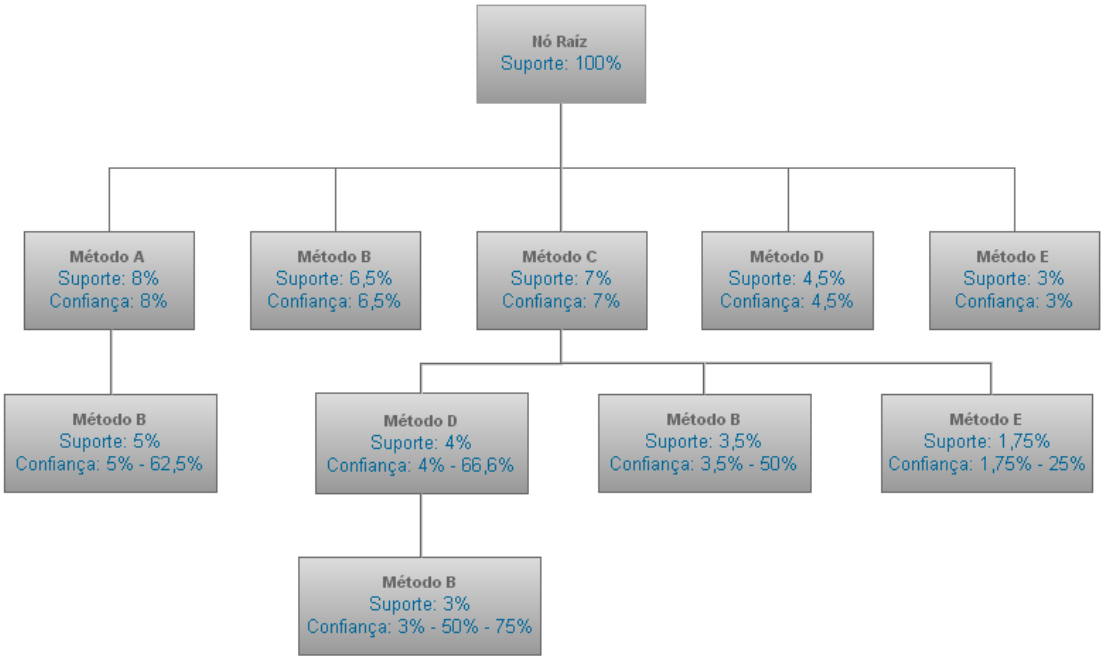

Figura 1: Exemplo de árvore de padrões frequentes

Na figura 1 pode-se ver como o suporte e a confiança dos padrões são armazenados. Observando a sequência frequente $< C, D, B >$ por exemplo, as confianças armazenadas no nó que representa o método B, 3%, 50% e 75%, são respectivamente:

- A confiança do padrão sequencial $< C, D, B >$ com relação a toda base de dados;
- A confiança do padrão sequencial $< C, D, B >$ com relação à sequência $< C >$;

- E a confiança do padrão sequencial $< C, D, B >$ com relação à sequência $< C, D >$.

É importante ressaltar que embora na figura representativa, o valor das confianças dos padrões sequenciais completos - no padrão $< A, B >$, o valor 5% por exemplo - esteja armazenado, essa prática tem caráter apenas ilustrativo. Na implementação da árvore isso não é necessário, visto que essas confianças sempre serão iguais ao suporte do padrão.

3.2 SUGESTÃO DE PADRÕES FREQUENTES DE CÓDIGO FONTE

Para que um padrão sequencial possa ser sugerido, é necessário que uma entrada seja disponibilizada pelo usuário. Nesse momento, diversas estratégias podem ser tomadas para decidir como será feita a pesquisa do que está sendo programado.

Enquanto o projeto VCC estava sendo desenvolvido, algumas dessas estratégias foram testadas com o intuito de realizar poucas consultas, otimizando o tempo de resposta do programa. Utilizar apenas as últimas linhas que foram programadas, ou apenas combinações de linhas contíguas mostrou-se infrutífero, pois muitos padrões interessantes passaram despercebidos por estarem dispostos de diversas maneiras no corpo do método.

Em um método com dez linhas por exemplo, um padrão sequencial frequente pode ser detectado a partir de chamadas que estão localizadas imediatamente uma após a outra, como em chamadas que se encontram uma no início e outra no fim do que já foi programado. Um exemplo interessante de padrões sequenciais que não se localizam contiguamente, são as aberturas e fechamentos de conexões com bancos de dados. Ao abrir uma conexão, espera-se que algum procedimento seja realizado na base de dados, antes que a mesma seja fechada.

