CONSIDERAÇÕES ACERCA DA MINERAÇÃO DE ASPECTOS
CONSIDERATIONS ABOUT ASPECT MINING

Fernando Pinciroli*

RESUMO

O emprego do paradigma orientado a aspectos aporta um salto de qualidade ao desenvolvimento do software ao resolver o espalhamento e o enredo do código por meio da separação das incumbências transversais. Assim, o código acrescenta a sua manutenibilidade, reusabilidade, compreensibilidade, etc. As aplicações herdadas, desenvolvidas com o paradigma de orientação a objetos, poderiam gozar destes benefícios, não em forma plena como se tivessem sido pensados com orientação a aspectos desde um começo, mais é possível avançar na optimização do código mediante a separação das incumbências entrelaçadas e espalhadas com a aplicação de duas técnicas complementárias: a mineração de aspectos, com a qual são achadas as incumbências transversais, e a refatoração do código encapsulando-o em aspectos. Neste trabalho recorreremos as técnicas de mineração de aspectos existentes até o momento.

Palavras chave: Mineração de aspectos. Incumbência transversal. Aspecto. Encapsulamento de aspectos. Refatoração de aspectos. Métrica. Requisito funcional. Requisito não funcional.

ABSTRACT

The use of the aspect-oriented paradigm strongly improves the quality of software development since it brings a solution to the problem of code scattered and tangled by means of cross-cutting concerns separation. Thus, the code improves its maintainability, reusability, understandability, etc. Legacy applications, developed with the object-oriented paradigm, could make the most with these benefits, not as well as it would have been developed with aspect-oriented techniques from the beginning, but it is possible to optimize the code by separating the scattered and tangled concerns with the application of two complementary techniques: aspect mining, with which cross-cutting concerns are found, and code refactoring in order to encapsulate those concerns into aspects. In this paper we will use see the state of art of aspect mining techniques.

Keywords: Aspect mining. Cross-cutting concern. Aspect. Aspect encapsulation. Aspect refactoring. Metric. Functional requirement. Non-functional requirement.

* Doutorando pelo programa de Doutorado em Ciências da Informática da Universidade Nacional de San Juan, Argentina, e Decano da Faculdade de Informática e Desenho da Universidade Champagnat, Godoy Cruz, Argentina. pincirolifernando@uch.edu.ar

Introdução

O paradigma da orientação a aspectos aporta uma série de benefícios bem descritos na bibliografia e nos trabalhos disponíveis. O desenvolvimento de sistemas orientados a aspectos tem como resultado, então, sistemas de maior qualidade, que podem-se obter aplicando as técnicas e ferramentas que oferecem os diferentes autores.

Além das atividades gerais que devem-se levar a cabo para o desenvolvimento de software existem outras específicas que pertencem ao paradigma de aspectos e que são a detecção das incumbências, sua separação, a posterior composição para a integração do sistema, a resolução de possíveis conflitos e a aplicação das melhores práticas que podem-se recomendar em cada uma das fases do ciclo de vida do desenvolvimento de software. O termo "concern" também pode ser traduzido como "interesse".

A primeira destas atividades específicas, que é a detecção ou o descobrimento das incumbências, pode-se fazer tanto para o desenvolvimento de um novo sistema como para a reengenharia de um sistema herdado. Em cada um destes casos conta-se com uma fonte de informação de diferentes características. No primeiro dos casos muito provavelmente se poderá contar com uma informação mais adequada, já que vai produzindo-se de forma ordenada e em base às técnicas orientadas a aspectos disponíveis, mas no segundo caso se dependerá da documentação que pudera estar disponível do sistema preexistente, donde a maior informação que se tem é o código fonte do sistema mesmo.

Refatorar o código dos sistemas herdados detectando e encapsulando as incumbências transversais como aspectos é uma forma de melhorar a qualidade de tais sistemas, favorecendo sua compreensibilidade, manutenibilidade, evolução, etc.

Assim, a atividade de descobrir incumbências no código e nos produtos intermédios dos sistemas preexistentes conhece-se como "mineração de aspectos", que é complementada com a refatoração de aspectos que consiste no encapsulamento dos aspectos achados.

