GLAUCO DE FIGUEIREDO CARNEIRO

SOURCEMINER: UM AMBIENTE INTEGRADO PARA VISUALIZAÇÃO MULTI-PERSPECTIVA DE SOFTWARE

Tese apresentada ao Programa Multiinstitucional de Pós-Graduação em Ciência da Computação da Universidade Federal da Bahia, Universidade Salvador e Universidade Estadual de Feira de Santana, como requisito parcial à obtenção do título de Doutor em Ciência da Computação.

Orientador: Prof. Dr. Manoel Gomes de Mendonça Neto

Salvador

2011

Carneiro, Glauco de Figueiredo.

SourceMiner: Um Ambiente Integrado para Visualização Multi-Perspectiva de Software / Glauco de Figueiredo Carneiro. – Salvador, 2011.

230f. : il.

Orientador: Prof. Dr. Manoel Gomes de Mendonça Neto.

Tese (doutorado) – Universidade Federal da Bahia, Instituto de Matemática, Doutorado Multiinstitucional em Ciência da Computação, 2011.

Referências bibliográficas.

1. Visualização de Software. 2. Compreensão de Software.

I. Carneiro, Glauco de Figueiredo. II. Universidade Federal da Bahia, Instituto de Matemática. III. Título.

CDU – 004.41

GLAUCO DE FIGUEIREDO CARNEIRO

SOURCEMINER: UM AMBIENTE INTEGRADO PARA VISUALIZAÇÃO MULTI-PERSPECTIVA DE SOFTWARE

Esta tese foi julgada adequada à obtenção do título de Doutor em Ciência da Computação e aprovada em sua forma final pelo Programa Multiinstitucional de Pós-Graduação em Ciência da Computação da UFBA-UNIFACS-UEFS.

Salvador, 17 de maio de 2011.

_____

Prof. Manoel Gomes de Mendonça Neto (orientador), Ph.D.
Universidade Federal da Bahia – UFBA

_____

Prof. José Carlos Maldonado, D.Eng.
Universidade de São Paulo – USP

_____

Prof. Guilherme Horta Travassos, D.Sc.
Universidade Federal do Rio de Janeiro – UFRJ

_____

Prof. Joberto Sérgio Barbosa Martins, Docteur en Informatique
Universidade Salvador – UNIFACS

_____

Prof[a]. Christina von Flach Garcia Chavez, D.Sc.
Universidade Federal da Bahia – UFBA

RESUMO

Atividades de compreensão têm papel importante em engenharia de software. A leitura e a busca de informações no código fonte não são atividades triviais e requerem esforço significativo em sistemas de médio e grande porte. Atualmente, apesar da maioria dos ambientes de desenvolvimento de software (ADS) oferecer apoio às atividades de compreensão, eles ainda não adotaram plenamente técnicas e recursos de visualização para tal finalidade. Esta é uma limitação significativa dado que o ser humano tem maior capacidade para obter informação através da visão do que todos os outros sentidos combinados. Além disso, as próprias técnicas de visualização de software ainda não adotaram recursos de interação e coordenação já consolidados na área de visualização de informação.

Esta tese propõe um ambiente interativo baseado em múltiplas visões chamado SourceMiner, desenvolvido como um plug-in da ADS Eclipse, para apoiar as atividades de compreensão de software. O SourceMiner traz novos recursos e técnicas para a área de visualização de software. O principal destes é o uso de múltiplas visões, já adotadas em visualização de informação e compatibilizadas para visualização de software através do conceito de perspectivas. O ambiente também utiliza mecanismos de coordenação entre as visões, zoom semântico e filtros interativos. O SourceMiner é um ambiente expansível no qual é possível incluir novas visões. Além disso, ele possui recursos de monitoramento de atividades a partir dos quais podem ser realizados estudos para a análise do perfil de uso do ADS e das funcionalidades oferecidas pelo próprio plug-in. A utilização integrada destes recursos é uma contribuição nova para a área de visualização de software.

Foram realizados seis estudos experimentais para avaliar o uso do SourceMiner em atividades de compreensão de software. Os resultados indicam que o SourceMiner pode ser utilizado para apoiar a construção de modelos mentais que se adéquam às necessidades de compreensão daqueles que lidam tanto com atividades de desenvolvimento como de manutenção de software.

Palavras chave: Visualização de Software, Compreensão de Software, Múltiplas Visões, Ambiente de Desenvolvimento de Software.

ABSTRACT

Software comprehension activities play an important role in software engineering. Reading and searching the source code is not a trivial task and requires significant effort, especially in medium to large projects. In spite of the fact that most integrated development environments provide support to software comprehension, they have not yet fully embraced the use of visualization techniques. This is a shortcoming due to the fact that vision is the dominant sense of human beings. Humans derive most of the information about the world around them from it. Furthermore, software visualization techniques themselves have not yet adopted many of the interaction and coordination resources commonly used in the information visualization field.

This dissertation proposes an interactive multiple view software comprehension environment called SourceMiner. Implemented as an Eclipse plug-in, SourceMiner adopts well established techniques and resources from the field of information visualization. The most important of them is the use of multiple views, adapted to software visualization through the concept of perspectives. The environment provides coordination among views, semantic zooming and interactive filtering. SourceMiner is an expandable environment in which new views can be included. Moreover, it monitors and registers the activities performed by the users for the analysis of the environment – plug-in and IDE – usage. The integrated use of these resources is a novelty in the software visualization domain.

To evaluate the claims of this thesis, we conducted six studies to analyze to which extent SourceMiner supports software comprehension activities. The results show that SourceMiner can support developers and maintainers in effectively building mental models to support software comprehension activities.

Keywords: Software Visualization, Software Comprehension, Multiple Views, Integrated Development Environment.

A minha Mãe Lucília, minha esposa Cristiane, meus filhos Henrique e Gustavo pelo apoio, companheirismo, alegria e entusiasmo proporcionados a cada dia da vida.

AGRADECIMENTOS

O sentido da vida está na nossa capacidade dialética. Conhecendo cada vez mais nossas limitações podemos sempre buscar caminhos para superá-las sem necessariamente subestimá-las. Este processo é mais efetivo quando compartilhado, discutido e redefinido com pessoas visionárias que temos a satisfação de encontrar ao longo da vida. Sem o apoio, influência e dinamismo destas pessoas, este trabalho agora apresentado não seria realidade.

Manoel Mendonça é uma destas pessoas. Com sua percepção singular das oportunidades de pesquisa, soube mostrar os caminhos possíveis a serem percorridos. Abriu portas. Proporcionou oportunidades de discussão nas diversas etapas deste trabalho com pesquisadores em reuniões e apresentações em eventos e em institutos de pesquisa renomados, tanto no Brasil como também na Alemanha, Inglaterra, Itália, Suiça, Espanha e Canadá. Orientou todas as etapas. Acompanhou e proporcionou a autonomia adequada para a realização deste trabalho. Aqui deixo o registro de agradecimento pelo inestimável apoio e confiança.

Apresento uma lista extensa, ainda que não exaustiva, de pessoas motivadas pelo interesse de pesquisa em Engenharia de Software. Todos acreditaram neste trabalho, participaram desta caminhada comigo, apoiaram a descoberta não somente das respostas, mas principalmente dos caminhos para alcançá-las: Angelo Orrico (UNIFACS), Eduardo Spinola (UNIFACS), Fábio Spinola (UNIFACS), Rodrigo Magnavita (UNIFACS), Edenilton (UFBA), Methanias (UFBA e UFS), José Pina (UFBA), Arleson Nunes (UFBA), Claudio Sant´Anna (UFBA), Christina von Flach (UFBA), Alessandro Garcia (Lancaster University e PUC-Rio), Eduardo Figueiredo (Lancaster University e UFMG), Carlos Fábio (UNIFACS), Paulo Roberto (UFBA), Felipe Berbert (UFBA), Diego Arize (UFBA), José Maria David (UNIFACS e UFJF), Leandra Mara (PUC-Rio), Marcos Silva (PUC-Rio), Renato Novais (UFBA e IFBA), Serge Rehem (SERPRO), Cleverson Sacramento (SERPRO), Thiago Mariano (SERPRO), João Marcelo (UNIFACS e IFBA) e Camila Nunes (PUC-Rio).

Agradeço ainda a todos aqueles que compartilharam comigo momentos de dedicação e, sobretudo, alegria ao longo destes anos, e aos membros da banca pela atenção dispensada na leitura e comentários para a melhoria deste trabalho.

O que é escrito sem esforço na maioria das vezes é lido sem prazer.

Samuel Jonhson

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

GLOSSÁRIO DE TERMOS

Modelo Mental	é uma representação interna que o ser humano elabora a partir de uma determinada situação, contexto ou entidade a ele apresentado.
Metáfora Visual	é um paradigma utilizado para apresentar visualmente uma situação, contexto ou entidade de uma forma diagramaticamente estruturada e organizada, por exemplo, um mapa, uma árvore, um grafo ou um diagrama de dispersão.
Visão	é uma representação visual de uma situação, contexto ou entidade específica instanciada a partir de uma metáfora visual, por exemplo, um mapa de uma cidade.
Perspectiva	é um conjunto de visões coordenadas que representam um grupo de propriedades específicas de uma situação, contexto ou entidade, geralmente utilizando diversas metáforas visuais.
Cenário Visual	é o conjunto de visões configurado em um dado instante, para execução de uma tarefa específica de compreensão de uma situação, contexto ou entidade.

LISTA DE ABREVIATURAS E SIGLAS

ACO	Relacionamentos de Acoplamento
ADS	Ambiente de Desenvolvimento de Software
AIMV	Ambiente Integrado baseado em Múltiplas Visões
AST	*Abstract Sintax Tree* ou Árvore Sintática Abstrata
CNA	Camada Núcleo do Ambiente
CVR	Camada de Visualização e Renderização
DC	*Divergent Change*
FE	*Feature Envy*
GC	*God Class*
HH	Hierarquia de Herança
InfoVis	*Information Visualization* ou Visualização de Informação
JDT	*Java Development Tool*
JPA	*Java Persistence API*
JSF	*JavaServer Faces*
JSR	*Java Specification Recommendation*
MVC	*Model View Controller*
p	Precisão
PA	Participante do Estudo
PCM	Estrutura Pacote, Classe, Método
PP	Pergunta de Pesquisa
r	Revocação
SoftVis	*Software Visualization* ou Visualização de Software
XML	*Extensible Markup Language*

Capítulo

1

Este capítulo aborda o cenário atual do tema de pesquisa em visualização de software. Em seguida é apresentada a solução proposta, o trabalho desenvolvido, a estratégia adotada e, por último, os resultados e contribuições desta tese de doutorado.

1 INTRODUÇÃO

Diversos trabalhos têm destacado a importância da visualização na análise de dados interativa e na exploração de informação (Wang, Woodruff e Kuchinsky, 2000; Becks e Seeling, 2004; Roberts, 2000; Roberts, 2007; Shneiderman e Plaisant, 2010; Spence, 2007). O ser humano tem a habilidade de identificar padrões visuais e esta habilidade pode ser explorada para apoiar atividades de compreensão de software (Roman e Cox, 1992). A visualização de software, também conhecida na comunidade como *SoftVis*, utiliza-se de símbolos perceptíveis visualmente para representar diversas propriedades dos sistemas de software com o objetivo de revelar padrões e comportamentos que de outra forma poderiam permanecer ocultos (Petre, 2010).

O ser humano tem maior capacidade para obter informação através da visão do que todos os outros sentidos combinados (Ware, 2004). Por esta razão, a concepção e uso de recursos visuais é muito importante para a engenharia de software. Esta atividade deve levar em consideração três questões importantes. A primeira é que o software é eminentemente complexo, o que aumenta a dificuldade de muitas das atividades de compreensão de software. A segunda, descrita pela segunda lei de Lehman (Lehman e Belady, 1985), é que o software evolui à medida que passa por modificações ao longo do tempo. A terceira é que o software é intangível, não tendo, portanto, forma física (Ball e Eick, 1996). Esta combinação de

complexidade, constante evolução e falta de presença física dificulta o desenvolvimento de mecanismos adequados de representação do software.

Várias questões devem ser consideradas no desenvolvimento de um ambiente de representação visual de software. A primeira delas é que atividades de engenharia de software, tais como a identificação de anomalias de modularidade (também conhecida como *code smells*) (Fowler, 1999), geralmente requerem a análise do software através de múltiplas perspectivas (Carneiro, Silva, *et al.*, 2010). Desta forma, para que os ambientes de visualização de software sejam efetivos, eles devem proporcionar perspectivas complementares. Mais que isso, propriedades complexas podem requerer múltiplas visões em casos em que uma única representação visual não seja suficiente para representá-las.

