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 [9]. 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 [8].

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 [15]. 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 [7].

Dessa forma, o objetivo deste trabalho é criar uma ferramenta que se propõe a ir além do *code completion* tradicional [15], 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[7].

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 [7].

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 [17]. A partir daí, foram desenvolvidas ferramentas para analisar e descobrir importantes padrões de dados, contribuindo para diversas áreas, tais como [7]: pesquisas médicas, negócios estratégicos, biologia molecular, entre outras.

Dentre as principais tarefas em Mineração de Dados, destacam-se [6]: 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 [7].

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 [7]. 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, 16, 18, 19].

Em [16], 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 [18], 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 [19], 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 monografia. 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 são armazenados em uma estrutura adequada.

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.

A Figura 1 ilustra todo o processo de uso do plugin VCC.

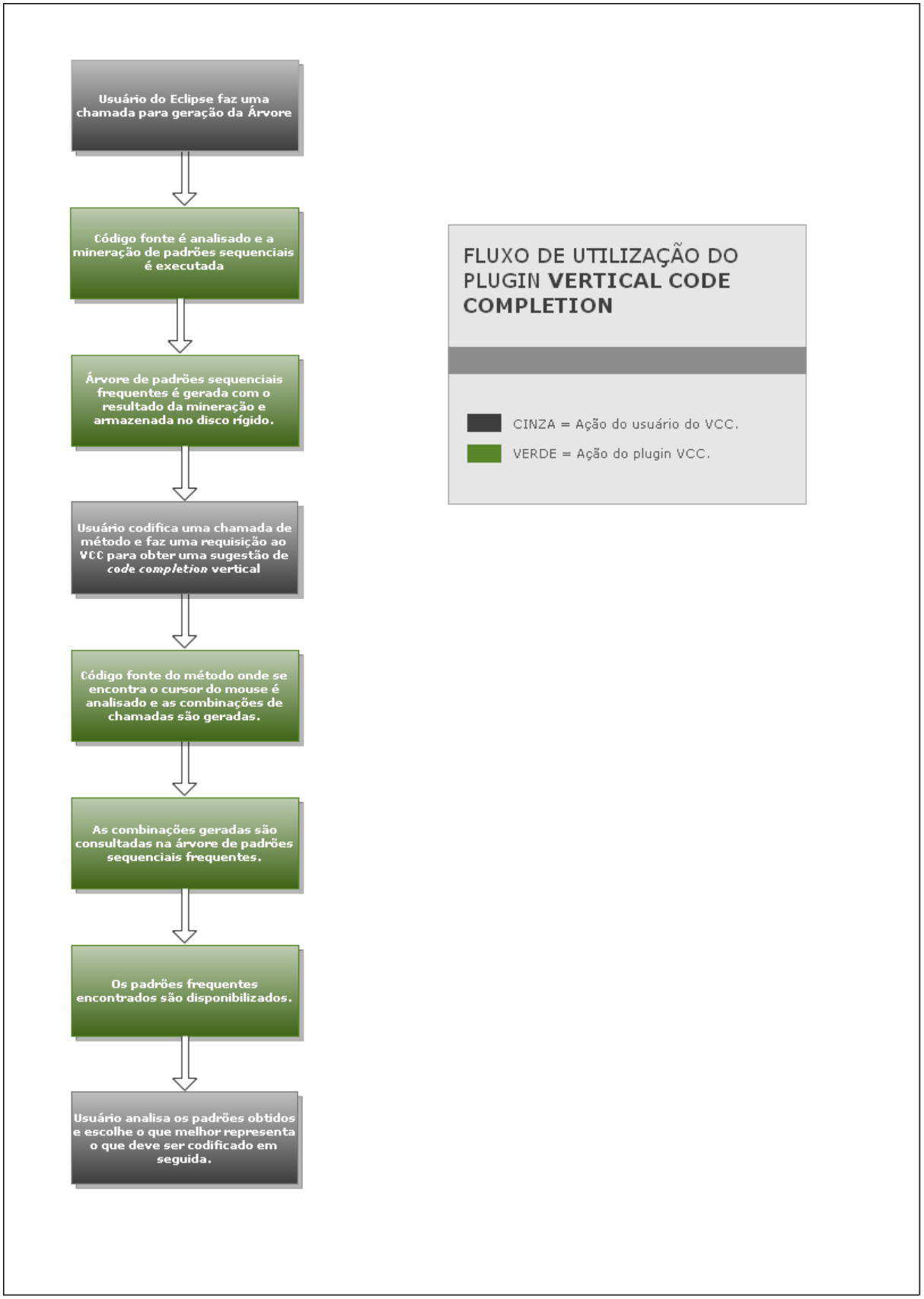

Figura 1: Fluxo de uso do VCC.

3.1 OBTENÇÃO DE PADRÕES FREQUENTES DE CODIFICAÇÃO DE SOFTWARE.

Nesta seção, o processo de análise de código fonte é apresentado na subseção 3.1.1. Em seguida a estratégia de mineração de dados é descrita na subseção 3.1.2. Por último, a estrutura de armazenamento dos padrões obtidos é detalhada na subseção 3.1.3.

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.

Como visto na seção 2.2.1.5, sequências de eventos relacionados são utilizadas como entrada para as implementações de algoritmos de mineração de padrões sequenciais. Todavia, em cada domínio de aplicação, eventos, sequências e os itens que compõem cada evento possuem significados distintos [7].

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.

Na Figura 2, é possível visualizar uma codificação de método genérica e sua respectiva transação gerada.