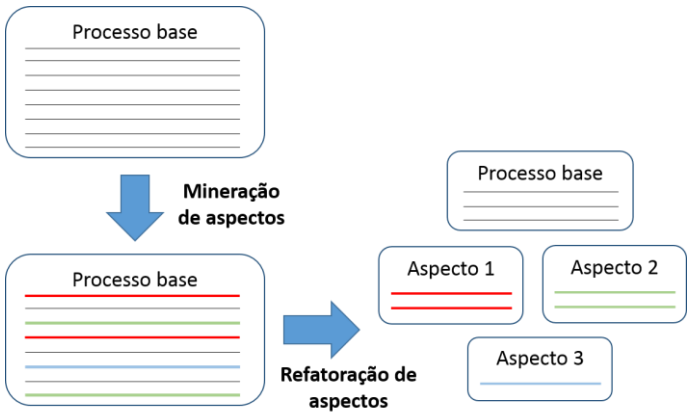

Figura 1. Mineração e refatoração de aspectos

Dependendo de sobre que informação de entrada se realiza a mineração de aspectos, podemos classificar as técnicas em:

- **Mineração de aspectos iniciais:** consiste em buscar aspectos nos artefatos que são produzidos nas etapas iniciais do ciclo de vida do desenvolvimento de software, tarefa que é um pouco mais difícil em sistemas herdados, e que deve brigar com a complexidade do tratamento de textos expressados em língua natural.

- **Emprego de técnicas semiautomáticas de mineração ao nível de código:** realizam uma busca semiautomática de incumbências no código, que precisam de um tratamento posterior ao complementário ao requerer algum tipo de assistência humana, que podem gerar falsos positivos e falsos negativos, etc.

Estas últimas técnicas, que estão baseadas nas técnicas de mineração de dados adaptadas para a orientação a aspectos, buscam os indícios de espalhamento e entrelaçamento que pudesse existir no código.

Mineração de aspectos iniciais

O emprego do paradigma de aspectos nas etapas iniciais do ciclo de vida do desenvolvimento de software é conhecido como “aspectos iniciais” (early aspects) e cobre as

fases do ciclo de vida desde o início até a conformação da especificação de requisitos de software completa. Desta maneira, é possível encontrar técnicas de mineração de aspectos nos produtos intermédios que são obtidos nas etapas iniciais do ciclo de vida em lugar de o fazer sobre o código fonte.

Podemos classificar estas técnicas em dos grandes grupos: as que realizam a mineração de aspectos nas especificações de requisitos de software e as que o fazem nos modelos de negócio.

a) Mineração de aspectos nas especificações de requisitos de software

As técnicas mais difundidas são:

- Análise de documentos relacionados: emprega técnicas de processamento de língua natural para identificar aspectos de uma maneira semiautomática [1].

- Clarke y Baniassad aportam um método para identificar incumbências nos requisitos na sua técnica Theme/Doc [2].

- Rashid et al. apresentam una técnica para derivação de incumbências com AORE [3].

- Bass et al. aportam um enfoque baseado num razoamento arquitetônico pela identificação de incumbências [4].

- Rago et al. propõem um método para obter incumbências a partir das especificações textuais dos casos de uso empregando técnicas de processamento de língua natural y desambiguação do sentido das palavras [5] e [6].

Existem ferramentas que intentam automatizar o processo de mineração de aspectos nas especificações textuais de requisitos de software. Por exemplo, Sampaio et al. [7] oferecem uma ferramenta automatizada pela detecção de aspectos iniciais no texto. Sua ferramenta, chamada EA-Miner, está baseada na análise das especificações de requisitos textuais mediante o processador de língua natural WMATRIX, que faz um analise (parsing) do texto e compõe um modelo interno específico. A ferramenta também

oferece uma interface para que o usuário veja este modelo interno, baseado na técnica AORE (Aspect-Oriented Requirement Engineering), para melhorar assim os resultados obtidos no modelo.

