CIDE+: Uma Ferramenta para Extração Semi-automática de Linhas de Produtos de Software Usando Coloração de Código

Virgílio Borges de Oliveira[1], **Rógel Garcia**[2], **Marco Túlio Valente**[2]

[1] Instituto de Informática, PUC Minas
[2] Departamento de Ciência da Computação, UFMG

virgilio@cotemig.com.br, idrogelgarcia@gmail.com, mtov@dcc.ufmg.br

Resumo. *The extraction of real-world software product lines is a complex and time-consuming task. In order to accelerate this task, this paper presents a tool – called CIDE+ – that automates the annotation of optional features of existing systems. For this purpose, CIDE+ supports the semi-automatic association of background colors to the lines of code that implement an optional feature. CIDE+ has been successfully applied in extracting features of three non-trivial systems: Prevayler (an in-memory database), JFreeChart (a chart library) and ArgoUML (an editor for UML diagrams).*

1 Introdução

A extração de linhas de produtos de software a partir de sistemas existentes é uma tarefa complexa e custosa [4]. Em primeiro lugar, desenvolvedores precisam identificar os componentes responsáveis pela implementação de cada *feature* da linha de produtos. Em seguida, eles devem identificar aqueles trechos de código que referenciam os componentes descobertos no passo anterior. Por último, deve-se anotar esses trechos de alguma forma ou movê-los para novas unidades de modularização. Assim, a fim de acelerar a extração de linhas de produtos, descreve-se neste artigo uma ferramenta desenvolvida com o objetivo de automatizar – de forma parcial – a anotação de *features* opcionais de sistemas existentes.

A ferramenta proposta é baseada em uma extensão da plataforma Eclipse, chamada CIDE (*Colored IDE*) [3]. CIDE estende o ambiente Eclipse com a possibilidade de se associar cores de fundo a trechos de código que implementam *features*. Portanto, em vez de anotações textuais, tais como `#ifdef` e `#endif`, desenvolvedores podem usar anotações visuais para delimitar as linhas de código que pertencem a uma determinada *feature*. Além disso, o ambiente CIDE permite a geração de projeções de um sistema nas quais todo código anotado com uma determinada cor é removido. Essa forma de projeção permite a geração de produtos da linha de software sem uma determinada *feature*.

A ferramenta descrita no artigo – chamada CIDE+ – estende o ambiente CIDE com a capacidade de anotação semi-automática de código relativo a *features*. Ou seja, usando apenas o CIDE, desenvolvedores devem manualmente localizar e mudar a cor de fundo de todos os trechos de código responsáveis pela implementação de uma *feature* que desejam incluir na linha de produtos. Normalmente, essa tarefa é custosa, tediosa e sujeita a erros. Por outro lado, usando CIDE+, tais tarefas são automatizadas. Para isso, inicialmente a ferramenta requer que engenheiros de linhas de produtos forneçam como entrada um conjunto de elementos sintáticos – incluindo, por exemplo, pacotes, classes, métodos e campos – responsáveis pela implementação de uma *feature*. Esses elementos

são chamados de sementes da *feature*. Em seguida, a ferramenta CIDE+ trata de colorir todos os elementos do programa que referenciam direta ou indiretamente os elementos definidos como sementes.

O restante deste resumo estendido está organizado conforme descrito a seguir. A Seção 2 descreve o processo de extração de *features* automatizado pela ferramenta CIDE+. Na Seção 3, apresenta-se um pequeno exemplo de uso da ferramenta. A Seção 4 sumariza os resultados obtidos na aplicação da ferramenta em três sistemas não-triviais. Por fim, a Seção 5 discute ferramentas relacionadas e a Seção 6 conclui o artigo.

2 Extração de Features com CIDE+

CIDE+ agrega ao ambiente CIDE a capacidade de anotação automática de trechos de código usando cores de fundo. A Figura 1 apresenta o processo iterativo e semi-automático para anotação visual de *features* pressuposto pela ferramenta. Os passos desse processo são discutidos nos parágrafos seguintes.