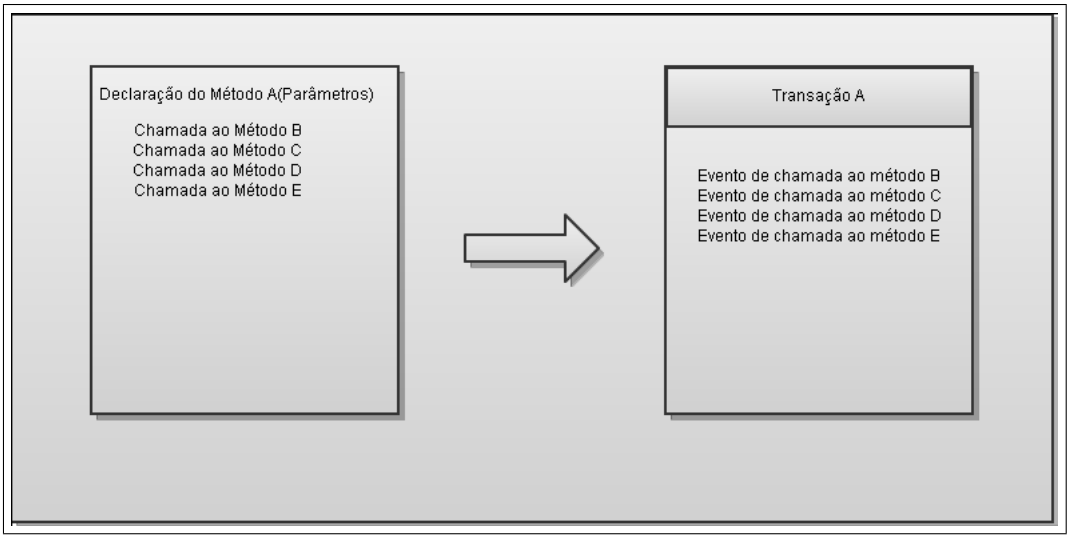

Figura 2: Exemplo de codificação de método e a transação gerada.

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 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.

3.1.2 MINERAÇÃO DO CÓDIGO FONTE

Existem diversos algoritmos para realizar mineração de padrões sequenciais, cada um com suas particularidades de entrada e saída. Organizando os dados de uma maneira que atenda a essas particularidades, 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 chave para a mineração sequencial utilizada nesse projeto.

O suporte já foi definido anteriormente no capítulo 2, 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, é muito importante nas regras de associação e representa uma métrica de avaliação que traz uma maior riqueza para a apresentação de regras mineradas. Entretanto, não é um conceito utilizado na mineração de padrões sequenciais, sendo introduzido por nós nessa área da mineração de dados.

Para definir a confiança em padrões sequenciais, primeiramente será apresentada a definição de confiança em regras de associação. Consequentemente, é necessário que também seja apresentada a definição do suporte de uma regra de associação.

O suporte de um item A, representa a porcentagem de transações da base de dados que contém esse item, 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 [6].

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) [6].

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 supersequência X, dentre todas as transações que possuem a subsequência Y.

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

$$\textbf{Confiança}_{\textbf{SuperSeq/SubSeq}} = \textbf{Suporte}_{\textbf{SubSeq}} \,/\, \textbf{Suporte}_{\textbf{SuperSeq}}$$

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.

3.1.3 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[14] $O(n)$, sendo n o tamanho da sequência pesquisada. 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. A Figura 3 ilustra a estrutura de uma árvore de padrões frequentes, conforme citado.

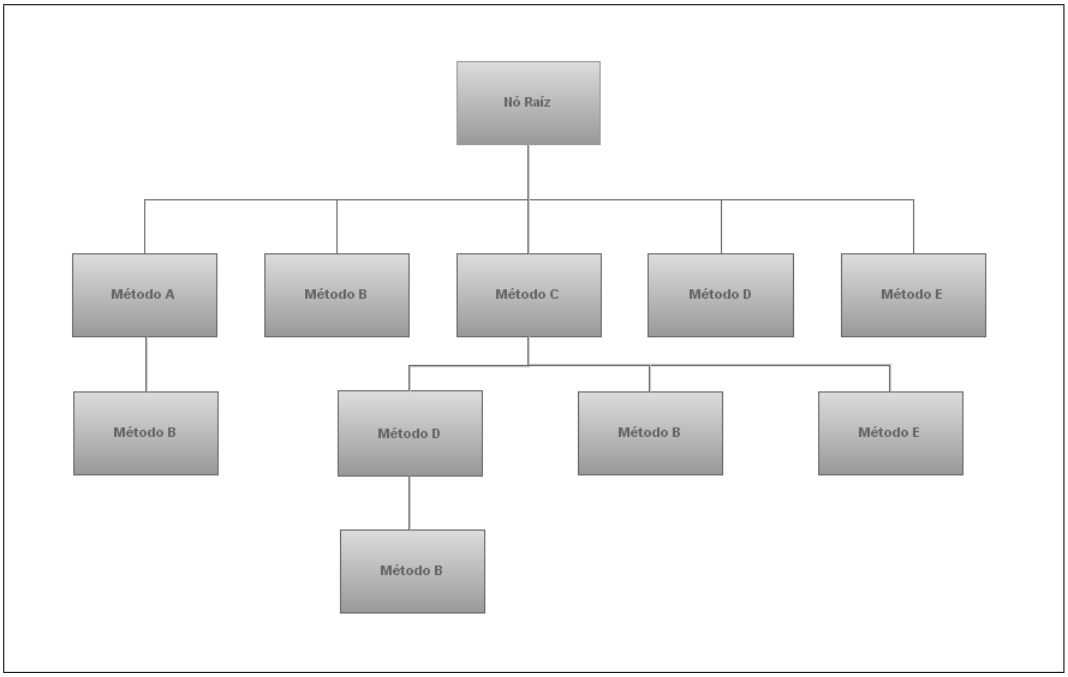

Figura 3: Exemplo de árvore de padrões frequentes.

Após definir a estrutura de armazenamento como uma árvore, é importante que seja deci-

dido 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 uma chamada ao método C foi codificada, 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 4 apresenta a árvore de padrões sequenciais frequentes e a forma com que os suportes e confianças são armazenados.