Sardinha et al. [8] propõem uma ferramenta automatizada, chamada EA-Analyzer, pela identificação de conflitos em especificações de requisitos orientados a aspectos em texto em língua natural. Está baseada no método de aprendizado bayesiano chamado Naïve Bayes. Tem a vantagem de que é requerida uma quantidade pequena de dados para o treinamento do classificador Naïve Bayes e pode-se personalizar o treinamento para cada organização de modo a poder detectar conflitos nas especificações de requisitos escritas pelo seu domínio de problema específico. Pela obtenção do conjunto de dados de treinamento, EA-Analyzer oferece uma interface que ajuda o usuário a etiquetar facilmente os dados de interesse e os possíveis conflitos. A ferramenta elabora um modelo de "sacos de palavras" com os exemplos que foram fornecidos: um saco conterá os elementos conflitivos e o outro os elementos não conflitivos; é possível que alguns termos estejam contidos nos dois sacos, mais o classificador fará as suas determinações em função da quantidade de palavras presentes em cada saco que acompanham as primeiras. Se bem foram realizados testes em especificações pequenas, os resultados foram alentadores, com uma precisão na classificação superior aos 90% na maioria dos casos.

b) Mineração de aspectos nos modelos de negócio

Os aspectos iniciais há tempo que avançaram mais na direção águas acima no ciclo de vida do desenvolvimento de software e pode-se encontrar uma boa quantidade de literatura de aspectos na etapa de modelagem de negócio. Assim, as técnicas de mineração de aspectos são susceptíveis de serem aplicadas também neste âmbito.

Amin Jalali [9][10] propõe um método de mineração de aspectos na modelagem de negócio que tem quatro passos:

1. descobrimento de incumbências transversais
2. eliminação de incumbências transversais

3. descobrimento de processos de negócio
4. descobrimento de relações

No primeiro lugar se elabora um repositório do processo de negócio baseado no standard XES, acrónimo de eXtensible Event Stream, que foi adoptado pela "task force" de mineração de processos da IEEE. Este repositório armazena os diferentes cenários de execução de cada processo de negócio e sobre o qual são executados o algoritmo difuso, que permite realizar mineração de processos sobre informação "com ruído", e o algoritmo alfa de mineração de processos, que realiza mineração de processos sobre informação "sem ruído".

Pelo descobrimento das incumbências transversais procura-se detectar, mediante as técnicas de mineração de processos mencionadas, atividades duplicadas dentro de um mesmo processo; ainda não tem um suporte razoável pela detecção entre processos. A continuação se realiza a eliminação das incumbências do fluxo principal dos processos em forma cíclica. O terceiro passo consiste em encontrar os processos de negócio "limpos" de aspectos. Por último, se estabelecem as relações entre as incumbências e os processos de negócio mediante a incorporação dos *pointcuts* correspondentes.

Pela sua parte, Di Francescomarino e Tonella [11] utilizam técnicas de "análise formal de conceitos" (FCA) para elaborar uma rede de conceitos a partir do processamento automático do repositório dos modelos de processos de negócio. O exemplo que aportam no seu trabalho é pequeno, razão pela qual é requerida uma aplicação mais importante que aquela da sua proposta ao fim de poder avaliar com maior rigor os resultados obtidos. Não obstante isto, os resultados do exemplo apresentado dão uma precisão do 75%, um recall do 100% e um F-measure de 0,86. Se em lugar de um limite de extensão de 3 empregam um limite de 4, os valores de precisão, recall e F-measure passam a ser 66%, 66% e 0,66 respectivamente. O limite de extensão corresponde à quantidade de elementos compartindo a mesma semântica.

Emprego de técnicas semiautomáticas de mineração ao nível de código

O segundo conjunto de técnicas de mineração de aspectos corresponde à mineração no código fonte mediante o emprego de técnicas automáticas e semiautomáticas, com as que se utilizam diversos enfoques para o descobrimento das incumbências. A continuação, e a partir de diferentes recopilações [12], mencionaremos os principais dentre a grande quantidade de propostas existentes:

a) Busca baseada em consultas

Este enfoque consiste em buscar no código fonte uma determinada informação que fornece o usuário e que é conhecida como "semente". A partir das descobertas obtidas se vai refinando a busca e se conforma assim um modelo de incumbências em forma iterativa.

As buscas se realizam com ferramentas específicas de rastreamento no código fonte como Feature Exploration and Analysis Tool, Aspect Browser, The Aspect Mining Tool, Prism, JQuery, Multi-Visualizer, Soul, QJBrowser, etc.

Este enfoque tem vários inconvenientes, como o fato de que o usuário deve ter um conhecimento prévio do modelo para poder realizar buscas mais eficazes, os resultados dependerão das sementes que sejam introduzidas, o processo pode demandar muito tempo e requer muito trabalho manual.

b) Técnicas automatizadas

Entre as principais técnicas automatizadas se acham as seguintes:

- **Análise formal de conceitos:** parte de um conjunto de elementos e de um conjunto de propriedades desses elementos e procura elaborar conceitos a partir dos maiores agrupamentos de elementos que possuam os mesmos valores pelas mesmas propriedades.