Outra questão importante é que o uso de múltiplas visões pode ser dificultado se as visões não estiverem coordenadas entre si e com o ambiente no qual estão sendo utilizadas. Elementos visuais de diferentes visões devem ser vinculados entre si quando representarem a mesma entidade de software. A seleção ou mudança de um elemento em uma visão deve ser refletida em todas as outras. A navegação de elementos visuais entre visões diferentes deve ser intuitiva e de fácil uso. Além disso, as ações envolvendo estes elementos visuais devem ter consistência através das visões. Embora, a coordenação entre visões seja um requisito para o uso de múltiplas visões em visualização de informação (Wang, Woodruff e Kuchinsky, 2000), e já seja utilizado nos ambientes de desenvolvimento de software atuais (Eclipse Foundation, 2011), este ainda é um conceito pouco explorado em ambientes de visualização de software (Storey, 2006).

A terceira questão é que visões são baseadas em metáforas visuais que devem ser compatíveis com os dados a serem representados e com as atividades a serem realizadas. Grafos, por exemplo, são efetivos para a visualização de dados relacionais, mas apresentam problemas de escalabilidade em relação ao número de entidades a serem representadas. Pesquisadores de visualização de informação e de software têm proposto diversas metáforas para a representação visual de dados (Ferreira e Levkowitz, 2003; Keim, 2002). Infelizmente, ainda não está claro para a comunidade de engenharia de software quais conjuntos de metáforas visuais são mais apropriados para quais atividades de engenharia de software. Por esta razão, um ambiente de visualização de software deve ser configurável e facilitar a inclusão de novas metáforas. Esta facilidade de extensão é justificada não somente pelo fato

de possibilitar o aumento do número de visões no ambiente, mas também por permitir investigar quais conjuntos de visões podem ser efetivamente combinados entre si.

Por último, mas não menos importante, os ambientes de visualização de software devem ser altamente interativos. Representações visuais buscam explorar as habilidades cognitivas do ser humano (Langelier, Sahraoui e Poulin, 2005). Programadores precisam interagir com o ambiente para configurar o cenário visual que mais se adéqüe às suas necessidades. Para tal finalidade são necessários recursos de filtragem, zoom, navegação e busca para refinar os dados a serem apresentados nas metáforas visuais. Estes recursos têm o objetivo de apoiar o ajuste dos cenários visuais em relação ao conteúdo que será apresentado, ao mapeamento visual e à configuração das visões utilizadas.

Ao se considerar todas as questões acima levantadas, pode se perceber que um ambiente de visualização de software deve suportar múltiplas perspectivas, possuir visões integradas entre si, ser extensível e altamente interativo. Esta tese lida com este tema.

1.1 VISUALIZAÇÃO DE SOFTWARE: CENÁRIO ATUAL E SUAS LIMITAÇÕES

Não existem ambientes de visualização de software com as características listadas na seção anterior. O estado da arte em ambientes de visualização de software tem se focado na proposta de novas metáforas visuais e visualização de informações específicas. Entretanto, geralmente, diferentes tipos de informação são necessários para executar as atividades de engenharia de software. Ambientes de visualização de software devem suportar múltiplas perspectivas integradas entre si.

Por outro lado, a comunidade tem utilizado ambientes de desenvolvimento de software (ADS) cada vez mais sofisticados. Estes ambientes provêem excelentes recursos de apoio às atividades engenharia de software. Infelizmente, apesar dos recursos proporcionados por estes ADSs, a compreensão de programas ainda persiste como uma atividade não trivial, especialmente para sistemas de software grandes e complexos (Gracanin, Matkovic e Eltoweissy, 2005; D'Ambros, Lanza e Lungu, 2009). Estes ambientes ainda têm recursos

limitados de visualização de software. Novas metáforas visuais e novos mecanismos de interação e coordenação de visões podem claramente contribuir neste contexto.

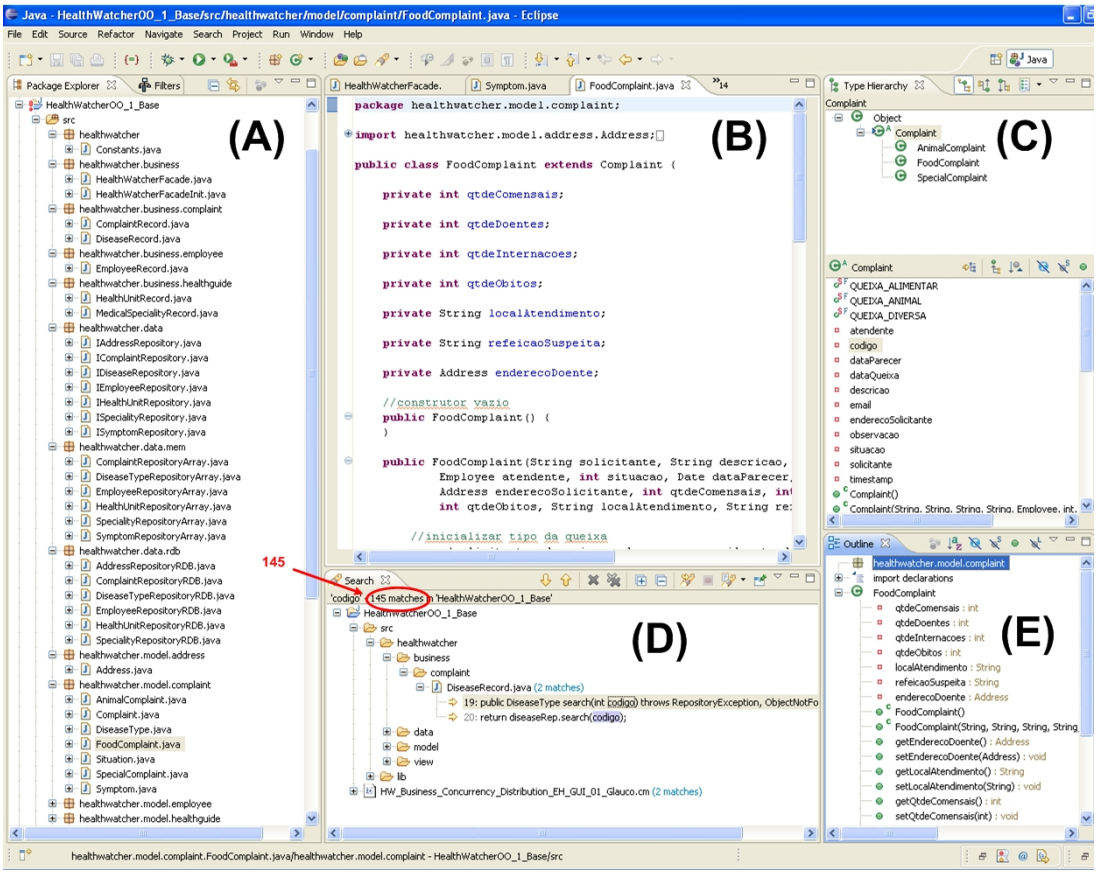

Figura 1 Um Cenário de Uso Típico de um ADS

Claramente há muito a se ganhar com a integração de recursos de visualização à ambientes de desenvolvimento de software (ADS). Para ilustrar isto, esta seção apresenta um cenário de uso do ADS Eclipse e, em seguida, discute como recursos de visualização podem tornar o seu uso mais efetivo.

A Figura 1 apresenta um cenário típico de uso do ADS Eclipse que servirá para a discussão de algumas das limitações dos ADS atuais. O cenário apresentado foi obtido durante a execução de um estudo para caracterizar a identificação de anomalias de modularidade de software (Lanza, Marinescu e Ducasse, 2005; Riel, 1996). Durante a execução da atividade, a árvore do Package Explorer (Parte A) apresenta dezenas de nós logo após alguns cliques para navegar através das classes e arquivos do projeto. A estrutura do

projeto, neste caso, não é completamente visível sem que seja necessário o uso da barra de rolagem para navegar ao longo da janela (Kersten e Murphy, 2006). Devido à facilidade de se navegar ao longo da estrutura hierárquica do projeto, o número de arquivos abertos no editor (Parte B) também pode aumentar rapidamente, dificultando a representação das classes e interfaces relevantes para uma determinada atividade (Alwis e Murphy, 2006).

A busca de referências a uma classe de um determinado projeto no Eclipse (Parte D) pode facilmente retornar centenas de itens e não há forma conveniente para se identificar somente os elementos que estão de fato relacionados à atividade em questão. Para este tipo de busca deve-se realizar uma inspeção manual para que seja encontrado o elemento de interesse (Alwis e Murphy, 2006). No caso do nosso exemplo, a inspeção manual tem como conjunto inicial as 145 ocorrências apresentadas na Parte D da Figura 1. Até mesmo a visão *Outline* (Parte E), que mostra somente a estrutura do arquivo selecionado pelo usuário, pode ser sobrecarregada por dezenas de elementos que podem não ser relevantes para a atividade em execução (Alwis e Murphy, 2006).

As limitações dos ADSs não são somente visuais. Considere novamente a informação da estrutura do código fonte apresentada pelo Package Explorer (hierarquia de pacotes, arquivos, classes, métodos e atributos). Esta informação é útil, porém limitada. O Package Explorer por si só não é suficiente para apoiar a maioria das atividades de desenvolvimento e manutenção de software. Na prática, ele deve ser usado em conjunto com outras visões. Mais que isso, o próprio Package Explorer poderia ser enriquecido com mais informação. Ele não apresenta dados relacionados a métricas importantes como tamanho do software, por exemplo. Na realidade, a maioria dos ADS não oferece visões que apresentem esta propriedade, apesar dela ser importante e poder ser facilmente representada em qualquer metáfora visual.

Os ADS atuais também precisam explorar recursos de interação de forma mais efetiva. Como exemplo, considere uma interação para a transformação de dados. A maioria das ferramentas de visualização de informação atuais, como o SpotFire (Tibco, 2011) e SmartMoney (SmartMoney, 2011), possibilitam a filtragem dinâmica de dados. Com ela é possível remover, de forma interativa do cenário visual, conjuntos de dados que não estejam de acordo com um critério de filtragem escolhido. Atualmente, os ADS não oferecem condições efetivas para este tipo de operação.

Desta maneira, observa-se que os ADS podem ser enriquecidos de diversas formas em termos de visões, interação e informação. Muitos deles inclusive já fornecem bibliotecas e APIs para sua melhoria. O Eclipse, por exemplo, disponibiliza uma ampla infra-estrutura para o desenvolvimento de novas funcionalidades (Clayberg e Rubel, 2008). A questão é qual abordagem adotar para oferecer um apoio efetivo à visualização de software em ADS modernos. Uma abordagem possível é a expansão do ADS para apoiar funcionalidades fundamentais já utilizadas em visualização de informação (Wang, Woodruff e Kuchinsky, 2000; Card, Mackinlay e Shneiderman, 1999; Spence, 2007). Sobre este substrato, pode-se iterativamente enriquecer o ADS com vários recursos integrados de visualização de software, e avaliá-los quanto à efetividade em diferentes atividades de engenharia de software.

1.2 VISUALIZAÇÃO DE SOFTWARE: SOLUÇÃO PROPOSTA

Esta tese de doutorado trata exatamente do projeto e construção de um substrato de software para enriquecer um ADS com recursos de visualização de software, e da instanciação deste substrato em um ambiente interativo baseado em múltiplas perspectivas. As visões disponibilizadas neste ambiente estão integradas entre si e também com o ADS. Além disso, elas podem ser configuradas dinamicamente para a realização de diversas atividades de engenharia de software.

A Figura 2 apresenta um cenário de uso do ambiente de software desenvolvido. As setas indicam como uma classe específica de uma aplicação chamada HealthWatcher (Greenwood, Bartolomei, *et al.*, 2007) é representada visualmente através de múltiplas visões. O editor do Eclipse (Parte B) apresenta o código fonte das classes selecionadas e o seu Package Explorer (Parte A) apresenta a estrutura de pacotes, classes, métodos e atributos através de uma visão estrutural tradicional. Estas são visões nativas do ADS.

As partes D, E e F mostram três visões do ambiente. Da mesma forma que o Package Explorer, a visão apresentada na Parte D representa a perspectiva pacote-classe-

método da aplicação analisada. Entretanto, a visão da Parte D utiliza um *mapa em árvores* (também conhecido como *treemap*) (Shneiderman, 1992), uma metáfora para visualização de hierarquias utilizando os retângulos aninhados. Ao contrário do Package Explorer, esta visão provê uma representação global da estrutura da aplicação. Nela não é necessário utilizar a barra de rolagem, pois todos os elementos estão sempre representados na cena visual.