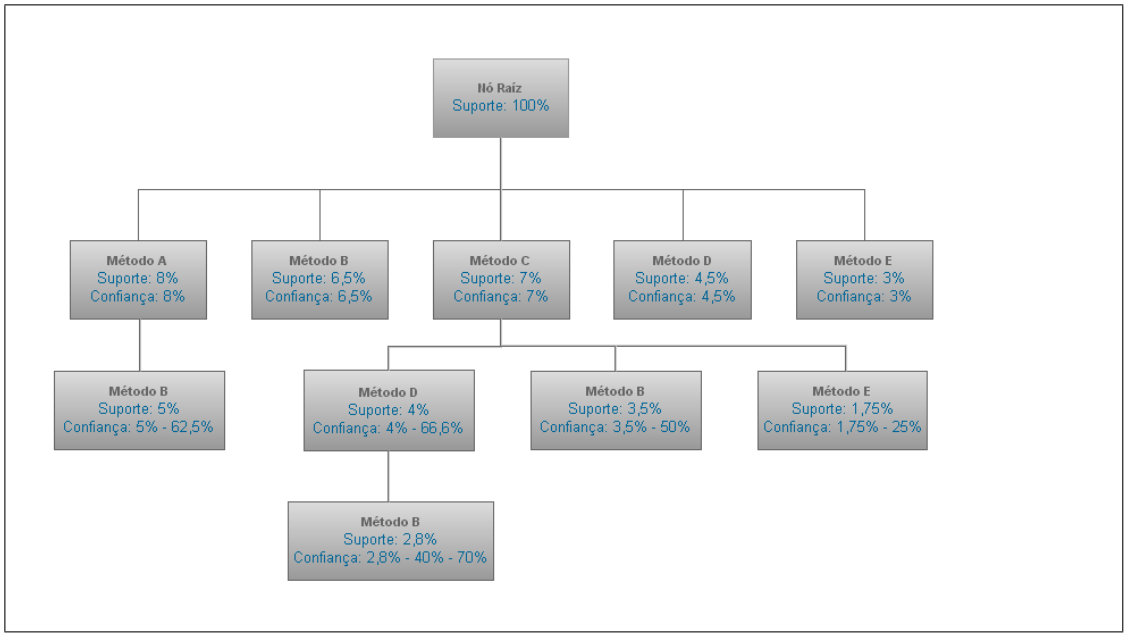

Figura 4: Exemplo de árvore de padrões frequentes com suportes e confianças.

Na Figura 4 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.

Utilizando como exemplo a árvore da Figura 1, pode-se supor que o usuário do VCC codifique uma chamada ao método A. Em seguida, após efetuar uma requisição ao VCC, a chamada ao método B seria sugerida com suporte de 5% e confiança de 62,5%. Já se uma chamada ao método C fosse codificada, as sugestões seriam:

- Chamada ao método D com suporte de 4% e confiança de 66,6%;
- Chamadas ao métodos D e B com suporte de 2,8% e confiança de 40%;
- Chamada ao método B com suporte de 3,5% e confiança de 50%;
- Chamada ao método E com suporte de 1,755% e confiança de 25%.

CAPÍTULO 4 - O *PLUGIN* VCC

Neste capítulo, os detalhes das tecnologias utilizadas na implementação do *plugin* VCC são abordados, com foco nos pontos detalhados no capítulo 3.

Para desenvolver uma ferramenta que auxilie o desenvolvedor de *software* no momento da codificação, primeiramente é necessário decidir em que ambiente a mesma será utilizada. Atualmente é difícil se pensar em desenvolver um sistema sem o auxílio de uma IDE, que além de ajudar os programadores, em geral é um ambiente ideal para se acoplar uma nova ferramenta. Por esse motivo, e ainda visando alcançar o maior número de usuários, decidiu-se utilizar o VCC acoplado em uma IDE.

Entretanto, para decidir em que IDE o VCC será acoplado, é preciso definir qual será a linguagem de programação em que o mesmo será utilizado. Neste trabalho a linguagem escolhida foi Java, mas utilizando os conceitos detalhados no capítulo 3 é possível construir uma ferramenta semelhante a esta. Com isso, utilizando como argumentos favoráveis a facilidade de acoplar novas ferramentas no formato de *plugins* e também a enorme abrangência na comunidade Java [10], a IDE Eclipse foi escolhida para receber o *plugin* VCC.

Uma das grandes vantagens de implementar este trabalho no formato de um *plugin* para o Eclipse é que a própria IDE já fornece uma interface de desenvolvimento de *plugins*. É possível então, utilizar diversas funcionalidades que a ferramenta disponibiliza, como por exemplo o ASTParser.

Essa ferramenta constrói uma AST do código fonte em que o *plugin* está sendo executado. Embora a AST seja uma representação estritamente sintática da estrutura do código fonte de uma aplicação, essa representação funciona exatamente como o volume de dados necessário para efetuar a mineração de padrões sequenciais frequentes. Na construção do VCC, o trabalho de análise do código fonte, que foi destacado no capítulo 3, se reduziu a acessar essa AST, sem que tenha sido preciso construir um parser textual para a linguagem Java.

4.1 A ANÁLISE DO CÓDIGO FONTE ATRAVÉS DA ASTPARSER

Para extrair os padrões sequenciais frequentes da aplicação em que o VCC está sendo utilizado, é preciso obter todas as chamadas de métodos utilizadas nos corpos dos métodos deste projeto. Para isso, se devem seguir as convenções de acesso à AST, que serão listadas a seguir.

Primeiramente, é necessário definir que no topo da hierarquia da AST está a classe ASTNode. Todas as construções Java são representadas por esta classe. Obviamente, através do mecanismo de herança, essas entidades Java são especializadas, permitindo que cada uma possua características próprias.

Sendo assim, para analisar o código fonte, os ASTNodes devem ser acessados, extraindo as chamadas de métodos e salvando-as para serem mineradas pelo algoritmo de mineração de padrões sequenciais. Para obter essas chamadas, uma hierarquia deve ser seguida para acessar todas as classes do projeto em questão. Essa hierarquia é intuitiva, pois obedece a estrutura adotada pela linguagem Java e pela IDE Eclipse.

A Figura 5 exibe a estrutura de um *Workspace* Java. Um *Workspace*, como a tradução já diria, é um espaço de trabalho em que os projetos do usuário ficam armazenados.