- **Análise de traças de execução:** Tonella e Ceccato [13] assumem que os cenários dos casos de uso são bons indicadores de incumbências

transversais; aplicam a técnica de "análise formal de conceitos" aos casos de uso tomando a estes como os elementos e aos seus métodos como as propriedades. Desta maneira, os mesmos métodos presentes em diferentes classes indicam o espalhamento e os métodos das mesmas classes que aparecem em diferentes casos de uso põem de manifesto o entrelaçamento.

- **Análise de leque de entrada (fan-in):** Marin et al. [14] propõem esta métrica como um indicador do espalhamento, já que partem da premissa de que os métodos que são chamados desde diferentes contextos o põem de manifesto.

- **Análise de padrões recorrentes de traças de execução:** consiste na revisão do código para tratar de encontrar pares de métodos que se levam a cabo um depois do outro em forma reiterada [15].

- **Análise de identificadores:** parte da suposição de que os métodos que pertencem às mesmas incumbências possuem nomes semelhantes, razão pela qual realiza os agrupamentos em função dos nomes das diferentes entidades de programação [16].

- **Análises do historial de versões:** realiza um análise da evolução das diferentes versões armazenadas dos arquivos de um sistema para identificar as mudanças que puderam ter sido feitas, as que se pegam como possíveis fontes de aspectos [17].

- **Detecção de clones:** parte do suposto de que as incumbências transversais espalhadas e entrelaçadas no código provocam duplicações de código [18].

- **Análise de clusters:** os clusters são conjuntos de elementos que possuem similitudes entre si e diferencias com os elementos de outros conjuntos; assim, se parte dum conjunto inicial de elementos e a informação da distância

entre eles para conformar grupos de elementos pertos em base a alguma função. Shepherd e Pollock [19] conformam os clusters em base aos métodos, Moldovan e Șerban [20] propõem realizar o clustering em função de atributos dos sintomas do espalhamento de código em base a um modelo espacial de vectores e usando algoritmos de clustering como k-means, fuzzy, aglomeração em jerarquias, clustering genético, etc. e também aportam um enfoque baseado em grafos [21]. A sua vez, He e Bai [22] empregam o algoritmo de clustering baseado na técnica "análise de padrões recorrentes de traças de execução" antes mencionada. McFadden [23] utiliza uma técnica de clustering baseada num espaço vectorial de características que considera mais efetivo do que o emprego das técnicas de clustering em jerarquias ou partições. É possível encontrar muitas outras propostas de clustering.

- **Análise de enlaces:** esta técnica está baseada na conformação de redes de elementos relacionados provenientes da tecnologia de recuperação de informação [24].

- **Análise de métodos únicos:** consiste em detectar as entidades que são chamadas desde numerosos pontos do código, proposto por Gybels e Kellens [25].

- **Análise de chamadas agrupadas:** parte da premissa de que existem grupos de métodos que invocam sistematicamente a um mesmo conjunto de métodos, mediante a exploração de métodos que sejam chamados pelos invocadores. Esta técnica foi proposta por Marin et al. [26].

- **Análise de localizadores de redireciones:** também proposto por Marin et al. [26], está baseada na busca dos redirecionamentos próprios do emprego de padrões de desenho GOF, como acontece por exemplo com o padrão Decorator, onde existem métodos que, por sua vez, delegam suas

responsabilidades em outros métodos de classes especificamente estabelecidas para tal fim.

Comparação de técnicas

Kellens et al. [27] realizam um análise de técnicas que transcrevemos neste ponto. É possível encontrar um análise mais exaustivo em [9].

Tabela 1. Tipos de dados e de análise que emprega cada técnica de mineração de aspectos

Técnica	**Tipo de dado de entrada**		**Tipo de análise**	
	Estático	**Dinâmico**	**Baseado em token**	**Estrutural/funcional**
Padrões de execução	X	X		X
Análise dinâmico		X		X
Análise de identificadores	X		X	
Pistas da linguagem	X		X	
Métodos únicos	X			X
Clustering de métodos	X		X	
Clustering de chamadas	X			X
Análise de leque de entrada	X			X
Detecção de clones – PDG[2]	X			X
Detecção de clones – token	X		X	
Detecção de clones – AST[3]	X			X

Na tabela 1 pode-se observar que a maioria das técnicas possuem entrada de dados estática, tem muito poucas técnicas com entrada dinâmica.