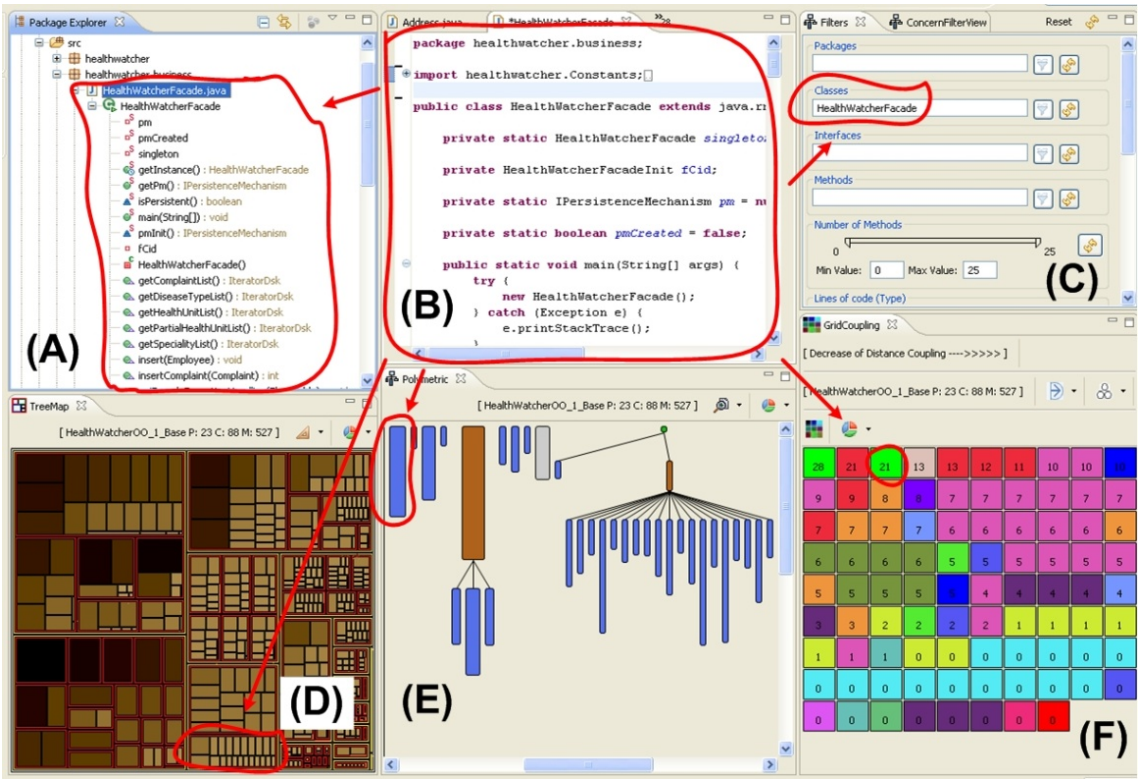

Figura 2 Um Cenário de Uso do Eclipse com o SourceMiner

A Parte E proporciona uma visão global da estrutura de herança do código e a Parte F dos módulos mais acoplados da aplicação. Elas, e outras, serão discutidas em detalhes no Capítulo 4 desta tese. O ponto chave aqui é entender que as visões não tradicionais apresentadas nas Partes D, E e F representam o software através de diferentes perspectivas. A Parte D proporciona uma visão hierárquica global da estrutura do código, a Parte E representa herança e a Parte F a força de acoplamento dos módulos. Juntas elas têm o objetivo de contribuir para melhorar a compreensão do software pelo usuário. Cabe a este mesmo usuário escolher a forma e ordem com que estas visões serão utilizadas de forma que elas se ajustem às suas necessidades.

O ambiente oferece ainda recursos de interação. As visões D, E e F são diretamente afetadas pela visão de filtragem apresentada na Parte C. Esta é a visão utilizada para configurar critérios que determinarão quais entidades serão apresentadas em D, E e F. No exemplo, o usuário digitou a expressão *HealthWatcherFacade* como opção para destacar as classes que possuam este nome em todas as visões disponíveis do ambiente. Este é um exemplo da interação para a transformação de dados através de um recurso de filtragem. Como ressaltado anteriormente, apesar de muito útil, este tipo de recurso ainda não está disponível em ADS modernos.

Todas as visões do ambiente e os seus recursos serão explicados em detalhes nos capítulos seguintes. O objetivo nesta subseção foi destacar que o ambiente desenvolvido representa o software através de diferentes perspectivas, melhorando as visões e recursos nativos do ADS.

1.3 O TRABALHO DESENVOLVIDO

Os ambientes de desenvolvimento de software (ADS) atuais como o Eclipse são altamente extensíveis. Neles há a possibilidade de organização dos dados de um projeto de software em estruturas apropriadas para sua exploração, do mapeamento destes dados – atributos do software – para atributos visuais – tais como formas, posições e cores – e, por último, da representação destes em metáforas visuais adequadas à interpretação humana.

Esta tese propõe, desenvolve e avalia um modelo para um ambiente extensível de visualização de software baseado em múltiplas visões coordenadas entre si para a execução de atividades de compreensão de software. O ambiente concebido neste trabalho, chamado de ***SourceMiner***, foi implementado como um plug-in do Eclipse para representar sistemas de software em Java através de visões coordenadas entre si. O SourceMiner proporciona várias formas de interação com as visões. Dentre elas, filtros que podem ser configurados para a representação visual de acordo com o critério adotado. Recursos de zoom semântico e geométrico que podem ser usados para melhor ajustar as representações visuais à tela de

acordo com a atividade a ser realizada. E, recursos de navegação da representação visual dos elementos para seu respectivo trecho no editor do código fonte.

O foco do trabalho é a visualização estática. O ambiente não analisa a execução do software, mas a sua estrutura estática e o relacionamento entre as entidades que o compõem (Caserta e Zendra, 2010). Os elementos básicos desta estrutura são pacotes, classes, métodos, e os seus respectivos atributos. As lições aprendidas na concepção deste ambiente podem ser aplicadas a outros tipos de visualização de software tais como na representação dinâmica do software e da sua evolução (Diehl, 2007).

1.4 A ABORDAGEM ADOTADA

Esta tese de doutorado adotou uma abordagem iterativa e incremental nas etapas de concepção, desenvolvimento e avaliação do SourceMiner. O modelo inicialmente concebido foi ampliado e aprofundado a cada nova etapa, permitindo sua evolução incremental. A Figura 3 apresenta uma visão geral destas etapas seguindo três eixos principais que serviram de referência para este trabalho. Os eixos I, II e III são representados na parte interna, intermediária e externa da Figura 3 respectivamente.

O primeiro eixo (**eixo I**) corresponde à infra-estrutura de visualização em relação ao ambiente no qual o SourceMiner é utilizado. O segundo eixo (**eixo II**) corresponde à evolução das funcionalidades do SourceMiner, sendo composta de treze etapas. O terceiro eixo (**eixo III**) corresponde às publicações ao longo do trabalho para avaliar e discutir o modelo proposto em diversos cenários e de diferentes formas.

Três etapas foram percorridas no Eixo I (infra-estrutura e ambiente). Na primeira, o SourceMiner era uma aplicação Java não integrada ao ambiente de desenvolvimento de software. Os estudos mostraram que o uso do SourceMiner de forma integrada ao ADS traria maior efetividade nas atividades de compreensão. Isto resultou no planejamento e migração para a segunda etapa. O SourceMiner passou a ser integrado ao ADS, no formato de um plug-in, e a utilizar de diversos recursos oferecidos pelo Eclipse. Na terceira etapa, o SourceMiner

passa a ser instalado pelo próprio Eclipse através do recurso de instalação automática oferecido pelo ADS.

Muitas etapas foram percorridas nos Eixos II e III (funcionalidades e publicações). Na **etapa A**, o SourceMiner utilizava o JavaCC (Viswanadha e Sankar, 2011), um gerador de *parser* para a linguagem Java, com o objetivo de obter informações do código fonte da aplicação a ser analisada. Na **etapa B**, tem-se a primeira versão desenvolvida do mapa em árvores (Shneiderman, 1992) para a representação visual da estrutura pacote-classe-método da aplicação. Na seqüência, na **etapa C** foram adicionados recursos de filtragem interativa, juntamente com zoom geométrico e semântico para uso e configuração da representação visual do mapa em árvores. Neste período foram escritos três trabalhos com o objetivo de analisar a viabilidade do uso desta versão do SourceMiner e também para discutir o modelo proposto para o ambiente de visualização: *Empirically Evaluating the Usefulness of Software Visualization Techniques in Program Comprehension Activities* (**P1**) (Carneiro, Araujo e Mendonça, 2007), *Using Visual Metaphors to Enhance Software Comprehension Activities* (**P2**, Simpósio de Doutorado do ESEM, e **P3**, Escola de Verão de Salerno/Itália) (Carneiro, 2007;Carneiro, 2007a).

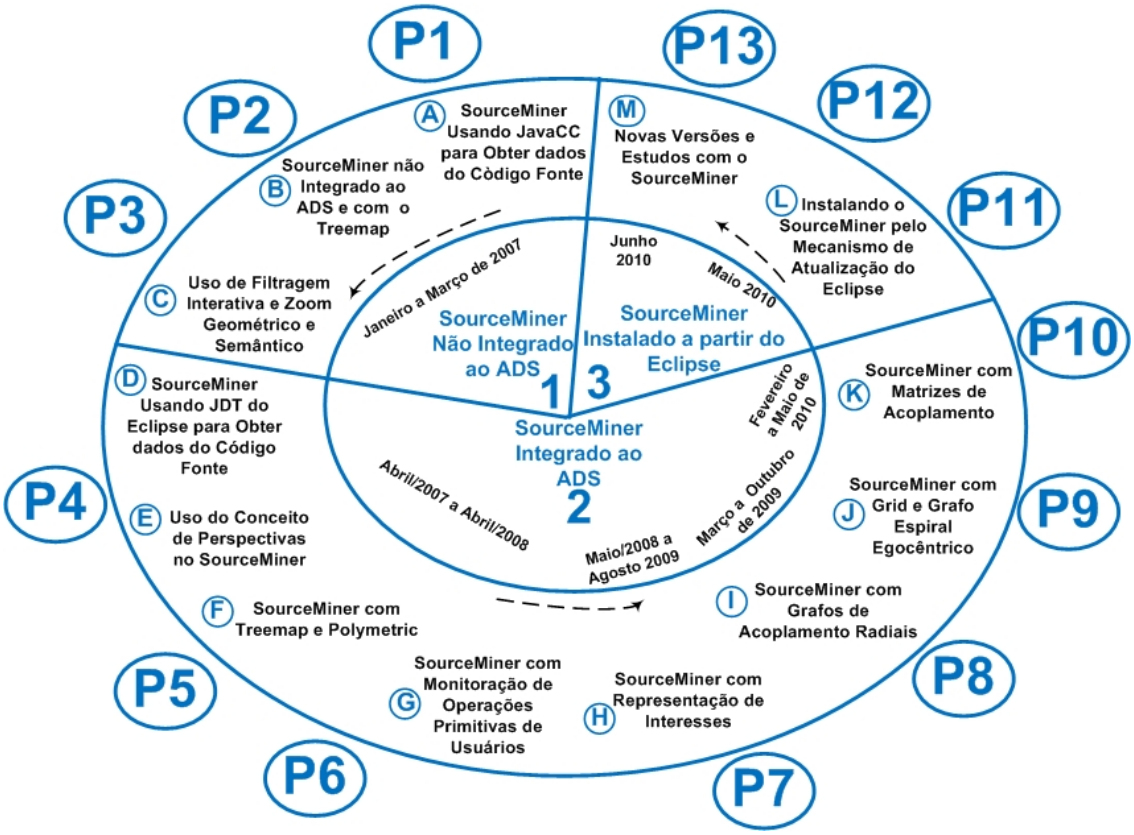

Figura 3 Concepção e Avaliação do SourceMiner

Os estudos iniciais mostraram que o uso do JavaCC não era efetivo em função do tamanho final do SourceMiner e da necessidade das adaptações recorrentes para se garantir compatibilidade com a versão da linguagem Java da aplicação a ser analisada. A partir da **etapa D**, o SourceMiner passa a ser integrado ao Eclipse. Agora os dados do código fonte são obtidos diretamente da árvore sintática abstrata disponibilizada pelo *Java Development Tool* (JDT) (Clayberg e Rubel, 2009) do Eclipse. Os recursos oferecidos pelo JDT facilitaram a obtenção de informações do código fonte. Na **etapa E** ficou evidente que uma única visão, e o conjunto de propriedades por ela representado, não seriam suficientes para apoiar atividades de compreensão de software. Nela foi incluído o conceito de perspectiva no SourceMiner. A visão mapa em árvores atendia bem aos seus objetivos, entretanto outras propriedades relevantes do software não podiam ser identificadas e analisadas com esta metáfora visual. A hierarquia de herança foi a próxima propriedade representada no SourceMiner (**etapa F**) utilizando visões polimétricas como metáfora visual (Lanza e Ducasse, 2003). Na **etapa G**, foram introduzidos recursos para a monitoração de operações primitivas executadas pelo

usuário no ambiente. Através dos registros destas operações tornou-se possível a coleta de dados do ambiente em estudos experimentais. A partir deste momento foi possível analisar o perfil de uso não somente das representações visuais oferecidas pelo plug-in, mas também do ambiente de desenvolvimento de software como um todo. Neste período foram publicados os seguintes trabalhos: *The Importance of Cognitive and Usability Elements in Designing Software Visualization Tools* (Carneiro e Mendonça, 2008) (**P4**); *Evaluating the Usefulness of Software Visualization in Supporting Software Comprehension Activities* (Carneiro, Magnavita, *et al.*, 2008) (**P5**), *Combining Software Visualization Paradigms to Support Software Comprehension Activities* (Carneiro, Magnavita e Mendonça, 2008) ***(P6)*** e *An Eclipse-Based Visualization Tool for Software Comprehension* (Carneiro, Magnavita e Mendonça, 2008a) ***(P7)***, sendo que neste último trabalho, o SourceMiner foi premiado como a melhor ferramenta do XXII Simpósio Brasileiro de Engenharia de Software.