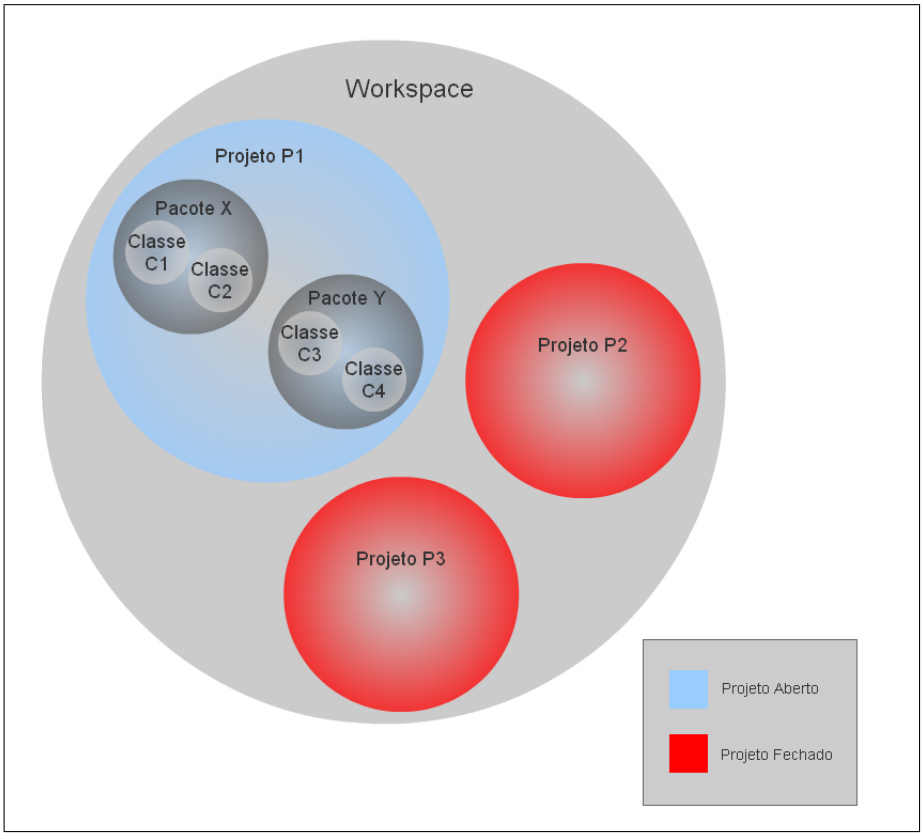

Figura 5: Estrutura de um Workspace Java.

O acesso ao Projeto começa então pelo *Workspace* em que o mesmo está hospedado. O código do *plugin* VCC acessa todos os Pacotes da aplicação. O Pacote é uma forma de organizar as Classes Java do software que está sendo construído. Com isso, o próximo passo é acessar todas as Classes que estão em cada Pacote e só então todos os Métodos que estão dentro das Classes.

Com isso, é possível obter todas as chamadas de Métodos que se encontram dentro do corpo de um Método, e armazená-las em formato de eventos para a execução da Mineração de Padrões Sequenciais. Por último, é importante citar que todos os acessos implementados pela ASTParser são realizados através do *Design Pattern Visitor*[13].

4.2 MINERAÇÃO DE PADRÕES SEQUENCIAIS

Existem diversas implementações de código aberto de algoritmos de mineração de padrões sequenciais, disponíveis na web. Por esse motivo, foi decidido que não seria necessário o desenvolvimento de um programa que realize a mineração desses padrões. Após a análise de algumas dessas implementações, o PLWAP [5] foi escolhido como o algoritmo que seria utilizado para a extração dos padrões.

Entretanto, embora o desempenho do PLWAP fosse satisfatório, o mesmo apenas informava quais padrões sequenciais eram frequentes, sem disponibilizar o suporte de cada padrão. Uma modificação no algoritmo foi necessária para obter esses valores e em seguida calcular as confianças de cada padrão sequencial frequente.

4.3 ÁRVORE DE PADRÕES FREQUENTES

Conforme visto no capítulo 3, todos os padrões sequenciais são armazenados em uma árvore, que é consultada quando um padrão sequencial está sendo buscado. Na implementação do VCC, está árvore é representada por um conjunto de nós, que são objetos da classe **MethodCallNode**. Essa classe possui além das confianças do padrão e da assinatura completa do método, uma referência para o nó pai e um **HashMap**, que contém todos os filhos desse nó. Um **HashMap** é uma classe Java, que através de uma tabela *hash*, organiza um mapa de dados [11].

Para exemplificar essa estrutura, supondo um padrão sequencial X = $< A, B, C >$, e outro padrão Y = $< A, B, D >$, o nó **B** será representado na árvore como um objeto do tipo **MethodCallNode**, que contém um **HashMap** com as referências para os nós **C** e **D**, além de uma

referência para o nó **A**.

Por fim, como a fase de construção da árvore de padrões sequenciais é independente da fase de consulta a esses padrões, utilizando a interface Serializable[12], o objeto que representa o nó raiz dessa árvore é armazenado na memória secundária do sistema em que o plugin está sendo executado, para ser acessado posteriormente na fase de consultas.

4.4 SUGESTÃO DE PADRÕES FREQUENTES

Diversas estratégias foram discutidas, com o intuito de decidir como as chamadas de método que já foram codificadas, seriam enviadas para serem consultadas na árvore de padrões frequentes. Duas opções principais foram analisadas e são expostas a seguir.

A primeira dessas opções é analisar todas as chamadas de métodos que estão disponíveis no corpo do método que está sendo codificado. A seleção deste método poderia ser feita através da digitação do nome do pacote, da classe e do próprio método. Embora essa opção representasse uma facilidade na implementação, prejudicava muito a usabilidade da ferramenta, além de não permitir uma melhor delimitação da área de pesquisa. Um usuário poderia estar realizando uma alteração na metade do corpo do método, por exemplo, e independentemente disso, todas as chamadas, do início ao fim do mesmo seriam utilizadas na pesquisa por padrões de código fonte.