[2] PDG: grafos de dependências de programas
[3] AST: árvores de sintaxes abstrata

Tabela 2. Granularidade e sintoma que detecta cada técnica de mineração de aspectos

Técnica	**Granularidade**		**Sintoma**	
	Método	**Porção de código**	**Espalhamento**	**Entrelaçamento**
Padrões de execução	X		X	
Análise dinâmico	X		X	X
Análise de identificadores	X		X	
Pistas da linguagem	X		X	
Métodos únicos	X		X	
Clustering de métodos	X		X	
Clustering de chamadas	X		X	
Análise de leque de entrada	X		X	
Detecção de clones		X	X	

Desde o ponto de vista da granularidade, observa-se que tem poucos métodos que podem chegar a níveis baixos, já que a maioria aponta a métodos completos e não a porções de código menores. Isso explica porque tem tão poucas técnicas que ajudem a detectar o entrelaçamento, como se observa nas colunas de sintomas da tabela.

Tabela 3. Validação das técnicas de mineração de aspectos

Técnica	**Maior caso estudado**	**Validação empírica**
Padrões de execução	3100 métodos / 82 KLOC	
Análise dinâmico	2800 métodos / 18 KLOC	
Análise de identificadores	2800 métodos / 18 KLOC	
Pistas da linguagem	10 KLOC	
Métodos únicos	3400 classes / 66000 métodos	
Clustering de métodos	2800 métodos / 18 KLOC	
Clustering de chamadas	12 métodos	
Análise de leque de entrada	2800 métodos / 18 KLOC y 172 KLOC	
Detecção de clones – PDG	38 KLOC	
Detecção de clones – token/AST	20 KLOC	X

Na tabela 3 observa-se que, se bem se trabalhou sobre casos importantes, existe muito pouca validação empírica.

Tabela 4. Considerações a priori sobre o código das que parte cada técnica

Técnica	**Considerações acerca do código**
Padrões de execução	O ordenamento das chamadas pela incumbência transversal é sempre o mesmo.
Análise dinâmico	Existe ao menos um caso que apresenta uma incumbência transversal e outro que não.
Análise de identificadores	Os nomes dos métodos que implementam as incumbências são semelhantes.
Pistas da linguagem	O contexto da incumbência tem chaves que são sinónimos pela incumbência transversal.
Métodos únicos	A incumbência é implementada por só um método.
Clustering de métodos	Os nomes dos métodos que implementam as incumbências são semelhantes.
Clustering de chamadas	A incumbência é implementada por chamadas aos mesmos métodos desde módulos diferentes.
Análise de leque de entrada	A incumbência é implementada em métodos separados que são chamados muitas vezes o tem muitos métodos implementando a incumbência chamam ao mesmo método.
Detecção de clones	A incumbência é implementada reutilizando uma certa porção de código.

Segundo a tabela 4, podemos ver que as técnicas não estão baseadas nas mesmas considerações acerca do código.

Tabela 5. Tipo de atividade manual que requer cada técnica

Técnica	Participação do usuário
Padrões de execução	Inspeciona os resultados dos padrões recorrentes.
Análise dinâmico	Seleciona os casos de uso e interpreta manualmente os resultados.
Análise de identificadores	Explora os aspectos detectados usando integração com IDE.
Pistas da linguagem	Interpreta manualmente os resultados das cadeias de texto.
Métodos únicos	Inspeciona os métodos únicos.
Clustering de métodos	Explora os aspectos detectados usando integração com IDE.
Clustering de chamadas	Inspeciona manualmente os clusters resultantes.
Análise de leque de entrada	Seleciona os candidatos da lista de métodos.
Detecção de clones	Explora e interpreta manualmente os clones que se descobriram.

Por último, a tabela 5 apresenta as diferentes necessidades de participação do usuário.

Inconvenientes detectados nas técnicas

Certamente as técnicas de mineração de aspectos foram melhorando substancialmente ao longo do tempo, embora ainda existem muitos desafios por resolver, e Cojocar afirma que se bem os resultados atuais são melhores, não são o suficientemente melhores ainda [12]. Mens et al. [28] realizaram um estudo onde põem de manifesto estas questões, destacam os inconvenientes que observam nas técnicas propostas e identificam sus causas. Os problemas que mencionam são:

- **Precisão pobre:** a porcentagem de aspectos candidatos relevantes é baixo com relação ao conjunto total de aspectos reportados pelas técnicas, que devolvem um grande número de falsos positivos que impactam negativamente na escalabilidade e na facilidade de uso.