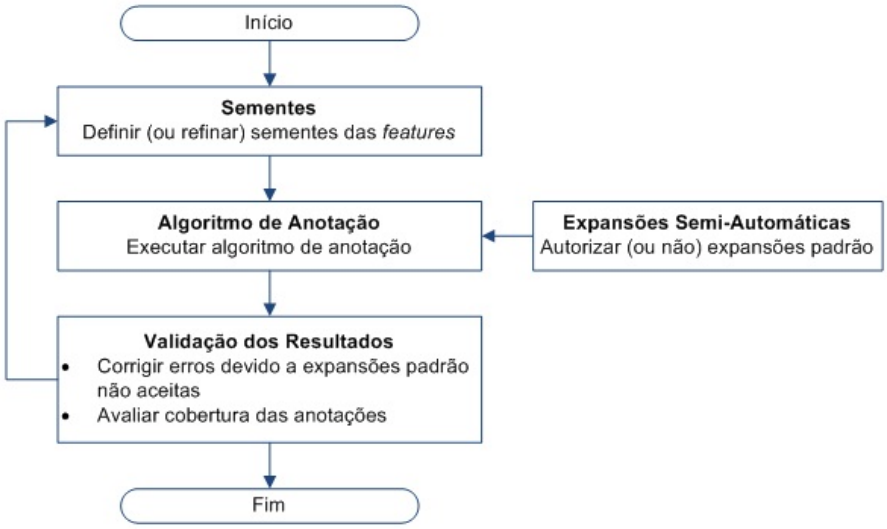

Figura 1. Processo iterativo e semi-automático para extração de features

Definição das Sementes: Inicialmente, desenvolvedores devem informar as sementes da *feature* a ser extraída. Sementes podem incluir os seguintes elementos sintáticos: campos, métodos, classes ou interfaces e pacotes. Portanto, a ferramenta pressupõe que os engenheiros tenham um conhecimento básico da arquitetura e do projeto do sistema alvo da extração. Mais especificamente, eles devem ser capazes de informar os elementos do programa que são integralmente responsáveis pela implementação da *feature* que desejam extrair do sistema.

Algoritmo de Anotação: Tendo como entrada as sementes definidas, esse algoritmo trata de colorir todas as partes do código que referenciam de forma direta ou indireta os elementos classificados como sementes. Basicamente, o algoritmo proposto é um algoritmo de ponto-fixo, com duas fases. Na primeira fase, chamada de propagação de cores, os elementos do programa que referenciam as sementes são visualmente anotados com uma cor informada pelo engenheiro da linha de produtos. Por exemplo, suponha que o método

m_1 tenha sido classificado como uma semente. Portanto, nessa primeira fase, qualquer chamada de m_1 será visualmente anotada com a cor fornecida. Na segunda fase, chamada de expansão de cores, o algoritmo verifica se o contexto léxico dos elementos anotados na primeira fase também pode ser colorido. Suponha, por exemplo, que a implementação de um método m_2 apenas inclui uma chamada a m_1, a qual foi anotada na primeira fase do algoritmo. Nesse caso, a implementação de m_2 também será colorida (assim como todas as suas chamadas). Como o foco do presente artigo é apresentar a ferramenta CIDE+, não são fornecidos mais detalhes sobre o algoritmo de anotação. O leitor interessado em tais detalhes deve se dirigir a um outro trabalho de nosso grupo [2].

Expansões Semi-automáticas: Em situações especiais, o algoritmo proposto pode gerar código com erros sintáticos ou de tipo. Suponha, por exemplo, a expressão `opcao == algumaConstante`, onde `opcao` foi anotada mas `algumaConstante` não foi. Dessa forma, a projeção incluindo apenas `== algumaConstante` não irá compilar por apresentar erros de sintaxe. Em outro exemplo, suponha que todos os comandos `return` de um determinado método foram anotados, mas os demais comandos não. Após a remoção do código anotado, um erro de tipo será reportado (comando `return` ausente). Em tais situações, a aplicação de qualquer tipo de "expansão corretiva" é uma tarefa desafiadora e complexa.

Quando uma anotação leva a erros de sintaxe ou de tipo e nenhuma expansão pode ser seguramente aplicada, o algoritmo proposto gera uma mensagem explicando o erro. Posteriormente, é sugerida uma expansão padrão para tratar o erro detectado, como descrito na Tabela 1. Por exemplo, a ferramenta pode sugerir a anotação de toda uma expressão (caso ela tenha sido parcialmente anotada) ou de todo um método (caso todos os seus comandos `return` tenham sido anotados). Se o desenvolvedor aceitar as sugestões, a ferramenta automaticamente aplica tais expansões ao código, a fim de eliminar os erros.