A segunda opção seria utilizar a posição do cursor do mouse do usuário do VCC, assim como funciona no *code completion* tradicional. Quando o usuário posiciona o mouse e faz uma chamada ao VCC, todas as chamadas de método que se localizam acima do cursor, até o início do corpo do método, são combinadas e utilizadas para a consulta na arvore de padrões frequentes. Pode ser visto então, que essa alternativa, além de delimitar melhor a área de consulta, também incrementa a usabilidade do *plugin*, fazendo com que tenha sido a escolhida para ser implementada.

4.5 GERAÇÃO E PODA DAS COMBINAÇÕES DE CHAMADAS DE MÉTODOS

Como foi citado no Capítulo 3, a geração de todas as combinações de chamadas de método, com o intuito de consultar a árvore de padrões frequentes, é a maneira mais eficaz de garantir que esses padrões não estão deixando de ser sugeridos ao usuário.

Para que essas combinações possam ser geradas, primeiramente é preciso ler todas as chamadas de métodos que se encontram antes do cursor. Para facilitar essa tarefa, mais uma

vez a **ASTParser** foi utilizada, entretanto, não foi preciso iterar em todos os pacotes, classes e métodos, pois a posição do cursor já informa em qual método devem ser lidas todas as chamadas. Após a leitura das chamadas de métodos, cada combinação é armazenada em um objeto da classe **ComparableList**. Essa classe pertence ao projeto VCC, é uma subclasse de **ArrayList** e implementa a interface **Comparable**. Esta interface exige que um método **compareTo** seja criado, para que dois objetos possam ser comparados. O critério de comparação utilizado para essa classe foi o tamanho da lista. Com isso, é possível armazenar todas as combinações em um outro **ArrayList** e ordená-lo pelo tamanho das combinações.

Essa ordenação é muito importante, pois conforme já foi citado anteriormente, gerar todas as combinações de chamadas de métodos pode produzir uma enorme quantidade de dados para serem consultados, fazendo com que seja necessária a criação de uma estratégia de poda das combinações. Essa estratégia depende da ordenação dos dados, pois permite que após consultar uma determinada combinação de chamadas de métodos que não é frequente, o **ArrayList** que contém todas as outras combinações geradas é percorrido, apenas a partir do índice dessa combinação. Todas as combinações que são supersequências da sequência que não é frequente são então removidas, evitando que sejam feitas consultas inúteis.

Após todas as consultas terem sido feitas, os padrões sequenciais obtidos são então sugeridos ao usuário da aplicação em uma janela, permitindo que o mesmo avalie de acordo com o suporte e a confiança da regra, qual melhor se aplica na sua codificação.

CAPÍTULO 5 - RESULTADOS EXPERIMENTAIS

Nesse capítulo serão discutidos os resultados experimentais obtidos a partir da utilização do *plugin* VCC sobre o sistema objeto de estudo, o IdUFF - Sistema de Gestão Acadêmica da Universidade Federal Fluminense. Na seção 5.1 será apresentado o sistema utilizado no experimento, na seção 5.2 será apresentado o planejamento do experimento, seguido pela seção 5.3, que aborda a aplicação do experimento. Finalmente, nas seções 5.4 e 5.5 serão apresentados, respectivamente, os resultados obtidos e as ameaças a validade desse experimento.

5.1 O SISTEMA OBJETO DE ESTUDO - IDUFF

O sistema IdUFF é o sistema de gestão acadêmica utilizado na Universidade Federal Fluminense. Ele foi escolhido para ser utilizado no experimento pela facilidade de contato com os desenvolvedores, que são ou foram alunos de graduação e pós-graduação da UFF, e por ser um sistema de médio porte, que vem sendo desenvolvido há 3 anos e apresenta uma grande quantidade de código fonte a ser minerada. O mesmo foi desenvolvido sobre a plataforma Java, a mesma linguagem que o *plugin* VCC foi desenvolvido para ser utilizado. Possuí aproximadamente 40 mil usuários, entre alunos, professores e funcionários da universidade. O repositório de código fonte do IdUFF possui 4920 commits e o sistema IdUFF tem aproximandamente 850 revisões. O sistema possui 779 classes JAVA, desconsiderando classes de teste, arquivos de interface HTML e arquivos de configuração.

5.2 PLANEJAMENTO DO EXPERIMENTO

Com o objetivo de avaliar o quanto as sugestões de código do *plugin* VCC seriam úteis para um desenvolvedor que o estivesse utilizando, duas abordagens de experimento foram discutidas, ambas utilizando como base a experiência de desenvolvimento do programador no sistema IdUFF.

Na primeira abordagem, o próprio desenvolvedor utilizaria o *plugin* VCC, instalado na sua IDE, ou seja, no seu ambiente de desenvolvimento do dia-a-dia de trabalho e, a partir disso, poderia avaliar se as sugestões do *plugin* seriam úteis. Porém, esta abordagem não se mostrou

interessante para ser aplicada nesse projeto, pois além de demandar muito tempo, causaria muitos transtornos, pois o ambiente de desenvolvimento do sistema IdUFF utiliza uma IDE diferente da qual o *plugin* VCC foi acoplado. Outro ponto negativo dessa abordagem é que a usabilidade do *plugin* não é o foco desse experimento, e sim os resultados apresentados pelo mesmo.

Na segunda abordagem, uma análise em ambiente controlado poderia ser aplicada, fornecendo aos desenvolvedores um questionário de avaliação contendo alguns padrões de código pré-selecionados, obtidos a partir da utilização do *plugin* VCC sobre o repositório de código fonte do sistema objeto de estudo. Assim, o mesmo poderia classificar, a partir de sua experiência no desenvolvimento do sistema, se as sugestões fariam sentido ou não, se lhe fossem apresentadas no ambiente de trabalho.

5.3 APLICAÇÃO DO EXPERIMENTO

A segunda abordagem foi escolhida, justamente por permitir um maior controle sobre a avaliação feita pelos desenvolvedores voluntários, garantindo-se que os mesmos padrões fossem avaliados por todos. Assim, poderíamos avaliar se os resultados eram interessantes para os diversos perfis de desenvolvedores.