- **Recall pobre:** a porcentagem de aspectos candidatos relevantes que são descobertos com respeito à totalidade dos aspectos candidatos presentes no código fonte é baixo.

- **Subjetividade:** para muitas das técnicas de mineração de aspectos seus resultados possuem certa ambiguidade; dependendo da persona e da definição de aspecto que usa tem diferencias entre aqueles que consideram que um aspecto é candidato e entre os que opinam que não é.

- **Falta de escalabilidade:** a necessidade de que o usuário esteja envolvido, devido a que as técnicas ainda são semiautomáticas, apresenta graves limitações à escalabilidade. Outra questão que também afeta à escalabilidade é a quantidade de tempo de processamento requerido.

- **Falta de validação empírica:** a dificuldade de medir o recall e outros fatores, como a subjetividade antes mencionada, atentam contra as possibilidades de validação, e esta última, por sua vez, contra o progresso da disciplina de mineração de aspectos.

Enfoques simétrico e assimétrico

Dentro da orientação a aspectos existem dois enfoques chamados simétrico e assimétrico. Ambos enfoques procuram obter aspectos, mas fazem a tarefa de jeito diferente, e os aspectos resultantes de cada enfoque são denominados pares e impares respectivamente.

Os aspectos pares são aspectos do domínio do problema, mais os ímpares não pertencem ao domínio do problema e pelo geral estão presentes em diferentes domínios. Por exemplo, num sistema de hotéis os aspectos pares poderiam ser “fazer check-in”, “fazer check-out”, “fazer uma reserva de habitação”, etc. Num sistema para universidades, os aspectos pares poderiam ser “cadastrar em uma matéria”, “dar um exame”, “obter um livro em empréstimo”, etc. Como é óbvio, cada um destes aspectos pertence a um domínio de

problema específico. E é possível notar, ao mesmo tempo, que os aspectos mencionados como exemplos tem que ver principalmente com requisitos funcionais.

Os aspectos ímpares, em contrário, estão presentes em numerosos domínios de problema diferentes. Exemplos de aspectos ímpares são "seguridade", "auditoria", "gestão de transações", etc. Estes aspectos poderiam se encontrar presentes nos dois domínios que mencionamos no parágrafo anterior, sistema para hotéis e sistema para universidades, como assim também em muitos outros domínios de problema. Estes aspectos, claramente, correspondem aos atributos de qualidade do software, também conhecidos como requisitos não funcionais.

A orientação a aspectos desenvolveu-se e evolucionou principalmente desde o enfoque assimétrico, ao redor dos aspectos assimétricos, razão pela qual muitos autores enfocaram-se mais que nada sobre o encapsulamento de requisitos não funcionais, e praticamente todas as técnicas que mencionamos neste trabalho facilitam a detecção deste tipo de aspectos. Em contrapartida, são muito poucos os trabalhos que contemplam a detecção de aspectos pares.

A separação de incumbências que termina na elaboração de aspectos simétricos é mais difícil de lograr, porque se bem o espalhamento destas incumbências pode ser menor, o entrelaçamento é bastante maior, o que faz que as técnicas de mineração de aspectos no código não sejam suficientemente eficazes para encontrar os aspectos simétricos, uma vez que a quantidade de propostas existentes é bastante reduzida com respeito àquelas que tem como objetivo detectar aspectos assimétricos. Assim, pelo momento, as técnicas mais adequadas para abordar o enfoque simétrico são aquelas que realizam a mineração nas etapas iniciais do ciclo de vida do desenvolvimento de software, principalmente na fase de elaboração das especificações de requisitos.

Refatoração

Uma vez detectados os aspectos deve-se refatorar o código de modo que fique corretamente constituído desde a perspectiva do paradigma de aspectos e, por suposto, em função da linguagem de programação orientado a aspectos em particular que se esteja empregando.

Os tipos de refatoração de aspectos que propõe Hannemann [29] são:

- **Refatoração orientada a objetos considerando aspectos:** trata-se de uma refatoração sobre estruturas orientadas a objetos tradicionais que são estendidas assegurando que sejam respeitadas as estruturas do paradigma orientado a aspectos.