	Definição	**Expansão Padrão**
SE1	Apenas partes de uma expressão foram anotadas	Anotar toda a expressão
SE2	Os comandos `return` de um método foram anotados, mas o método tem outros comandos que não foram anotados	Anotar todo o corpo do método
SE3	Um parâmetro de chamada foi anotado, mas o parâmetro formal associado a ele não foi anotado	Anotar toda a chamada
SE4	O lado direito de uma expressão foi anotado mas o lado esquerdo não anotado	Anotar o lado esquerdo e suas referências

Tabela 1. Expansões Semi-automáticas

Avaliação dos Resultados: Finalizada a execução do algoritmo de anotação, resta ao engenheiro responsável pela extração duas tarefas. Primeiro, ele deve detectar e corrigir eventuais erros de sintaxe que tiverem sido inseridos no código. Em geral, esses erros são inseridos quando o engenheiro optou por não aceitar uma das expansões padrão sugeridas pela ferramenta durante a execução do algoritmo de anotação. Em seguida, o engenheiro

deve avaliar a cobertura das anotações visuais inseridas no código, isto é, se todas as linhas de código dedicadas à implementação da *feature* foram de fato coloridas. Para isso, recomenda-se que ele gere uma projeção do sistema sem a *feature* em questão e que execute o produto gerado, de forma a se certificar de que a *feature* realmente não se encontra mais presente no sistema. Nessa execução, ele pode perceber que ainda existe parte da funcionalidade provida pela *feature* no produto gerado. Nesse caso, ele deve refinar as sementes inicialmente escolhidas e iniciar um novo ciclo do processo de extração.

Em resumo, o processo proposto possui as seguintes características: (a) semi-automático, pois requer a definição das sementes das *features* extraídas e a aprovação de determinadas expansões de cores; (b) iterativo, pois pode requerer mais de um ciclo de execução, até obter uma cobertura de todo o código responsável pela *feature*. O grau de cobertura obtido – e o número de iterações requeridas – depende basicamente de sementes que incluam todas os elementos do programa que respondem pela implementação da *feature* sob extração.

3 Exemplo de Uso

Suponha uma classe hipotética `Stack` com um método `push`. Suponha que essa classe faça parte de um sistema maior com as *features* descritas na Tabela 2. Além do nome das *features* e de uma breve descrição do papel das mesmas na classe `Stack`, essa tabela mostra também as sementes escolhidas como entrada para a ferramenta CIDE+.

Feature	Descrição	Sementes
Logging	*Log* de eventos internos da pilha	Classe `Log`
Snapshot	Salva elementos em meio persistente	Método `snapshot`
Multithreading	Versão *thread-safe* da pilha	Classe `Lock`

Tabela 2. Features da classe Stack e suas respectivas sementes

A Figura 2 apresenta o código da pilha colorido pela ferramenta CIDE+. Nesse exemplo, chama a atenção a chamada de *log* realizada no interior do primeiro `if` do método `push`. Como essa chamada está aninhada em um bloco de código relativo à *feature Multithreading*, ela recebeu duas cores: a cor de *Logging* (verde) e a cor de *Multithreading* (vermelho). Quando isso ocorre, a ferramenta usa a cor resultante da mistura física das duas cores entrelaçadas (isto é, vermelho+verde = amarelo). Na parte inferior da Figura 2 mostra-se uma representação da AST do programa gerada pela ferramenta CIDE+. Veja que os nodos responsáveis pelos blocos de código coloridos no editor são também mostrados coloridos.

4 Estudos de Caso

CIDE+ foi usada com sucesso para extrair *features* de três sistemas: Prevayler (um banco de dados em memória principal), JFreeChart (uma biblioteca gráfica) e ArgoUML (um editor de diagramas UML). A Tabela 3 sumariza a experiência de uso de CIDE+ nesses três sistemas. Para cada sistema, a tabela informa o tamanho de seu código fonte (em megabytes), uma breve descrição das *features* extraídas, o percentual do código fonte anotado para extração de cada uma das *features*, o número de expansões semi-automáticas (SE) requeridas durante a extração e o número de iterações que foram requeridas até se obter uma cobertura completa do código de cada *feature* extraída.