Para participar do experimento, os voluntários deveriam pertencer obrigatoriamente à equipe de desenvolvimento do sistema IdUFF. Sete desenvolvedores foram voluntários na participação desse experimento. Não houve nenhum tipo de compensação para os participantes.

Inicialmente, o voluntário foi informado do estudo através do Formulário de Consentimento (Apêndice I). Caso concordasse em participar, ele preenchia o Questionário de Caracterização (Apêndice II), que avalia o nível de conhecimento e experiência do voluntário em diferentes temas relacionados ao estudo. Essas informações foram de grande utilidade, ajudando na interpretação dos resultados obtidos por cada um dos participantes.

A tabela 6 apresenta um resumo da caracterização desses participantes. Os níveis de experiência variam de 0 a 4, onde 0 representa o nível mais baixo e 4 representa o nível mais alto de experiência.

Na situação proposta aos participantes, um cenário fictício foi sugerido, onde os mesmos estariam programando uma linha de código, por exemplo, chamando pelo método A, e o *plugin* VCC iria exibir um padrão, sugerindo que desenvolvedores que chamam o método A também chamam o método B.

Critério				P1	P2	P3	P4	P5	P6
1			Formação acadêmica	Mestrando	Graduado	Graduando	Graduado	Mestrando	Graduado
2	2.1		Experiência em desenvolvimento	Indústria	Indústria	Indústria	Indústria	Indústria	Indústria
	2.2		Tempo de experiência (anos)	4	4	1	8	3	2
	2.3		Tamanho máximo da equipe (pessoas)	11	11	10	35	30	20
	2.4	2.4.1	Experiência em Java (0-4)	4	4	4	4	4	4
		2.4.2	Experiência em Hibernate (0-4)	4	4	4	4	4	4
		2.4.3	Experiência em Spring (0-4)	4	4	4	4	4	4
		2.4.4	Experiência em Mockito (0-4)	4	0	3	4	0	1
3	3.1		Tempo de experiência no sistema objeto de estudo - IdUFF	3	2	1	4	3	2
	3.2		Cargo na equipe (idUFF)	Gerente	Gerente	Desenvolvedor	Diretor	Gerente	DBA

Figura 6: Resumo do Questionário de Caracterização dos participantes.

Assim, 10 padrões de código foram retirados do total de 424 padrões minerados através da utilização do *plugin* VCC, para serem analisados pelos desenvolvedores. Esses padrões foram obtidos utilizando-se um suporte mínimo de 0,3%, que foi escolhido após testes com diversos valores, por apresentar um número razoável de sequências mineradas e ter um tempo de processamento relativamente rápido. O critério de escolha dos 10 padrões adotados foi uma combinação entre os valores de suporte e confiança dos mesmos: suporte alto e confiança alta, suporte alto e confiança baixa, suporte baixo e confiança alta, suporte baixo e confiança baixa.

Os voluntários então preenchiam o Questionário de Avaliação (Apêndice III), por meio do qual expressavam sua opinião com relação a cada sugestão, podendo optar pelas seguintes respostas: não sei, discordo totalmente, discordo parcialmente, concordo parcialmente e concordo plenamente. As respostas foram dispostas exatamente nessa ordem de aceitação da sugestão.

5.4 RESULTADOS OBTIDOS

Como visto na seção 5.3, 7 voluntários participaram do experimento, todos eles com experiência em desenvolvimento de software. Os resultados obtidos do questionário de avaliação foram, em geral, positivos com relação à utilidade das sugestões de código informadas. Grande parte dos voluntários se mostrou interessado no *plugin*, e gostaria de vê-lo sendo utilizado em seu ambiente de trabalho, se disponibilizando para ajudar a implementar as alterações necessárias para que o mesmo seja utilizado na IDE *NetBeans*.

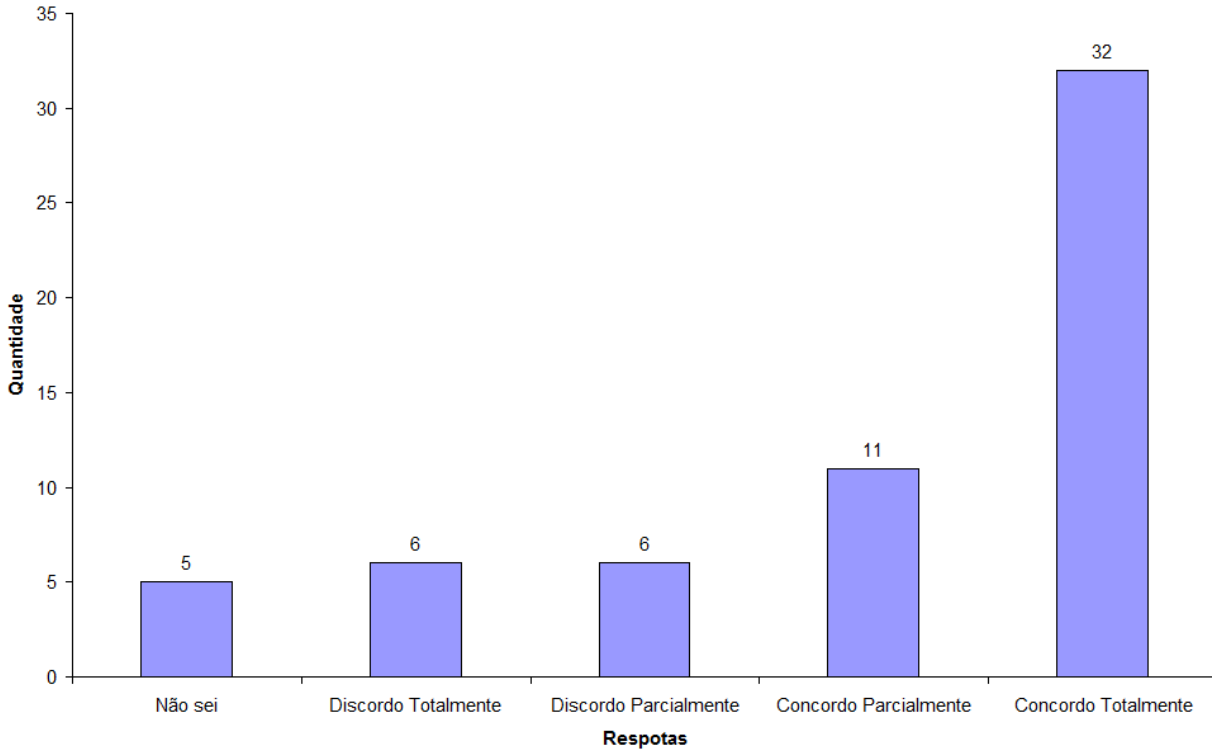

Figura 7: Total de respostas dos participantes para cada item.