- **Refatoração para estruturas orientadas a aspectos:** consiste em realizar a refatoração como no ponto anterior, mas agregando explicitamente estruturas de programação orientada a aspectos que puderam existir no código.

- **Refatoração de incumbências transversais:** realiza a refatoração mediante a transformação em aspectos das incumbências transversais que puderam não estar modularizadas no código.

Considerações Finais

Pela mineração de aspectos a nível de código, entre os inconvenientes mencionados por Mens et al. [28] destacam-se por cima de todos a inadequação das técnicas atuais, o que deixa um desafio importante pela pesquisa de novas técnicas e o trabalho interdisciplinar entre a mineração de dados e o desenvolvimento de sistemas orientados a aspectos.

Com relação a este último, é importante reconhecer que não são todas as incumbências transversais que são susceptíveis de serem encapsuladas em aspectos, o que deve levar necessariamente às técnicas a trabalhar somente com aqueles aspectos que podem-se encapsular. Igualmente, é importante distinguir-se o que se está realizando é realmente mineração de aspectos ou se é de mineração de *joinpoints*.

Adicionalmente, é preciso obter resultados da aplicação destas técnicas em desenvolvimentos complexos, porque os estudos atualmente disponíveis não possuem a complexidade suficiente como para poder emitir conclusões categóricas.

A falta de ferramentas e sua integração com IDEs é outro fator que tem que ser melhorado para poder fazer um uso adequado das diferentes técnicas de mineração de aspectos. Também destacam-se outros inconvenientes nas técnicas disponíveis [28], como o

fato de serem técnicas de propósito geral, fazer demasiadas suposições, serem demasiado otimistas, falta de emprego de informação semântica, falta de precisão, representação inadequada de resultados, etc.

Por outro lado, neste longo caminho por recorrer tem numerosas questões que podem-se considerar:

- trabalhar com um catálogo de incumbências transversais conhecidas;
- combinar técnicas em forma complementaria para obter melhores resultados;
- distinguir que incumbências podem ser detectadas melhor com que técnica;
- avançar sobre a mineração de *joinpoints*, tratando de obter as definições dos *pointcuts* e separar os *advices*.

Por último, é importante destacar que a maioria das técnicas do desenvolvimento orientado a aspectos em geral centram-se somente no enfoque assimétrico. Isso significa que põem a sua ênfase nas incumbências transversais que são independentes do domínio do problema, também chamadas incumbências ímpares. Quase nenhuma destas técnicas de mineração de aspectos aponta à detecção de incumbências pares, desde um enfoque simétrico. No entanto, isso não quer dizer que não seja possível aplicar os princípios que dão base a essas técnicas, o que também deixa um amplo caminho por recorrer neste sentido.

Em relação à mineração de aspectos nas etapas iniciais, o caminho por recorrer é ainda muito maior, devido à menor madures da disciplina a nível de aplicação das técnicas de mineração de dados nas especificações de requisitos e ainda muito mais na adaptação destas técnicas que atualmente utilizam-se da modelagem de negócio pela busca de aspectos nesta etapa tão inicial do ciclo de vida do desenvolvimento de software.

Referências

[1] A. Sampaio, N. Loughran, A. Rashid, and P. Rayson, “Mining Aspects in Requirements,” *Asp. Requir. Eng. Archit. Des. Work. (held with AOSD 2005), Chicago, Illinois, USA*, 2005.

[2] S. Clarke and E. Baniassad, *Aspect-oriented analysis and design. The Theme approach.* Boston: Addison-Wesley, 2005.

[3] A. Rashid, P. Sawyer, A. Moreira, and J. Araújo, “Early aspects: a model for aspect-oriented requirements engineering,” *Proc. IEEE Jt. Int. Conf. Requir. Eng.*, pp. 0–3, 2002.

[4] L. Bass, M. Klein, and L. Northrop, “Identifying Aspects Using Architectural Reasoning,” *Early Asp. Asp. Requir. Eng. Archit. Des.*, pp. 50–56, 2004.