Com os estudos até então realizados, surgiram evidências de que as informações disponibilizadas pelas visões não eram suficientes para o apoio efetivo às muitas atividades de compreensão de software. Verificou-se que o conjunto de metáforas visuais oferecido pelo SourceMiner poderia ser enriquecido uniforme e simultaneamente também com informações relacionadas aos requisitos funcionais e não funcionais de uma aplicação. Nosso primeiro passo nesta direção foi estender o SourceMiner para suportar a representação visual de interesses[1](*concerns*) (Robillard e Murphy, 2007; Tarr, Ossher, *et al.*, 1999; Cacho, Sant'Anna, *et al.*, 2006; Garcia, Sant'Anna, *et al.*, 2005). Na **etapa H** foi incluída a representação de interesses para enriquecer a representação das metáforas visuais adotadas.

A próxima melhoria foi a inclusão de uma perspectiva para representar relacionamentos de acoplamento. Na **etapa I** foi incluída a metáfora visual de grafos radiais para a representação de três níveis distintos de acoplamento: acoplamento entre pacotes, entre classes e entre métodos. Foi adicionada a funcionalidade de zoom semântico para a navegação entre estes três níveis.

Na **etapa J** as visões tabular e grafo espiral egocêntrico foram incluídas. O objetivo destas duas visões é fornecer uma visão panorâmica dos módulos mais acoplados da

[1] O termo "interesse" como tradução de "concern" seguiu a tradução proposta no I Workshop Brasileiro de Desenvolvimento de Software Orientado a Aspectos (WASP 04) e disponível em http://wiki.dcc.ufba.br/WASP04/Termos

aplicação analisada. Seu foco é em representar a força dos acoplamentos entre módulos (classes e interfaces) do software.

Em função das ocorrências de oclusão características dos grafos, na **etapa K** foi incluída a metáfora de matrizes de relacionamento de acoplamento. Esta metáfora é equivalente aos grafos radiais, pois tem foco nos relacionamentos de acoplamento, mas ela escala melhor visualmente. De forma análoga aos grafos, as matrizes também apresentam os três níveis de acoplamento de pacotes, classes e métodos.

Neste período foram publicados os seguintes trabalhos: *On the Use of Software Visualization to Support Concern Modularization Analysis* (Carneiro, Sant´Anna, *et al.*, 2009) (**P8**); *An Experimental Platform to Characterize Software Comprehension Supported by Visualization* (Carneiro, Magnavita e Mendonça, 2009a) (**P9**); *Proposing a Visual Approach to Support the Characterization of Software Comprehension Activities* (Carneiro, Magnavita e Mendonça, 2009b) (**P10**).

Na **etapa L,** o SourceMiner foi modificado para instalação a partir do próprio Eclipse. Esta etapa concluiu o trabalho de desenvolvimento do SourceMiner relatado neste tese. A partir deste momento, **etapa M**, o foco do trabalho passou a ser a avaliação do SourceMiner. Foram realizados três estudos neste período. Um estudo observacional foi realizado com o objetivo de analisar o uso do SourceMiner no apoio à identificação de anomalias de modularidade. Foram realizados também dois estudos de caso na indústria. Nestes dois estudos, o SourceMiner apoiou atividades de compreensão decorrentes de situações reais e foi possível verificar a efetividade do ambiente no apoio a estas atividades. As publicações deste período foram as seguintes: *On the Design of a Multi-Perspective Visualization Environment to Enhance Software Comprehension Activities* (Carneiro, Sant´Anna e Mendonça, 2010) (**P11**); *An Eclipse-Based Multi-Perspective Environment to Visualize Software Coupling* (Carneiro, Nunes, *et al.*, 2010) (**P12**) e *Identifying Code Smells with Multiple Concern Views* (Carneiro, Silva, *et al.*, 2010) (**P13**). Sendo este ultimo premiado como o melhor artigo do XXIV Simpósio Brasileiro de Engenharia de Software (SBES).

A Figura 3 será utilizada como referência da evolução do trabalho nos capítulos desta tese, sendo que o Capítulo 3 descreve os estudos realizados para definição e evolução do ambiente (etapas A à K da Figura 3) , o Capítulo 4 descreve o modelo conceitual resultante

destes estudos, o Capítulo 5 descreve a arquitetura e implantação do SourceMiner, e o Capítulo 6 e 7 descrevem os estudos de caracterização e avaliação industrial do SourceMiner (etapa L e M da Figura 3).

1.5 RESULTADOS E CONTRIBUIÇÕES DESTE TRABALHO

As contribuições desta tese podem ser sumarizadas conforme descrito a seguir:

- Proposta de um modelo de referência para visualização de software baseado em múltiplas perspectivas;
- Proposta uma arquitetura para um ambiente de visualização de software suportando o modelo proposto;
- Concepção de duas novas metáforas para visualização de software, nominalmente: a visão tabular de força de acoplamento e o grafo espiral egocêntrico para representação de propriedades de acoplamento;
- Desenvolvimento de um ambiente interativo baseado em múltiplas visões (**AIMV**) integrado ao ADS Eclipse para apoio às atividades de compreensão de software;
- Concepção e desenvolvimento de uma solução para a representação visual de interesses;
- Concepção e desenvolvimento de um serviço de monitoramento de ações executadas no ADS Eclipse para apoiar a realização de estudos de perfil de uso do ambiente e das múltiplas visões desenvolvidas;
- Definição e aplicação de uma abordagem iterativa e incremental para o desenvolvimento e evolução de ambientes de visualização de software.

Os principais fundamentos utilizados para a concepção do SourceMiner foram obtidos das áreas de compreensão de software (Storey, Fracchia e Muller, 1999; Alwis e Murphy, 2006; Baecker e Marcus, 1989; Brooks, 1983; Mayrhauser e Vans, 1993; Ko, Myers,

et al., 2006; Murphy, Kersten e Findlater, 2006; Robillard, Coelho e Murphy, 2004) e visualização da informação (Wang, Woodruff e Kuchinsky, 2000; Card, Mackinlay e Shneiderman, 1999; Roberts, 2000; Roberts, 2007; Graham e Kennedy, 2008; Shneiderman e Plaisant, 2010; Spence, 2007; Ware, 2005; Ware, 2004). Ao longo dos capítulos desta tese, serão apresentados conceitos e lições aprendidas que contribuíram para a concepção, implementação e avaliação do SourceMiner.

1.6 ORGANIZAÇÃO DA TESE

Esta tese está organizada conforme descrito a seguir. O Capítulo 2 apresenta conceitos de Compreensão e Visualização de Software relevantes para este trabalho. O Capítulo 3 apresenta os estudos preliminares conduzidos com o objetivo de identificação e concepção das funcionalidades a serem oferecidas pelo SourceMiner. O Capítulo 4 apresenta o modelo conceitual resultante e as visões concebidas para o ambiente, incluindo a forma como estas visões podem ser combinadas e utilizadas para a execução de atividades de compreensão de software. O Capítulo 5 apresenta a arquitetura e descreve os recursos de interatividade e integração da versão do SourceMiner resultante dos estudos preliminares. O Capítulo 6 descreve um estudo observacional com o objetivo de caracterizar o uso da versão do SourceMiner descrita no capítulo anterior. O Capítulo 7 descreve dois estudos de caso com o objetivo de avaliar o uso do SourceMiner na indústria. O Capítulo 8 apresenta as conclusões, com um resumo do trabalho realizado, as contribuições ao estado da arte, as limitações e os trabalhos em andamento e previstos com o SourceMiner.

Capítulo

2

Este capítulo contextualiza conceitos de compreensão e visualização de software. Inicialmente são apresentadas teorias cognitivas selecionadas da literatura sobre compreensão de software. Em seguida é discutida a importância da visualização interativa, das múltiplas visões e das múltiplas perspectivas para a visualização exploratória. O capítulo, enfim, discute o tema de visualização de software, as diversas ferramentas já propostas e como elas têm sido validadas experimentalmente.

2 COMPREENSÃO E VISUALIZAÇÃO DE SOFTWARE

As atividades de engenharia de software são por natureza complexas, e de escopo variado passando pela especificação, projeto, desenvolvimento, teste e manutenção. Cada uma destas etapas requer diferentes demandas cognitivas tais como buscar, identificar, compreender, representar, analisar, e propor soluções. A complexidade das atividades é resultante da natureza dos problemas tratados, da sua diversidade, da natureza dos artefatos produzidos, e dos ambientes sociais e organizacionais nos quais são conduzidos.

A visualização é uma proposta viável para lidar com atividades como a busca, identificação, e análise necessárias para se compreender o software. Neste contexto, este capítulo apresenta os conceitos de compreensão e visualização de software que são considerados relevantes para o trabalho proposto nesta tese.

2.1 COMPREENSÃO DE SOFTWARE

A compreensão de software consiste na obtenção de conhecimento das funcionalidades, estrutura e comportamento de um sistema de software (Mayrhauser e Vans, 1995). É um requisito fundamental na grande maioria das atividades de desenvolvimento e de manutenção de software. Na fase de desenvolvimento, técnicas de compreensão de software devem ser usadas para assegurar que o sistema em desenvolvimento está de acordo com o planejado no projeto. Na fase de manutenção, a compreensão de software é usada para apoiar, dentre outras, as atividades de evolução para melhorar ou corrigir o sistema, na engenharia reversa para extrair informações do software e na reengenharia para modificar funcionalidades existentes. A compreensão de software também tem suas aplicações na área de reuso (Fayad, Schmidt e Johnson, 1999; Szyperski, 2002). A literatura relata que aproximadamente 60% do esforço nas atividades de engenharia de software estão diretamente relacionadas à compreensão do sistema em questão (Corbi, 1989; Pigoski, 1996).

A contextualização de compreensão de software neste trabalho tem o objetivo de enfatizar que compreender o software não é uma atividade trivial pelos três motivos apresentados a seguir:

- **Software é eminentemente complexo**. O tamanho e a complexidade dificultam a compreensão do software, tanto parte como o seu todo (Caserta e Zendra, 2010). Existem diversas técnicas de engenharia reversa para apoio à compreensão de software (Briand, 2006). Infelizmente, nenhuma destas técnicas isoladamente é efetiva para a compreensão do software como um todo em função do seu tamanho e complexidade.
- **Software é intangível**. E não possui forma física (Ball e Eick, 1996; Caserta e Zendra, 2010). Além disto, tem-se que o ser humano obtém muito mais informação por intermédio da visão do que todos os outros sentidos juntos e combinados (Ware, 2004). Como resultado, tem-se que a compreensão de software é por si só dificultada pelo fato do mesmo ser intangível.
- **Software evolui**. Sua documentação de alto nível pode se tornar rapidamente desatualizada. Mais importante que isto, a segunda lei de Lehman (Lehman e

Belady, 1985) afirma que com a evolução do software, também aumenta sua complexidade. O resultado disto é que mais recursos deverão ser disponibilizados para preservar e simplificar sua estrutura. Tem-se também que mais dados deverão ser compreendidos, incluindo agora a possibilidade de análise de várias versões.

Técnicas e ferramentas têm sido propostas para apoiar as diversas atividades do ciclo de vida do software. Várias destas atividades não são completamente automatizáveis e são suportadas por técnicas e ferramentas que apóiam a sua execução por seres humanos (Storey e Muller, 1995; Souza, Quirk, *et al.*, 2007; In e Roy, 2001). Este apoio é chamado na literatura de "suporte cognitivo". O estudo e o desenvolvimento de ferramental para o suporte cognitivo em engenharia de software é um importante tema de pesquisa (Storey, 2006; Hundhausen, 1998; Hundhausem, Douglas e Stasko, 2002).