Com isso, não é possível prever se o padrão sequencial deve ser sugerido de acordo com o que foi programado nas primeiras, ou nas últimas linhas de código do método em que está sendo realizada a consulta. Portanto, a estratégia adotada nesse trabalho é a de combinar todas as chamadas de métodos disponíveis para realizar a consulta à arvore de padrões sequenciais, mas isso proporciona uma nova dificuldade com relação ao tamanho máximo de cada combinação.

Chamando a profundidade máxima da árvore de padrões sequenciais de **n**. Em um cenário ideal, todas as combinações possíveis, com tamanho variando de 1 até n - 1, deveriam ser utilizadas para consultar os de padrões sequenciais frequentes. Entretanto, não se pode prever qual será a profundidade máxima da árvore de padrões. Mesmo sabendo que as boas práticas de programação recomendam a filosofia de dividir para conquistar [3], métodos em um projeto de software podem se tornar grandes demais. Dessa forma, o tempo de resposta para a consulta

de todas essas combinações pode se tornar inaceitável.

Com isso, para que a consulta a todas as combinações de chamadas de métodos seja realizada em tempo hábil, é necessário que o tamanho máximo dessas combinações seja limitado. No projeto VCC, é possível que o tamanho máximo das combinações seja configurado, permitindo que um valor que atenda às características do projeto em que o mesmo está sendo utilizado seja alcançado. Entretanto, esse valor pode ser alto, gerando uma enorme quantidade de combinações, e fazendo com que o tempo de resposta das consultas não seja satisfatório.

Com o intuito de minimizar esse problema, a partir da análise das combinações geradas, uma estratégia de poda foi desenvolvida, evitando que todas essas combinações sejam consultadas. Essa estratégia parte do princípio de mineração de padrões sequenciais que garante que se uma sequência não é frequente, então todas as supersequências dessa sequência também não são frequentes. No projeto VCC, a poda das sequências a serem consultadas na árvore de padrões acontece após se consultar uma sequência que não é frequente. Todas as outras sequências de método que são supersequências desta são então descartadas.

Finalmente, após todas as combinações de chamadas de métodos geradas terem sido consultadas, os padrões sequenciais obtidos podem ser classificados de acordo com seus valores de suporte e confiança e então devem ser sugeridos para o usuário do VCC. Este usuário pode então analisar as sugestões e escolher a que se adéqua melhor ao que está sendo desenvolvido.

CAPÍTULO 4 - IMPLEMENTAÇÃO

Capítulo 4

CAPÍTULO 5 - RESULTADOS EXPERIMENTAIS

Capítulo 5

CAPÍTULO 6 - CONCLUSÕES

Capítulo 6

REFERÊNCIAS BIBLIOGRÁFICAS

[1] T. Ball, J. Kim, A.A. Porter e et al. If your version control system could talk. *Workshop on Process Modelling and Empirical Studies of Software Engineering, Boston, MA*, 1997.

[2] C. Dantas, L. Murta e C. Werner. Mining change traces from versioned uml repositories. *XXI Simpósio Brasileiro de Engenharia de Software (SBES 2007)*, pp. 236 – 252, October de 2007.

[3] H. M. Deitel. *Java, How To Program*. Prentice Hall, 4 edição, 2002.

[4] M. A. Domingues. Generalização de regras de associação. Dissertação de Mestrado, USP - São Carlos, 2004.

[5] E.C. Gonçalves. Regras de associação e suas medidas de interesse objetivas e subjetivas. *INFOCOMP Journal Computer Science*, pp. 27–36, 1999.

[6] J. Han e M. Kamber. *Data Mining: Concepts and Techniques (2nd edition)*. Morgan Kaufmann, 2006.

[7] Reid Holmes. Using structural context to recommend source code examples. Dissertação de Mestrado, The University of British Columbia, 2004.

[8] Walcelio Melo, Caroline B. S. Holanda e Clarissa Angélica de A. de Souza. Proreuso: um repositório de componentes para web dirigido por um processo de reuso. *XV Simpósio Brasileiro de Engenharia de Software*, pp. 208–223, 2001.

[9] Ian Parberry e William Gasarch. *Problems on Algorithms*. Prentice Hall, 2002.

[10] Romain Robbes e Michele Lanza. How program history can improve code completion. *23rd IEEE/ACM International Conference on Automated Software Engineering*, pp. 317–326, 2008.

[11] J.S. Shirabad, T. Lethbridge e S. Matwin. Supporting software maintenance by mining software update records. *International Conference on Software Maintenance (ICSM)*, pp. 22–31, November de 2001.

[12] R. Srikant e R. Agrawal. Mining sequential patterns: Generalizations and performance improvements. Em *5th International Conference on Extending Database Technology - EDBT*, 1996.