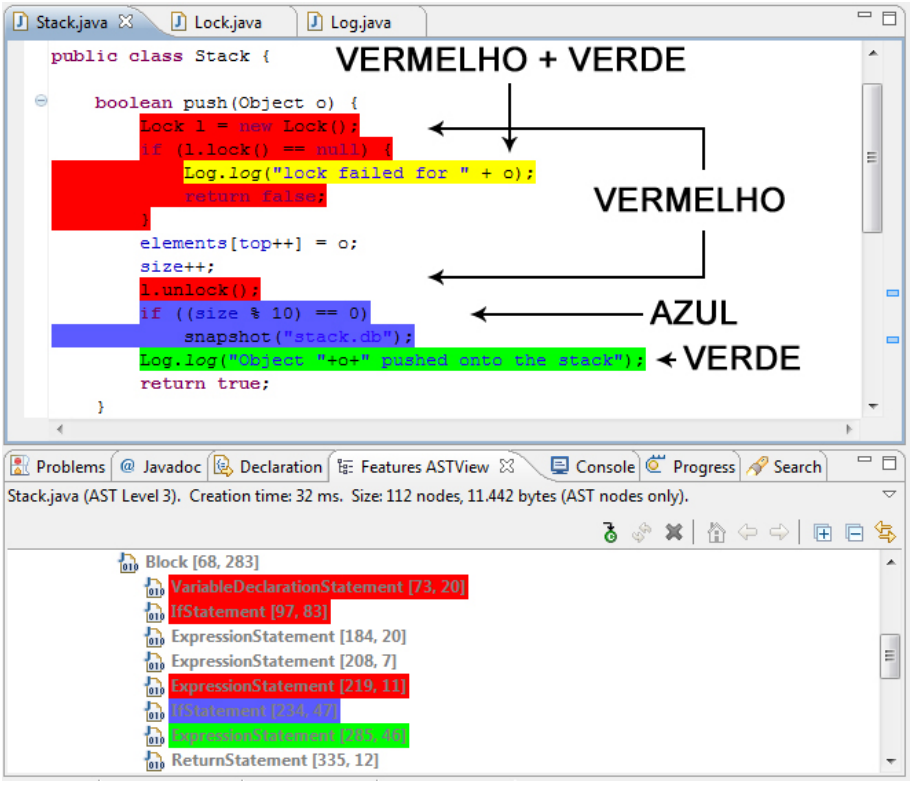

Figura 2. Método push com features anotadas pela ferramenta proposta

Sistema	Tam. (MB)	Feature	% Tam.	# SE	# Iter.
Prevayler	0.16	Monitor	4.0	3	1
		Replication	6.7	0	1
		Censoring	2.7	1	1
JFreeChart	7.5	Pie Charts	5.0	21	1
		3D charts	2.3	7	1
ArgoUML	8.95	StateDiagrams	3.7	100	2
		ActivityDiagrams	1.7	40	1
		Design Critics	14.9	136	2
		Logging	1.8	19	2

Tabela 3. Resultados sumarizados dos estudos de caso

5 Ferramentas Relacionadas

Pelo menos que seja de nosso conhecimento, não existem outras ferramentas para extração automática de *features* em linhas de produtos usando anotações (sejam elas textuais, como diretivas de pré-processamento, ou visuais, como cores de fundo).

Sendo assim, as ferramentas mais próximas são aquelas projetadas para apoiar a extração física de *features* para unidades de modularização, normalmente para aspectos. Como exemplo, podemos citar as ferramentas AOP-Migrator [1] e FOR [4]. No entanto, essas ferramentas requerem uma participação e envolvimento muito maior dos engenheiros responsáveis pelo processo de extração. O principal motivo é que linguagens como AspectJ se baseiam em um modelo de granularidade grossa para extensão de progra-

mas [3]. Em outras palavras, essas linguagens pressupõem que o código responsável pelas *features* esteja presente em determinadas localizações estáticas do código fonte. Como nem sempre isso ocorre, essas ferramentas requerem um passo preliminar de preparação do código fonte visando a sua extração para unidades próprias de modularização. Em geral, esse processo de preparação requer um conhecimento detalhado do código, que impede a sua automatização [6, 5].

6 Comentários Finais

Neste artigo, foi apresentada a ferramenta CIDE+, a qual estende o ambiente CIDE com a coloração automática de código responsável pela implementação de *features* em linhas de produtos de software. Mais informações sobre a ferramenta podem ser obtidas em: `http://www.dcc.ufmg.br/~mtov/cideplus`.

Agradecimentos: Este trabalho foi apoiado pela FAPEMIG, CAPES e CNPq.

Referências

[1] David Binkley, Mariano Ceccato, Mark Harman, Filippo Ricca, and Paolo Tonella. Tool-supported refactoring of existing object-oriented code into aspects. *IEEE Transactions Software Engineering*, 32(9):698–717, 2006.

[2] Virgilio Borges and Marco Tulio Valente. Coloração automática de variabilidades em linhas de produtos de software. In *III Simpósio Brasileiro de Componentes, Arquiteturas e Reutilização de Software (SBCARS)*, pages 67–80, 2009.

[3] Christian Kästner, Sven Apel, and Martin Kuhlemann. Granularity in software product lines. In *30th International Conference on Software Engineering (ICSE)*, pages 311–320, 2008.

[4] Jia Liu, Don Batory, and Christian Lengauer. Feature oriented refactoring of legacy applications. In *28th International Conference on Software Engineering (ICSE)*, pages 112–121, 2006.

[5] Marcelo Nassau, Samuel Oliveira, and Marco Tulio Valente. Guidelines for enabling the extraction of aspects from existing object-oriented code. *Journal of Object Technology*, 8(3):1–19, 2009.

[6] Marcelo Nassau and Marco Tulio Valente. Object-oriented transformations for extracting aspects. *Information and Software Technology*, 51(1):138–149, 2009.