A literatura já dispõe de trabalhos que analisam como desenvolvedores executam atividades de compreensão de software. O resultado foi o desenvolvimento de várias teorias cognitivas para descrever o processo de compreensão de programas. Estas teorias podem ser organizadas em três categorias. Teorias *bottom-up* (Shneiderman e Mayer, 1979) propõem que a compreensão de programas inicia-se na formação de abstrações de baixo nível através da leitura do código fonte. Teorias *top-down* (Soloway e Ehrlich, 1989; Brooks, 1983) descrevem o processo de compreensão de software como o mapeamento de conhecimento previamente adquirido a respeito de domínio do programa em estruturas encontradas no código fonte. A terceira categoria considera o processo de compreensão como uma combinação dos modelos *bottom-up* e *top-down* (Mayrhauser e Vans, 1993). Estas teorias têm o objetivo de descrever como se constrói e adquire representações mentais dos programas, mas não oferecem orientações de como desenvolvedores podem conduzir a exploração de sistemas de software (Alwis e Murphy, 2006).

Pesquisadores realizaram estudos com o objetivo de utilizar as teorias cognitivas na concepção de abordagens práticas e no desenvolvimento de ferramentas que suportem atividades de compreensão de software. Maryhauser e Vans (1993) analisaram um conjunto de teorias cognitivas para identificar uma lista de atividades típicas de compreensão e as informações necessárias para executá-las. Storey e colegas (1999) examinaram as teorias cognitivas e identificaram quatorze elementos a serem considerados na concepção de ferramentas. As orientações fornecidas por estes trabalhos são úteis para a identificação de

oportunidades de melhorias nas ferramentas que proporcionam suporte cognitivo aos desenvolvedores de software.

Outra linha de trabalho corresponde à caracterização detalhada da forma como os desenvolvedores executam as atividades de programação. Por exemplo, Ko, Myers e colegas (2006) analisaram como desenvolvedores utilizaram a ADS Eclipse para analisar o código fonte e propor alterações na aplicação analisada. Murphy, Kersten e Findlater (2006) analisaram padrões de uso das funcionalidades oferecidas pela ADS Eclipse. Robillard, Coelho e Murphy (2004) investigaram as características que diferenciam desenvolvedores bem e mal sucedidos na execução de atividades de manutenção. Apesar de terem sido restritos às atividades solicitadas aos participantes, todos estes estudos identificam a existência de padrões de uso do ADS em atividades de compreensão de software.

Compreensão do software a partir do código fonte

A compreensão de software implica no estudo de representações do software para obter o conhecimento necessário para a execução de uma atividade de engenharia de software. No cenário ideal, esta representação está na forma de uma documentação de alto nível consistente com a atividade a ser realizada. Entretanto, muitas vezes este tipo de documentação não está disponível ou atualizada (Deursen, Hofmeister, *et al.*, 2004). Nesta situação, o código fonte é o principal ativo do software disponível para a obtenção de informação.

Infelizmente, a extração direta de informação do código fonte pelo ser humano não é uma atividade trivial. Ela requer um grande esforço cognitivo, especialmente se realizado manualmente. Além disso, o volume daquilo que tem que ser compreendido varia de acordo com a atividade a ser realizada. Técnicas de engenharia reversa podem extrair informações de alto nível do software, tais como os seus módulos e o relacionamento existente entre eles. Informações de baixo nível tais como aquelas disponíveis no código fonte podem servir de ponto de partida para a execução deste tipo de atividade.

Algumas atividades de engenharia de software podem ser completamente baseadas em informações de baixo nível, requisitando basicamente a análise sintática do mesmo. Um exemplo típico são as operações de refatoração atualmente implementadas e assistidas parcialmente nos ADS. Estas atividades podem ser automatizadas.

Outras atividades não são difíceis de automatizar, pois requerem uma compreensão de mais alto nível. Todavia, elas têm como ponto de partida uma informação que pode ser diretamente extraída a partir do ADS. A depuração de uma função que ordena um arranjo incorretamente é um exemplo disto. Ela se inicia com informações de baixo nível para o entendimento de uma informação de alto nível. O programador terá que compreender o algoritmo de ordenação em questão, mas poderá obter toda informação necessária para isto através da análise manual de um pequeno trecho de código fonte.

Por outro lado, outras atividades requerem informações de alto nível que só podem ser obtidas a partir da análise de um grande volume de informações de baixo nível. Considere uma atividade de reestruturação arquitetural como exemplo. Ela requer a obtenção de uma visão abrangente dos módulos do sistema, sua organização e relacionamentos, para compreensão de como a arquitetura do sistema está organizada. No lugar de se analisar diretamente um grande volume de código fonte, atualmente, a melhor alternativa para uma atividade deste tipo é o oferecimento de alguma forma de representação de mais alto nível como ponto de partida.

Pesquisadores de engenharia de software propuseram várias formas de representar esta informação de mais alto nível, já que a inspeção manual de grandes volumes de código é inviável em função da sobrecarga cognitiva para a sua execução. Diagramas que descrevem a arquitetura de um sistema, por exemplo, ficam rapidamente desatualizados à medida que o software evolui. Uma solução para este problema é a criação de ferramental para criar e manter atualizado este tipo de informação diretamente a partir do código fonte. Este tipo de abordagem de permitir o mapeamento mental entre os elementos utilizados no código e seus respectivos relacionamentos arquiteturais. A utilização de técnicas de visualização de software é uma das alternativas mais promissoras neste aspecto. Para isto, o grande número de informações de baixo nível que pode ser extraída do código fonte deve ser mapeado para abstrações visuais significativas (do ponto de vista arquitetural) e facilmente exploráveis por seres humanos. Esta tese lida com esta problemática, através da definição de um modelo de referência para visualização de software a partir do seu código fonte e da construção e avaliação do SourceMiner, um ferramental que implementa este modelo.

O restante deste capítulo apresenta conceitos e definições de visualização de informação e de software incluindo os termos usados para descrever e compreender o modelo

proposto para o SourceMiner. A maioria das definições foi obtida de trabalhos relevantes da área de compreensão de software, de visualização de informação e de visualização de software.

2.2 VISUALIZAÇÃO INTERATIVA EM PROCESSOS COGNITIVOS

Visualização é um importante meio de compreensão e é fundamental para apoiar a construção de um modelo mental ou uma imagem mental a respeito de alguma representação visual (Spence, 2007). Visualizar é uma atividade cognitiva, apoiada por representações visuais externas através das quais se constrói uma representação mental interna do cenário visual observado (Spence, 2007; Ware, 2004). No processo de visualização, dados são transformados em imagem. A imagem, por sua vez, é interpretada pelo ser humano. A interpretação de uma imagem pode conduzir à descoberta de informação a partir do que foi codificado graficamente. Isto fecha o ciclo da visualização que tem o objetivo de permitir que informação relevante seja obtida a partir de um conjunto de dados.

O nosso foco visual em um dado instante é influenciado pelas atividades que estamos executando. Portanto, nossa visão busca aquilo que nos interessa e este processo é uma composição de varreduras visuais que fazemos (Ware, 2005). Dependendo da atividade, executamos diversas buscas visuais para alcançar o nosso objetivo.

Uma visualização consiste em estruturas visuais e conjuntos de símbolos. Estruturas são embutidas em mapas e em vários outros tipos de diagramas tais como grafos ou árvores. Os símbolos podem ser de vários tipos tais como palavras, formas geométricas ou ícones. Se os símbolos são familiares, eles automaticamente excitam os conceitos correspondentes e são carregados na memória. De acordo com Mukherjea e Foley (1996), a visualização deve buscar permitir que pessoas utilizem o raciocínio perceptivo (fortemente baseado na percepção visual) ao invés do raciocínio cognitivo na detecção de padrões que revelam estruturas implícitas nos dados. Isto facilita a compreensão, pois reduz o esforço cognitivo à atividade essencial de interpretação dos padrões.

O poder da visualização está, então, na possibilidade de se ter uma estrutura conceitual complexa representada externamente através de uma exposição visual intuitiva que revele padrões de interesse do usuário. Daí o uso de ferramentas de visualização de informação no apoio ao processo cognitivo. A combinação do uso de tais ferramentas com a capacidade cognitiva do usuário torna mais efetiva a busca por informação e a identificação de padrões na informação apresentada. Para representações visuais não interativas como mapas, o movimento dos olhos é o recurso principal para a obtenção de informação. Simplesmente focamo-nos no mapa e em suas regiões com diferentes focos e partimos para a busca de padrões. Com o uso de técnicas de interação disponibilizadas pelos mais modernos sistemas de visualização de informação, há a possibilidade de aumentar a efetividade das buscas por padrões através de configurações interativas do cenário visual (Ware, 2005).

O processo de visualização de informação pode ser mapeado como parte de um processo de compreensão. O processo de compreensão parte de um conjunto de dados para atingir um conhecimento aprofundado (sabedoria) a respeito de um tema (Jacobson, 2000). A Figura 4 apresenta este processo. A visualização da informação está localizada entre os dados e a informação no processo continuado de compreensão, proporcionando os métodos e técnicas para se organizar e representar os dados para que deles seja obtida informação. Quando a informação é integrada à experiência tem-se o conhecimento. Baseando-se na experiência é possível adquirir conhecimento com os quais somos capazes de compreender os fatos que nos cercam. A sabedoria é o nível mais alto e avançado do processo de compreensão. Através da sabedoria pode-se fornecer julgamento qualificado a respeito dos dados. A sabedoria é conseqüência da habilidade de estudar e interpretar o próprio conhecimento. Mas, diferentemente do conhecimento, não pode ser transmitida ou ensinada (Mazza, 2009).

Segundo Keim (2002), os objetivos de visualização de informação dividem-se em três tipos:

- **Análise Exploratória**: não há uma hipótese a respeito dos dados; o conhecimento é descoberto através da busca interativa;
- **Análise Confirmativa**: há uma hipótese formulada; o objetivo é conhecido e o conhecimento é confirmado ou rejeitado durante a busca do objetivo.

- **Apresentação**: fatos e informações são previamente conhecidos e apresentados a outros com o auxílio de uma ferramenta de visualização.

O foco deste trabalho é no primeiro objetivo. Na subseção a seguir serão apresentadas as metáforas e técnicas para visualização exploratória.

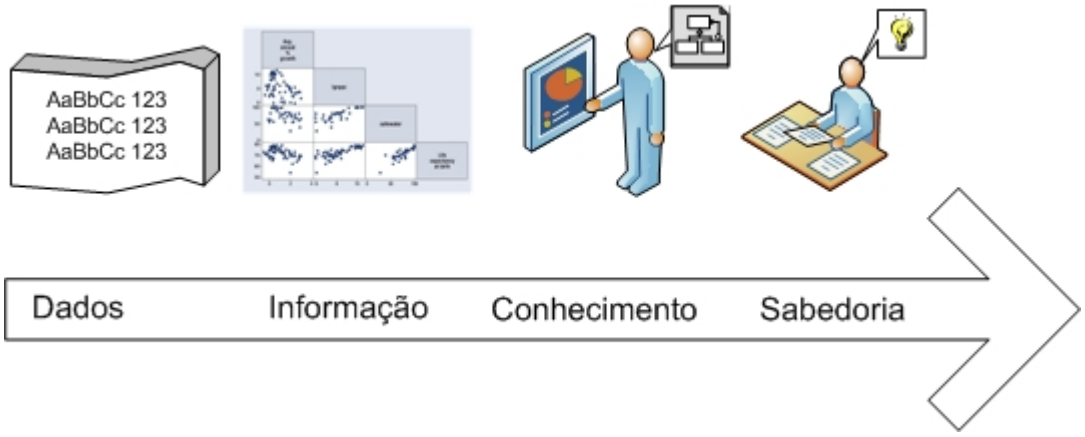

Figura 4 O processo de Compreensão

2.3 METÁFORAS E TÉCNICAS PARA VISUALIZAÇÃO EXPLORATÓRIA

A análise exploratória é geralmente aplicada em dados multidimensionais, aonde há um número expressivo de variáveis (ou atributos) envolvidas. Técnicas de visualização de informação multidimensional são de grande importância para o processo de extração de informação. Estas técnicas utilizam a capacidade humana em detectar padrões através da visualização. Diversas técnicas foram propostas para a exploração visual de dados (Wong e Bergeron, 1997; Eick, 2000; Keim, 2000; Chen, 2006). Keim e Kriegel (1996) classificaram as técnicas e metáforas multidimensionais em orientadas a pixel, geométricas, iconográficas, e hierárquicas. Em 2002, Keim (Keim, 2002) expandiu a classificação para que também fossem considerados os tipos de dados e as técnicas de interação, podendo ser combinados ortogonalmente. Estas categorias serão apresentadas a seguir.