[13] A.T.T. Ying, G.C. Murphy, R. Ng e et al. Predicting source code changes by mining change history. *IEEE Transactions on Software Engineering (TSE)*, 30(9):574–586, 2004.

[14] T Zimmermann, P. Weisgerber, S. Diehl e et al. Mining version histories to guide software changes. *International Conference on Software Engineering (ICSE)*, pp. 563–572, May de 2004.

Apêndice I

Formulário de Consentimento

Estudo

Este estudo visa avaliar o quanto as sugestões de códigos, informadas através do plugin Vertical Code Completion (VCC), são consistentes e fazem sentido no desenvolvimento, tanto para um desenvolvedor experiente quanto para um desenvolvedor novo na equipe.

Idade

Eu declaro ter mais de 18 anos de idade e concordar em participar de um estudo conduzido por Thiago Nazareth de Oliveira e Luiz Laerte Nunes da Silva Junior na Universidade Federal Fluminense.

Procedimento

Este estudo acontecerá em uma única sessão, que incluirá a análise de algumas sugestões de código, retiradas do sistema IdUFF através do plugin VCC. Eu entendo que, uma vez o experimento tenha terminado, os trabalhos que desenvolvi serão estudados visando entender a eficiência dos procedimentos e as técnicas propostas.

Confidencialidade

Toda informação coletada neste estudo é confidencial, e meu nome não será divulgado. Da mesma forma, me comprometo a não comunicar os meus resultados enquanto não terminar o estudo, bem como manter sigilo das técnicas e documentos apresentados e que fazem parte do experimento.

Benefícios e liberdade de desistência

Eu entendo que os benefícios que receberei deste estudo são limitados ao aprendizado do material que é distribuído e apresentado. Eu entendo que sou livre para realizar perguntas a qualquer momento ou solicitar que qualquer informação relacionada à minha pessoa não seja incluída no estudo. Eu entendo que participo de livre e espontânea vontade com o único intuito de contribuir para o avanço e desenvolvimento de técnicas e processos para a Engenharia de Software.

Pesquisadores responsáveis

Thiago Nazareth de Oliveira
Instituto de Computação - Universidade Federal Fluminense (UFF)

Luiz Laerte Nunes da Silva Junior
Instituto de Computação - Universidade Federal Fluminense (UFF)

Professores responsáveis

Prof. Leonardo Paulino Gresta Murta
Instituto de Computação - Universidade Federal Fluminense (UFF)

Prof. Alexandre Plastino
Instituto de Computação - Universidade Federal Fluminense (UFF)

Nome (em letra de forma): _____

Assinatura: _____ **Data:** _____

Apêndice II

Questionário de Caracterização

Este formulário contém algumas perguntas sobre sua experiência acadêmica e profissional.

1) Formação Acadêmica

( ) Doutorado
( ) Doutorando
( ) Mestrado
( ) Mestrando
( ) Graduação
( ) Graduando

Ano de ingresso: _____ Ano de conclusão (ou previsão de conclusão): _____

2) Formação Geral

2.1) Qual é sua experiência com desenvolvimento de software? (marque aqueles itens que melhor se aplicam)

( ) Nunca desenvolvi software.
( ) Já li material sobre desenvolvimento de software.
( ) Já participei de um curso sobre desenvolvimento de software.
( ) Tenho desenvolvido software para uso próprio.
( ) Tenho desenvolvido software como parte de uma equipe, relacionado a um curso.
( ) Tenho desenvolvido software como parte de uma equipe, na indústria.

2.2) Por favor, explique sua resposta. Inclua o número de semestres ou número de anos de experiência relevante em desenvolvimento de software. (E.g. “Eu trabalhei por 2 anos como programador de software na indústria”)

2.3) Qual é sua experiência com desenvolvimento de software em equipes? Qual a maior equipe de que você participou?

_____

2.4) Por favor, indique o grau de sua experiência nesta seção seguindo a escala de 5 pontos abaixo:

0 = nenhum
1 = estudei em aula ou em livro
2 = pratiquei em projetos em sala de aula
3 = usei em projetos pessoais
4 = usei em projetos na indústria

2.4.1) Linguagem JAVA
Respota: _____

2.4.2) Hibernate
Resposta: _____

2.4.3) Spring
Resposta: _____

2.4.4) Mockito
Respota: _____

3) Experiência no desenvolvimento do sistema objeto de estudo - IdUFF

3.1) Há quanto tempo você está na equipe de desenvolvimento do sistema IDUFF?
Resposta: _____

3.2) Qual o seu cargo na equipe?

Resposta: _____