5.4.1 ANÁLISE QUANTITATIVA

Das 60 respostas obtidas a partir do preenchimento do questionário de avaliação pelos voluntários, 43 foram positivas, sendo 11 marcadas como *concordo parcialmente* e 32 marcadas como *concordo totalmente*, indicando assim um índice de 71,66% de aceitação das sugestões de código fonte. Foram 12 respostas negativas, sendo 6 marcadas como *discordo totalmente* e 6 marcadas como *discordo parcialmente*, indicando um índice de 20% de reprovação das sugestões apresentadas. Do total das 60 respostas, 5 respostas foram marcadas como *não sei*, sendo 8,33% desse total.

A figura 7 apresenta o total de respostas para cada item que poderia ser escolhido.

5.4.2 ANÁLISE QUALITATIVA

As questões 6 e 8 foram as que mais receberam respostas negativas, contabilizando 8 do total de 12 respostas nesse contexto. Esse resultado já era esperado já que essas questões foram

escolhidas por apresentarem confiança baixa. Os voluntários que marcaram a questão 6 negativamente, comentaram que a sugestão apresentada não fazia sentido por não ser muito usada, e sugeriram outros padrões de chamada de método.

A questão 10 recebeu a resposta *discordo totalmente* por um dos voluntários e o comentário do mesmo foi pertinente à sugestão apresentada: "Se usa o método *BaseMB.info(String)* para mensagens que indiquem sucesso, não faz sentido exibir uma mensagem proveniente de uma situação de erro. Na minha opinião, essa regra apresentada é aplicável aos métodos *BaseMB.error(String)* e *BaseMB.warn(String)*". Esse comentário reflete uma das limitações desse projeto: não levar em conta blocos de repetição, condição e tratamento de erros, como no caso da estrutura *try catch*. Geralmente o método *BaseMB.info(String)* é usado dentro de uma estrutura *try*, e é executado se houver sucesso na operação, enquanto o método *java.lang.Throwable.getMessage()* é utilizado dentro de uma estrutura *catch*, indicando um erro sendo lançado na aplicação. Sendo assim, se a estrutura *try catch* fosse levada em consideração ao se buscar os padrões sequências, essa sugestão não seria minerada, já que o método *java.lang.Throwable.getMessage()* não se enquadraria como uma sequência para o método *BaseMB.info(String)*.

A questão 1 foi a que mais recebeu respostas *não sei*, contabilizando 3 de um total de 5. Isso pode ser justificado pelo questionário de caracterização, onde os voluntários que responderam essa questão dessa maneira são os que indicaram ter pouca ou nenhuma experiência com o *framework* de testes *Mockito* já que a sugestão apresentada está relacionada a esse *framework*. As outras respostas obtidas para essa questão foram bem satisfatórias: 3 respostas *concordo plenamente*. Essa questão apresentava um suporte baixo, porém uma confiança alta, com valor de 100%. Os desenvolvedores que comentaram essa questão disseram que a sugestão apresentada se dá por conta da estrutura do *framework Mockito* e seria bastante útil sua utilização.

As questões que foram respondidas como *concordo totalmente* e *concordo parcialmente*, em geral não receberam comentários.

5.5 AMEAÇAS A VALIDADE

A seleção dos participantes foi feita por meio da solicitação de voluntários dentro da equipe de desenvolvimento do sistema IdUFF, a qual um dos pesquisadores faz parte. Como consequência, os resultados podem ter sido influenciados pela relação de amizade entre os voluntários e o pesquisador. O pequeno número de participantes e o tempo curto para aplicação do experimento foi outro fator importante, pois é possível que os resultados sejam influenciados pelo tamanho e pelas características específicas do grupo.

A quantidade de sugestões avaliadas pelos voluntários era pequena e simplificada, exibindo somente uma sugestão onde poderiam ser sugeridas várias outras. Isso foi necessário devido a limitação de tempo para o preenchimento do questionário por meio dos voluntários e também para análise dos resultados.

CAPÍTULO 6 - CONCLUSÕES

Já ficou evidenciado que o desenvolvimento de *software*, sem ferramentas de apoio a essa atividade, se tornou praticamente impensável nos últimos anos [9]. Essas ferramentas, na maioria das vezes, incrementam a produtividade da construção de sistemas de informação, e novas implementações e técnicas continuam surgindo. Nesse trabalho foi apresentado um aperfeiçoamento da técnica de *code completion*, o Vertical Code Completion, uma ferramenta que utiliza a Mineração de Dados para extrair padrões sequenciais frequentes, e sugere de linhas de código de fonte de acordo com o que já foi programado.

Nossa análise de resultados, mostrou que essa técnica pode trazer resultados positivos no desenvolvimento de aplicações de nível comercial. Entretanto, a falta de um ambiente que se mostrasse propício a aplicação do questionário de sugestões da ferramenta, como o IdUFF, porém com um número maior de desenvolvedores, desfavoreceu a confiabilidade do experimento.