Metáforas Orientadas a Pixel: Nesta categoria, os atributos de dados são mapeados em pixels para que sejam coloridos de acordo com o valor apresentado. O conjunto de valores de cada atributo é apresentado em uma janela individual de forma que, para um conjunto de dados com *m* atributos, a tela será dividida em *m* janelas (Keim e Kriegel, 1996) (Figura 5). O arranjo espacial dos pixels que representam os elementos de dados na tela é efetivo para apresentar agrupamentos de elementos semelhantes mantendo significado semântico dos mesmos. Diversos aspectos podem ser considerados na aplicação desta metáfora, entre eles o arranjo dos pixels nas janelas, o mapeamento da cor e o formato das janelas (Keim, 1996; Keim, 2000). Vale ressaltar que estas metáforas têm dependência direta com a resolução da tela. Neste caso, quanto maior a dimensionalidade do conjunto de dados, menor o número de elementos que podem ser exibidos simultaneamente (Keim, 2002).

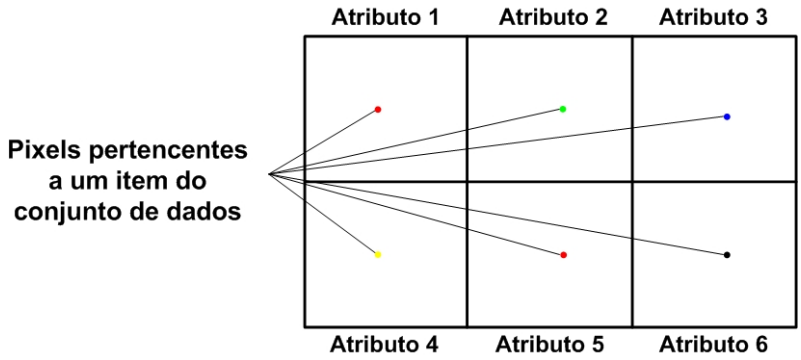

Figura 5 Metáfora Orientada a Pixel. Adaptado de (Keim, 2000)

Metáforas de Projeções Geométricas: Nesta categoria, busca-se por projeções bi ou tridimensionais que auxiliem na representação de informações de interesse em conjuntos de dados multidimensionais. Esta é a metáfora mais familiar para a maioria das pessoas, exemplos conhecidos são os diagramas de dispersão utilizando coordenadas cartesianas. Outro exemplo interessante de projeção são as coordenadas paralelas através da qual um espaço de dimensão m é mapeado em um espaço visual bidimensional, usando m eixos eqüidistantes e paralelos a um dos eixos principais (x ou y). Cada eixo está associado a um atributo, e sobre ele é mapeado linearmente o respectivo intervalo de valores de dados. De acordo com a Figura 6, cada item de dados é exibido como uma linha poligonal que intercepta os eixos no ponto correspondente ao valor do atributo associado ao eixo (Inselberg, 1985).

Esta técnica permite representar muitas dimensões ao mesmo tempo. Ela transforma relações multivariadas em padrões bidimensionais permitindo identificar características como diferenças na distribuição dos dados e correlação entre atributos (Inselberg, 1997; Wegman e Luo, 1996).

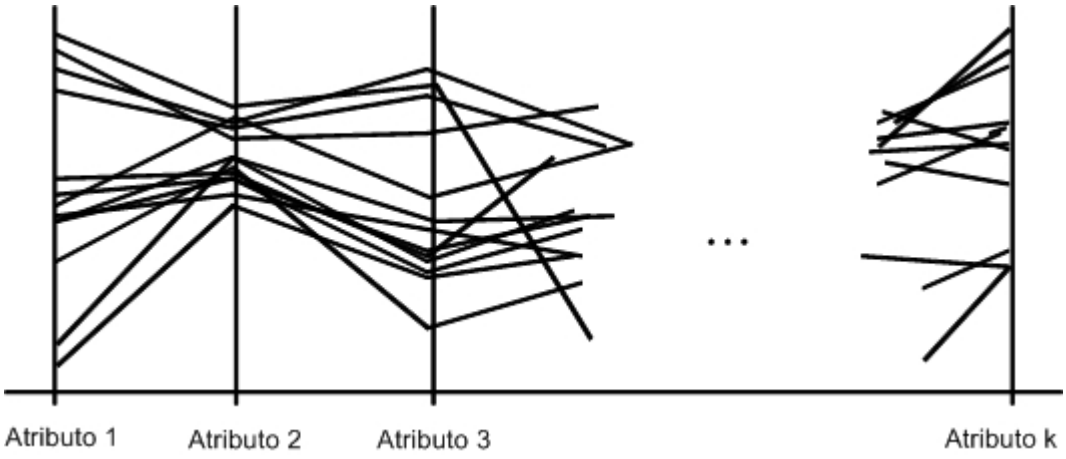

Figura 6 Metáfora de Projeção com Coordenadas Paralelas

Metáforas Iconográficas: Nesta categoria, cada item de dado multidimensional é mapeado em um ícone cujos atributos visuais podem ser configurados para exibir atributos dos dados. Diferentes ícones podem ser usados, um primeiro exemplo é o ícone estrela (*Star Glyphs*) (Ward, 1994). As estrelas são geradas variando-se o comprimento de segmentos de reta arranjados radialmente de acordo com os valores dos a eles associados. Um segundo exemplo, vide Figura 7, é um tipo de ícone que permite visualizar grande volume de dados através da figura de arestas também conhecida como *stick figure* (Thompson, 1977). As duas dimensões da tela são usadas no mapeamento de dois atributos dos dados, com os demais atributos sendo mapeados para ângulos e ou comprimentos de segmentos da figura de arestas. Variações de comprimento, espessura e cor das arestas criam outras possibilidades de representação. Um conjunto de figura de arestas gera padrões de textura que podem facilitar a sua detecção e interpretação, sendo que a configuração escolhida para as arestas influencia na percepção visual dos padrões. A Figura 8 é uma imagem composta por ícones do tipo figura de arestas geradas a partir de cinco imagens de satélite de uma determinada região. Na mesma figura é possível identificar diversas texturas. Nas técnicas iconográficas, o número de

atributos que pode ser apresentado é limitado pelos atributos visuais perceptíveis do ícone adotado. Essas técnicas são adequadas para a detecção de padrões, exceções e variações bruscas nos dados, mas são menos efetivas para comparações. Outros problemas do seu uso são a dificuldade em elaborar um ícone significativo através de uma metáfora visual que revele padrões e também a dificuldade em aumentar a dimensionalidade de um mapeamento, pois implica em modificar o ícone.

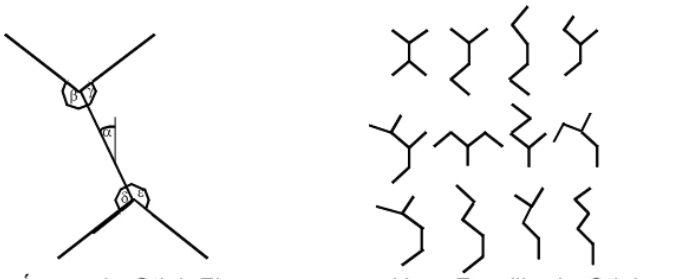

Figura 7 Metáfora Iconográfica (Keim e Kriegel, 1996)

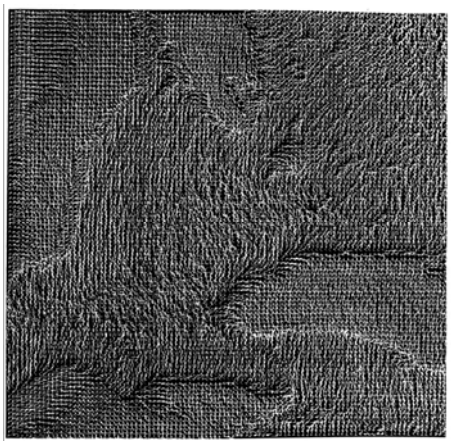

Figura 8 Metáfora Iconográfica (Grinstein et al, 2001)

Metáforas Hierárquicas: Nesta categoria, o espaço de dados é dividido em subespaços hierárquicos para exibição. A idéia é mapear dados não hierárquicos de *k* dimensões em subespaços bidimensionais, mantendo a relação entre eles, sendo que o arranjo de atributos é importante para a representação dos dados (Keim e Kriegel, 1996). Esta metáfora pressupõe que o espaço de dados é hierárquico e mapeia a hierarquia para a representação gráfica. Um exemplo conhecido da técnica hierárquica são as árvores usadas em organogramas ou divisões de tarefas. Outro exemplo é o *mapa em árvores* (Shneiderman,

1992). Esta técnica é particularmente adequada para a visualização de grandes estruturas, mapeando diretamente uma hierarquia pela subdivisão recursiva de retângulos aninhados, vide Figura 9. Os nodos folha são geralmente mapeados em um espaço proporcional ao valor de um atributo escolhido. Um mapa de cores pode representar outros atributos dos nodos folha, e rótulos podem ser exibidos. A representação de dados não hierárquicos é possível, desde que seja definida uma hierarquia de atributos para a exibição.

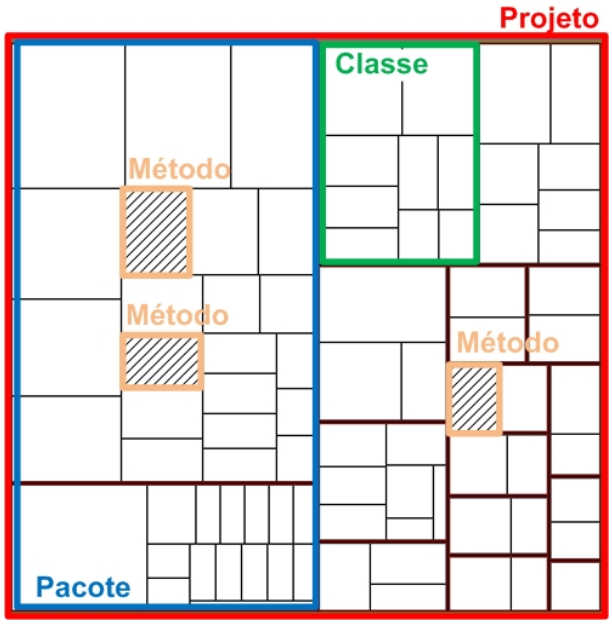

Figura 9 Metáfora Hierárquica – Exemplo do Mapa em Árvores

Na Figura 10 é apresentado um exemplo do mapeamento da estrutura de uma aplicação orientada a objetos. Os retângulos aninhados representam o projeto, os pacotes, as classes e os métodos conforme indicado na mesma figura. Os retângulos mais internos representam os métodos, enquanto que o mais externo representa o projeto. O tamanho do retângulo na figura indica o número de linhas do módulo (projeto, pacote, classe ou método), enquanto que o tom da cor marrom indica a complexidade ciclomática do método (quanto mais escuro maior a complexidade).

Uma vantagem desta metáfora é a representação de hierarquias presentes no conjunto de dados diretamente sobre toda a amplitude do espaço de apresentação. Entretanto, quando a relação hierárquica não é inerente pode ser difícil definir uma organização hierárquica adequada. Algumas limitações desta categoria de visualização são a apresentação

de vários níveis de hierarquia e a dificuldade em se visualizar áreas que representam valores muito pequenos ao utilizar a área segmentada para representar atributos cujos valores têm alta amplitude.