[5] A. Rago, E. S. Abait, C. Marcos, and A. Díaz-Pace, "Early aspect identification from use cases using NLP and WSD techniques," *15th Work. Early Asp. - EA '09*, pp. 19–26, 2009.
[6] A. Rago and C. Marcos, "Uncovering Quality-attribute Concerns in Use-case Speci cations via Early Aspect Mining."
[7] A. Sampaio, P. Rayson, A. Rashid, and R. Chitchyan, "EA-Miner : a Tool for Automating Aspect-Oriented Requirements Identification," *Proc. 20th IEEE/ACM Int. Conf. Autom. Softw. Eng.*, pp. 352–355, 2005.
[8] A. Sardinha, R. Chitchyan, J. Araújo, A. Moreira, and A. Rashid, "Conflict Identification with EA-Analyzer," in *Aspect-Oriented Requirements Engineering*, 2013, pp. 209–224.
[9] A. Kellens and K. Mens, "A survey of aspect mining tools and techniques," *Proj. IWT*, vol. 40116, 2005.
[10] A. Jalali, "Aspect Mining in Business Process Management," *Lect. Notes Bus. Inf. Process.*, vol. 194, pp. 246–260, 2014.
[11] C. Di Francescomarino and P. Tonella, "Crosscutting concern mining in business processes," *Software, IET*, vol. 5, no. 6, pp. 552–562, 2011.
[12] G. Cojocar, "Aspect mining. past, present, future," vol. LVII, no. 4, pp. 85–96, 2012.
[13] P. Tonella and M. Ceccato, "Aspect mining through the formal concept analysis of execution traces."
[14] M. Marin and L. Moonen, "Identifying Aspects using Fan-In Analysis."
[15] S. Breu and J. Krinke, "Aspect mining using event traces," *Proc. 19th Int. Conf. Autom. Softw. Eng. 2004*, vol. 32, no. 9, pp. 310–315, 2004.
[16] T. Tourwe and K. Mens, "Mining aspectual views using formal concept analysis," *Source Code Anal. Manip. Fourth IEEE Int. Work.*, pp. 97–106, 2004.
[17] S. Breu and T. Zimmermann, "Mining aspects from version history," *Proc. - 21st IEEE/ACM Int. Conf. Autom. Softw. Eng. ASE 2006*, vol. 76595, pp. 221–230, 2006.
[18] G. E. P. L. Shepherd D., "Design and evaluation of an automated aspect mining tool," *Proc. Int. Conf. Softw. Eng. Res. Pract. SERP'04*, vol. 2, pp. 601–607, 2004.
[19] D. Shepherd and L. Pollock, "Interfaces , Aspects , and Views The Discoveries of a Clustering Aspect Miner and Viewer," *Interface*, pp. 1–6.
[20] G. Moldovan and G. Serban, "Aspect Mining using a Vector-Space Model Based Clustering Approach," *Proc. Link. Asp. Technol. Evol. Work.*, pp. 36–40, 2006.
[21] G. S. Moldovan, "A GRAPH ALGORITHM FOR IDENTIFICATION OF," vol. LI, no. 2, pp. 3–10, 2006.
[22] L. He and H. Bai, "Aspect Mining Using Clustering and Association Rule Method," *J. Comput. Sci.*, vol. 6, no. 2, pp. 247–251, 2006.
[23] R. R. McFadden and F. Mitropoulos, "Aspect mining using model-based clustering," *2012 Proc. IEEE Southeastcon*, no. 978, pp. 1–8, 2012.
[24] J. Huang, Y. Lu, and J. Yang, "Aspect Mining Using Link Analysis," *2010 Fifth Int. Conf. Front. Comput. Sci. Technol.*, pp. 312–317, 2010.
[25] K. Gybels and A. Kellens, "Experiences with Identifying Aspects in Smalltalk Using ' Unique Methods '," *Development*, pp. 1–6, 2005.
[26] M. Marin, L. Moonen, and A. Van Deursen, "A common framework for aspect mining based on crosscutting concern sorts," *2006 13th Work. Conf. Reverse Eng.*, pp. 29–38, 2006.

[27] A. Kellens, K. Mens, and P. Tonella, "A Survey of Automated Code-Level Aspect Mining Techniques," *Trans. Asp. Softw. Dev. IV*, vol. 4640, pp. 143–162, 2007.
[28] K. Mens, A. Kellens, and J. Krinke, "Pitfalls in aspect mining," *Proc. - Work. Conf. Reverse Eng. WCRE*, pp. 113–122, 2008.
[29] J. Hannemann, "Aspect-Oriented Refactoring : Classification and Challenges."