Um possível trabalho futuro, que pode aumentar a qualidade das sugestões produzidas pelo VCC, é a análise do código fonte levando em consideração as estruturas de repetição e principalmente de condição. Com isso, uma estrura condicional, que possui mais de uma alternativa, não seria vista como uma única sequência de chamadas, e sim como sequências independentes. Além disso, outra possível melhoria, com o intuito de aumentar a usabilidade da ferramenta, é o preenchimento do código fonte no editor da IDE, ao invés de exibir as sugestões em uma janela *popup*.

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] C. I. Ezeife, Yi Lu e Yi Liu. Plwap sequential mining: Open source code. *International Workshop on Open Source Data Mining: Frequent Pattern Mining Implementations*, 1:26–35, 2005.

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

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

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

[9] 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.

[10] Gail C. Murphy, Mik Kersten e Leah Findlater. How are java software developers using the eclipse ide? *IEEE Software*, 23:76–83, 2006.

[11] Andrew C. Myers, Joseph A. Bank e Barbara Liskov. Parameterized types for java. *ACM Symposium on Principles of Programming Languages (POPL)*, 24:132–145, 1997.

[12] Lukasz Opyrchal e Atul Prakash. Efficient object serialization in java. *IEEE International Conference on Distributed Computing Systems Workshops on Electronic Commerce and Web-based Applications/ Middleware*, 19:96–101, 1999.

[13] Jens Palsberg e C. Barry Jay. The essence of the visitor pattern. *IEEE. Int. Computer Software and Applications Conf. (COMPSAC)*, 22:9–15, 1998.

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

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

[16] 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.

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

[18] 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.

[19] 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: _____

Apêndice III

Questionário de Avaliação

Imagine que você está codificando e que acabou de escrever uma linha de código para abrir uma conexão com o banco de dados e que a IDE que você está utilizando sugira algo do tipo: desenvolvedores que abrem uma conexão com o banco de dados também fecham a conexão com o banco de dados. Este é o objetivo do *plugin Vertical Code Completion*, que estamos desenvolvendo como projeto final: completar o código com sugestões simples (como no exemplo anterior), assim como com sugestões mais complexas, indo além do atual Ctrl-Espaço.

Para avaliar o quanto as sugestões de códigos informadas são consistentes e fazem sentido no desenvolvimento, escolhemos você, desenvolvedor, que faz parte da equipe do IdUFF. Para isso, mineramos o repositório de códigos do sistema IdUFF para encontrar possíveis padrões de desenvolvimento.

Para medir a utilidade das sugestões encontradas, tanto para um desenvolvedor experiente, quanto para um desenvolvedor novo na equipe, as seguintes respostas podem ser dadas: não sei, discordo totalmente, discordo parcialmente, concordo parcialmente, concordo totalmente.

1) Usuários que chamam o método:

org.mockito.internal.progress.NewOngoingStubbing¡Integer¿.thenReturn(java.lang.Integer)

Também costumam chamar:

org.mockito.Mockito.when(java.lang.Integer)

( ) não sei
( ) discordo totalmente
( ) discordo parcialmente
( ) concordo parcialmete
( ) concordo totalmente

Comentário: _____

2) Usuários que chamam o método:

br.uff.iduff2.managedbeans.BaseMB.error(java.lang.String)

Também costumam chamar:

java.lang.Throwable.getMessage()

( ) não sei
( ) discordo totalmente
( ) discordo parcialmente
( ) concordo parcialmete
( ) concordo totalmente

Comentário: _____

3) Usuários que chamam o método:

br.uff.commons.utils.oracle.ConexaoOracle.getInstance()

Também costumam chamar:

java.sql.Connection.prepareStatement(java.lang.String)

( ) não sei
( ) discordo totalmente
( ) discordo parcialmente
( ) concordo parcialmete
( ) concordo totalmente

Comentário: _____

4) Usuários que chamam o método:

org.hibernate.Query.setParameter(I, java.lang.Object)

Também costumam chamar:

org.hibernate.Query.list()

( ) não sei
( ) discordo totalmente
( ) discordo parcialmente
( ) concordo parcialmete
( ) concordo totalmente

Comentário: _____

5) Usuários que chamam o método:

br.uff.iduff2.relatorio.RelatorioFactory.getRelatorio(I)

Também costumam chamar:

br.uff.iduff2.relatorio.Relatorio.gerarRelatorio(java.util.List, java.util.Map)

( ) não sei
( ) discordo totalmente
( ) discordo parcialmente
( ) concordo parcialmete
( ) concordo totalmente

Comentário: _____

6) Usuários que chamam o método:

br.uff.iduff2.modelo.academico.Turma.getDisciplina()

Também costumam chamar:

br.uff.iduff2.modelo.academico.Disciplina.getCargaHorariaTeorica()

( ) não sei
( ) discordo totalmente
( ) discordo parcialmente
( ) concordo parcialmete
( ) concordo totalmente

Comentário: _____

7) Usuários que chamam o método:

br.uff.publico.core.model.Identificacao.getNome()

Também costumam chamar:

br.uff.publico.core.model.Identificador.getIdentificacao()

( ) não sei
( ) discordo totalmente
( ) discordo parcialmente
( ) concordo parcialmete
( ) concordo totalmente

Comentário: _____

8) Usuários que chamam o método:

java.lang.String.isEmpty()

Também costumam chamar:

br.uff.iduff2.managedbeans.BaseMB.info(java.lang.String)

( ) não sei
( ) discordo totalmente
( ) discordo parcialmente
( ) concordo parcialmete
( ) concordo totalmente

Comentário: _____

9) Usuários que chamam o método:

java.util.logging.Logger.log(java.util.logging.Level, java.lang.String, java.lang.Throwable)

Também costumam chamar:

java.lang.Throwable.getMessage()

( ) não sei
( ) discordo totalmente
( ) discordo parcialmente
( ) concordo parcialmete
( ) concordo totalmente

Comentário: _____

10) Usuários que chamam o método:

br.uff.iduff2.managedbeans.BaseMB.info(java.lang.String)

Também costumam chamar:

java.lang.Throwable.getMessage()

( ) não sei
( ) discordo totalmente
( ) discordo parcialmente
( ) concordo parcialmete
( ) concordo totalmente

Comentário: _____