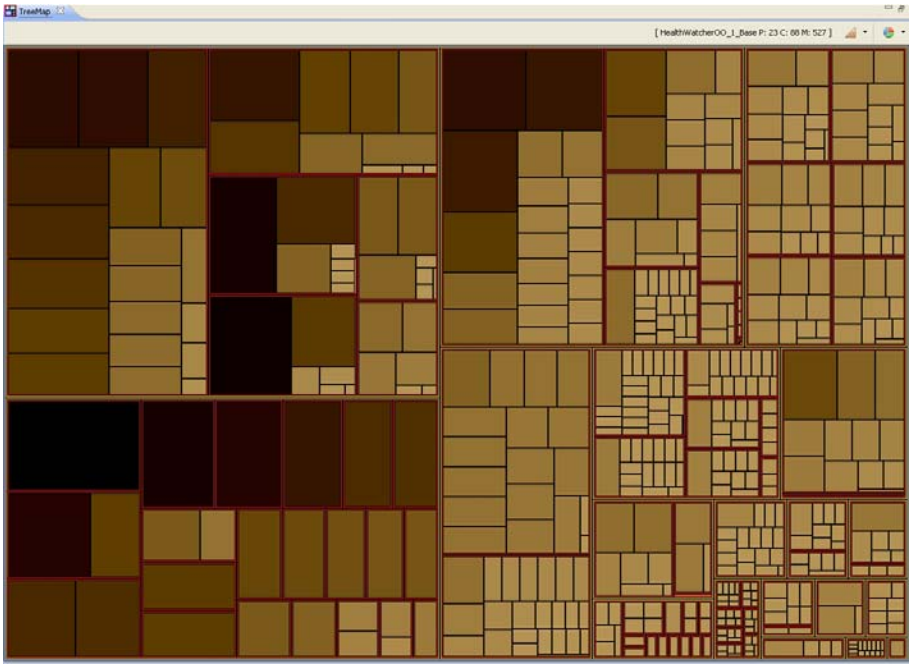

Figura 10 Exemplo de Mapa em Árvores de uma Aplicação OO

2.4 TÉCNICAS DE INTERAÇÃO E DISTORÇÃO

Para a visualização exploratória, recursos de interação constituem ferramentas essenciais, permitindo ao analista selecionar diferentes aspectos do conjunto de dados, ou da representação visual, alterando dinamicamente a visualização para facilitar a detecção de padrões (Keim, 2002). As técnicas de distorção, por outro lado, permitem exibir regiões de interesse da visualização com maior detalhe, enquanto que as demais permanecem inalteradas (Keim, 2002). As subseções a seguir apresentam técnicas de interação e distorção.

2.4.1 Filtragem Interativa

Técnicas de filtragem possibilitam ao analista selecionar interativamente um subconjunto de interesse do conjunto de dados. A seleção pode ser feita diretamente nos dados, ou pela especificação de propriedades específicas dos mesmos.

Nesta categoria estão as consultas dinâmicas (Shneiderman, 1994) habilitadas visualmente por *widgets* tais como controles deslizantes, também chamados de *sliders* na literatura, caixas de seleção, e botões de rádio. Considere controles deslizantes como um exemplo. Eles são constituídos por uma barra graduada, que indica os valores possíveis, e um ou dois cursores que o usuário utiliza para delimitar a faixa de valores de interesse. A Figura 11 ilustra este tipo de controle.

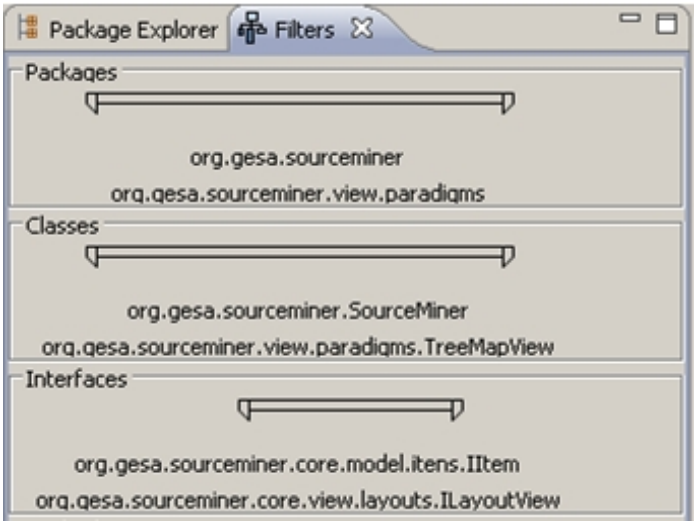

Figura 11 Controle Deslizante

Outro exemplo são as caixas de seleção. Nelas, a consulta é feita através de botões ou caixas de verificação que habilitam e desabilitam os valores de interesse de um dado atributo. A Figura 12 ilustra este tipo de controle.

Qualquer que seja o *widget* de filtragem sugere-se a inclusão de informações sobre os dados selecionados no próprio controle.

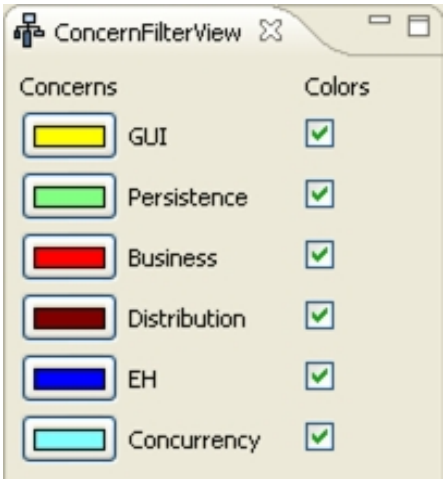

Figura 12 Caixas de Verificação

2.4.2 Zoom Interativo

O uso do zoom interativo permite ter tanto uma visão geral dos dados como também visões detalhadas de áreas de interesse nos dados. O uso desta técnica implica na modificação automática da representação dos dados para que os mesmos sejam apresentados em diferentes níveis de detalhes (Keim e Kriegel, 1996). O uso do zoom interativo para o ajuste do nível de detalhes dependerá da necessidade de compreensão em um dado instante. Para uma resolução baixa, os marcadores gráficos utilizados podem ser constituídos, por exemplo, por um único pixel. Para uma resolução intermediária os marcadores gráficos utilizados podem ser ícones ou elementos gráficos. Para uma resolução alta os marcadores gráficos utilizados podem ser figuras e rótulos contendo a descrição do objeto. Dois importantes tipos de zoom são apresentados a seguir: zoom geométrico e semântico.

O zoom geométrico é usado para aumentar e reduzir o tamanho dos elementos visuais em uma visualização. Aumentando-se o zoom será também aumentado o número de pixels para uso por cada elemento visual. Reduzindo-se o zoom terá efeito oposto (Spence, 2007).

O zoom semântico é usado para mudar o nível de detalhes no qual um conjunto de elementos será apresentado. Aumentando-se ou reduzindo-se o zoom semântico resultará na apresentação de uma nova visualização decorrente da navegação na herança ou estrutura de uma árvore, ou da expansão ou redução do volume de informação de dependências em grafos ou matrizes, por exemplo. A navegação de uma visualização para a seguinte implica em uma nova visualização, pois tem novo significado em relação à visão anterior, daí o nome zoom semântico (Spence, 2007).

2.4.3 Distorção Interativa

A distorção interativa apóia a exploração interativa dos dados preservando a visão geral dos dados. A idéia principal é apresentar parte da área gráfica com maior nível de detalhes, enquanto as demais são apresentados com menor nível de detalhes (Keim e Kriegel, 1996). A representação visual da subárea alvo é apresentada com variações no tamanho dos elementos gráficos. Essas técnicas permitem a visualização de uma subárea de interesse sem perder o relacionamento com o conjunto como um todo. Este é um recurso importante para os casos que existem muitos detalhes ou relacionamentos entre os objetos gráficos, como ocorre em árvores ou redes. Técnicas conhecidas na área são as distorções hiperbólicas e esféricas. Elas podem ser aplicadas em qualquer tipo de metáfora visual.

2.5 UM MODELO DE REFERÊNCIA PARA VISUALIZAÇÃO DE INFORMAÇÃO

Visualização de software é uma especialização de visualização de informação. Por esta razão, a fim de estabelecer um modelo de referência para um ambiente interativo baseado em múltiplas perspectivas para visualização de software (**AIMV**), serão utilizados conceitos do domínio de visualização de informação.

Pesquisadores de visualização de informação identificaram três níveis de interação em ambientes baseados em visualização (Card, Mackinlay e Shneiderman, 1999; Chi, 2000; Santos e Brodlie, 2004). O primeiro, e mais comum, é a interação para a configuração da visão. Ela é relacionada à forma como os elementos serão configurados e posicionados no cenário visual. A distorção interativa é um exemplo de interação para a configuração da visão.

O segundo nível de interação lida com o mapeamento dinâmico entre os atributos reais (do software, no nosso caso) e os atributos visuais utilizados para representar os atributos reais nas visões. Neste tipo de interação, os usuários podem configurar a maneira como propriedades importantes do software tais como complexidade e tamanho serão representados nas visões. Por exemplo, o tamanho de um elemento de software (um atributo real) pode ser representado tanto através do tamanho como através da cor do elemento visual na tela. Para o domínio de engenharia de software é desejável que o usuário tenha a oportunidade de definir quais são os mapeamentos mais adequados para a tarefa em mãos. Apesar da sua importância, o mapeamento dinâmico de atributos é uma questão ainda pouco explorada em visualização de software (Storey e Muller, 1995; Lintern, Michaud, *et al.*, 2003).

O terceiro nível consiste na filtragem dinâmica e seleção dos dados a serem representados nas visões. A presença de muitos dados pode dificultar a interpretação do cenário visual (Langelier, Sahraoui e Poulin, 2005). A visualização seletiva de dados é efetiva para facilitar a identificação de informação relevante, para restringir a análise a determinadas partes dos dados e para controlar o nível de detalhes no qual os dados são apresentados (Consens, Eigler, *et al.*, 1994). Várias iniciativas foram tomadas no domínio de visualização de software para proporcionar a filtragem (elidir temporariamente) objetos, resultando na apresentação parcial dos dados (Ball e Eick, 1994; Chen, Nishimoto e Ramamoorthy, 1990; Eick e Wills, 1993; Linos, Aubet, *et al.*, 1994; Muller, Orgun, *et al.*, 1993; Rajlich, Damaskinos, *et al.*, 1990).

Um bom ponto de partida para o estabelecimento de um modelo de referência para visualização de software é o modelo proposto por Card, Mackinlay e Shneiderman (1999) e apresentado na Figura 13. De acordo com este modelo, a criação de visões ocorre através das seguintes etapas: pré-processamento e transformação de dados, mapeamento visual e criação da visão (Card, Mackinlay e Shneiderman, 1999).

O modelo enfatiza que o processo de visualização de informação é altamente interativo. Ele inicia-se com os dados originais obtidos de um repositório. Os dados então passam por um conjunto de transformações para serem organizados em estruturas de dados apropriadas para a exploração. Esta etapa corresponde à *transformação dos dados* conforme mostrado na Figura 13.

Em seguida, estes dados são usadas para montar as estruturas de dados visuais. Estas estruturas organizam as propriedades dos dados e propriedades visuais de forma a facilitar a construção das metáforas visuais. Esta etapa define o mapeamento de atributos reais – atributos derivados das propriedades dos dados, no nosso caso os atributos do software – para atributos visuais tais como formas, cores e posições na tela do computador. Esta etapa corresponde ao *mapeamento visual* mostrado na Figura 13. É importante ressaltar que esta etapa não lida com renderização, mas com a construção de estruturas de dados a partir das quais as visões podem ser construídas.

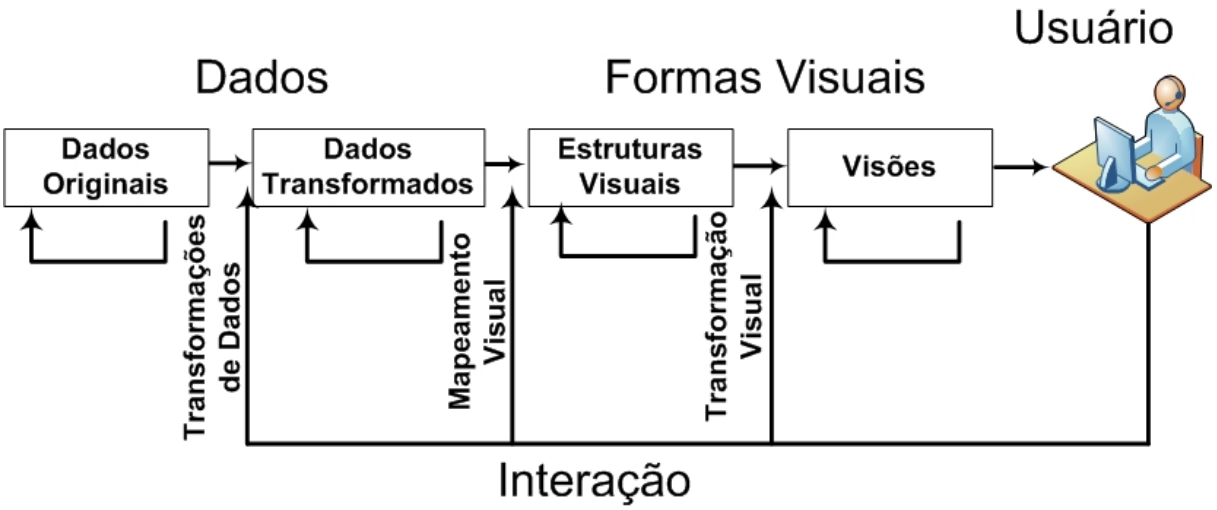

Figura 13 Um Modelo de Referência para Visualização

A última etapa, a *transformação visual* ilustrada na Figura 13, tem o objetivo de representar na tela do computador as informações organizadas na etapa anterior. Metáforas visuais e o mapeamento dos atributos reais em atributos visuais são instanciados para produzir *visões* dos dados selecionados.

Como mencionado anteriormente, sistemas de visualização de informação geralmente são altamente interativos. Neste processo, o usuário deve ter condições de selecionar os dados a serem representados visualmente, modificar o mapeamento entre os

atributos reais e atributos visuais, e alterar a forma como as visões são renderizadas na tela (por exemplo, aplicando recursos de zoom ou visão panorâmica). Para que o processo seja efetivo, o tempo de resposta entre as interações e a reapresentação das visões deve ser o menor possível.

2.6 MÚLTIPLAS PERSPECTIVAS E MÚLTIPLAS VISÕES

Uma visão é uma representação visual de um conjunto de dados. A análise de conjuntos de dados complexos tipicamente requer múltiplas visões, cada uma revelando um aspecto diferente dos dados sob análise (Boukhelifa e Rodgers, 2003). Este é o caso em engenharia de software, pois uma única visão não necessariamente apoiará a compreensão efetiva dos dados nela representados (Storey, 2006). Múltiplas visões incentivam a construção de conhecimento mais aprofundado a respeito dos dados analisados e evitam interpretações distorcidas que poderiam emergir de uma única visão (Ainsworth, 1999).

Apesar de serem perfeitamente compatíveis com a Figura 13, as múltiplas visões não são explicitamente realçadas no modelo original de Card. A Figura 14 apresenta uma adaptação do modelo Card, realizada no contexto desta tese, para a obtenção de um modelo de referência para visualização de informações baseada em múltiplas visões.

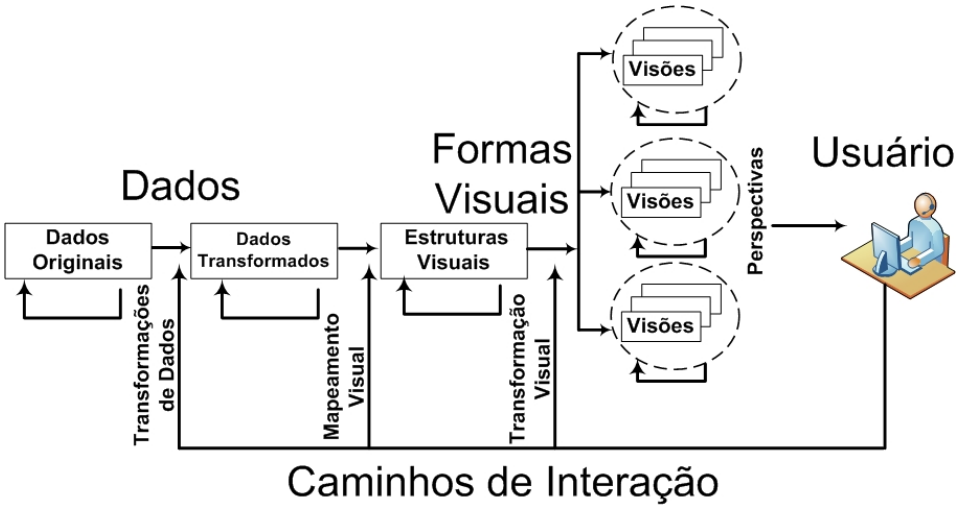

Figura 14 Um Modelo Adaptado para Múltiplas Visões

Coordenando Múltiplas Visões

Sistemas baseados em múltiplas visões têm sido propostos para a investigação de um vasto conjunto de tópicos em visualização de informação. Estes sistemas usualmente utilizam mais de uma metáfora visual para representar os dados (Wang, Woodruff e Kuchinsky, 2000). Metáforas diferentes devem ser utilizadas para revelar aspectos distintos das entidades conceituais analisadas (Roberts, 2000; Becks e Seeling, 2004). Estas metáforas ou formas visuais (vide Figura 14) representam a informação sob diferentes *perspectivas*, cuja análise ocorre através de processos cognitivos complementares.

Múltiplas visões devem ser concebidas de forma consistente para que seja viável a coordenação e integração entre elas. Visões (e formas visuais) devem ser complementares entre si. Um subconjunto de visões deve ser selecionado para uso coordenado durante execução de uma determinada atividade de análise de dados (Wang, Woodruff e Kuchinsky, 2000). A exploração interativa das visões deve ocorrer para que sejam descobertas informações e relacionamentos que se fossem avaliados da forma tradicional permaneceriam ocultos (Wang, Woodruff e Kuchinsky, 2000).

Um desafio no projeto de ferramentas de visualização é como proporcionar um grau de liberdade aos usuários para a execução de tarefas diferentes através de recursos de interação com as representações visuais disponíveis. Neste contexto, as ferramentas de visualização devem ser flexíveis para permitir: a seleção de itens individuais ou subconjuntos de interesses a eles relacionados, a focalização em determinados atributos, a visão panorâmica ou detalhada de trechos de dados, navegação pelos dados para acesso a diversas partes do conjunto, navegação em estruturas hierárquicas, e controle do mapeamento dos atributos nas imagens (North e Shneiderman, 2000).

Cada visão tem um conjunto de funcionalidades (por exemplo, seleção de itens que compõem a visão e navegação através de zoom e visão panorâmica) que devem ser coordenadas de forma que as ações desencadeadas em uma visão possam ter efeito em outras (Wang, Woodruff e Kuchinsky, 2000). Neste cenário há três importantes conceitos para a concepção e uso efetivo de múltiplas visões. Um sistema baseado em múltiplas visões deve suportar:

- *Amarração de navegação* para possibilitar que as ações executadas em uma visão sejam automaticamente propagadas para outras (Shneiderman e Plaisant, 2010; Keim e Kriegel, 1996);
- *Vinculação de dados* para conectar os dados de uma visão com as demais (Shneiderman e Plaisant, 2010; Keim e Kriegel, 1996);
- *Pincelamento coordenado* para possibilitar que os itens de dados correspondentes em diferentes visões sejam destacados simultaneamente (Shneiderman e Plaisant, 2010; Keim e Kriegel, 1996).

2.7 VISUALIZAÇÃO DE SOFTWARE

Apesar do foco desta tese ser em visualização de software, as seções anteriores abordaram basicamente o tema sob a perspectiva da visualização de informação. Esta seção foca-se especificamente no tema visualização de software.

A visualização de software é uma especialização da visualização de informação e pode representar o software do ponto de vista estático, dinâmico e de evolução (Diehl, 2007). A visualização estática tem foco na forma como o software foi construído, incluindo a estrutura, relacionamento e propriedades das entidades que o compõem. A visualização dinâmica representa o software em execução e contribui para o entendimento do seu comportamento. A visualização da evolução do software adiciona a dimensão temporal à visualização estática do mesmo (Caserta e Zendra, 2010). Apesar dos outros dois temas serem muito importantes para a compreensão de software, esta tese limita seu escopo à visualização estática do software. Esta é a mais fundamental das três áreas, e por si só apresenta um amplo conjunto de desafios e oportunidades de pesquisa. A despeito do foco em visualização estática muitas das contribuições deste trabalho podem ser estendidos e adaptados para as áreas de visualização dinâmica e de evolução de software.

Na visualização estática, a representação visual é baseada em metáforas que têm o objetivo de representar a estrutura e/ou os relacionamentos de entidades do software. Esta tese

utiliza módulos de software (pacotes, classes, interfaces e métodos) como as principais entidades para análise e visualização de sistemas.

Na prática, as atividades de engenharia de software requerem a análise de mais de uma propriedade combinada (por exemplo, de um lado a estrutura pacote, classe, método da aplicação e do outro o relacionamento de acoplamento entre estas entidades), o que implica no uso de mais de uma metáfora visual de forma combinada.

Nesta tese definimos que uma *perspectiva* é um conjunto de metáforas visuais utilizadas para representar uma propriedade do software (relacionamento de acoplamento, por exemplo). Perspectivas podem, portanto, ser representadas por diferentes metáforas. Relacionamentos de acoplamento entre módulos de software, por exemplo, podem ser representados por grafos ou matrizes de relacionamentos. Estas metáforas serão instaciadas em visões que em ambientes de desenvolvimento de software atuais podem ser arranjadas e posicionadas de acordo com as necessidades dos seus usuários. Duas questões são quais metáforas visuais adotar e como elas devem ser utilizadas em um sistema para visualização de software. As respostas para estas questões não são simples.

Escolhendo Formas Visuais e Visões para Visualização de Software

Maletic e colegas (2002) consideram cinco dimensões para a concepção de soluções de visualização de software:

- *Atividades*: para que a visualização será necessária;
- *Audiência*: quem vai utilizar a visualização;
- *Alvo:* qual a fonte de dados a ser representada;
- *Representação:* como o conjunto alvo de dados será representado;
- *Meio:* mecanismo através do qual a visualização será apresentada.

A *atividade* descreve para que a visualização será necessária. Em outras palavras, descreve quais atividades de engenharia de software serão apoiadas pela ferramenta de visualização. Por este motivo, para execução de diferentes tipos de atividades, os usuários precisam obter diferentes níveis e tipos de apoio à compreensão do software. Isto resulta em diferentes visões, metáforas e perspectivas. Em relação ao modelo de referência da Figura 14,

tem-se que a atividade determina o tipo das visões e as formas visuais (perspectivas) necessárias para sua composição (Maletic, Marcus e Collard, 2002).

Para compensar a intangibilidade do software, diversas metáforas visuais têm sido utilizadas para sua representação, de forma que os atributos e propriedades do software possam ser mapeados para os atributos visuais disponíveis. Uma metáfora pode ser considerada como um relacionamento sistemático entre dois domínios conceituais, considerados como fonte e destino. Através da projeção da estrutura do domínio fonte para o domínio destino, é possível expressar o domínio destino em função do domínio fonte. Isto implica que é possível conhecer o novo em função daquilo que já se conhece (Lakoff e Johnson, 1980). O potencial de uma metáfora visual é decorrente de sua riqueza semântica, simplicidade e nível de abstração. Sempre excluindo possibilidades de ambigüidade ou perda de significado (Storey, 2006; Maletic, Marcus e Collard, 2002).

MacKinley (1986) definiu dois critérios a serem considerados no mapeamento de estruturas visuais de dados para metáforas visuais: expressividade e efetividade. A expressividade refere-se à capacidade de uma metáfora visual representar toda a informação. A efetividade refere-se à forma como a metáfora visual vai representar a informação. No caso de dados quantitativos, tais como tamanho e complexidade dos módulos de software, devem ser selecionados atributos visuais que sejam efetivos na sua representação. Por exemplo, a forma de um elemento gráfico não é um atributo visual efetivo para representar dados quantitativos, ao passo que o tamanho é um atributo visual efetivo para esta finalidade. A efetividade implica na categorização dos atributos visuais de acordo com a sua capacidade para codificação de diferentes tipos de dados. Esta codificação pode ser sugerida pela ferramenta ou ser deixada a critério do usuário. Neste último caso, o usuário define quais atributos de software serão apresentados por quais atributos visuais de uma determinada visão.

Metáforas visuais são efetivas na transferência de informação (Storey, 2006) (Diehl, 2007). Entretanto, aprender um novo paradigma de visualização não é uma tarefa simples. Segundo Hundhausen (Hundhausen, 1998), a efetividade das metáforas visuais nas atividades de compreensão de software é diretamente influenciada por quatro fatores conforme descrito nos parágrafos a seguir.

O primeiro deles é a fidelidade epistemológica que parte do princípio que o ser humano mantém modelos simbólicos do mundo ao seu redor, sendo tais modelos a base para suas ações e julgamentos (Hundhausen, 2005; Wenger, 1987). A característica principal desta teoria, conforme ilustrado na Figura 15, é a possibilidade de uso dos recursos de visualização para o mapeamento do modelo mental de um indivíduo conhecedor do software para uma metáfora visual que permita a transferência eficiente deste modelo para outro indivíduo. Esta teoria enfatiza o valor da compatibilidade denotativa entre a representação gráfica e o modelo mental do indivíduo conhecedor do artefato.

O segundo é o código dual que se baseia na hipótese de que a cognição está fortemente associada à atividade de dois sistemas simbólicos parcialmente conectados, mas funcionalmente independentes e distintos (Mayer e Anderson, 1991; Paivio, 1983). Um sistema codifica eventos verbais (palavras) enquanto outro codifica eventos não verbais (figuras). Segundo a hipótese de Mayer e Anderson (1991), a visualização que codifica o conhecimento em modelos verbais e não verbais permite ao indivíduo que necessita compreender o software construir representações duais e conexões referenciais entre estas representações. Como resultado, tem-se que este tipo de visualização proporciona a transferência do conhecimento de forma mais eficiente e robusta quando comparado àquelas que não utilizam o código dual.

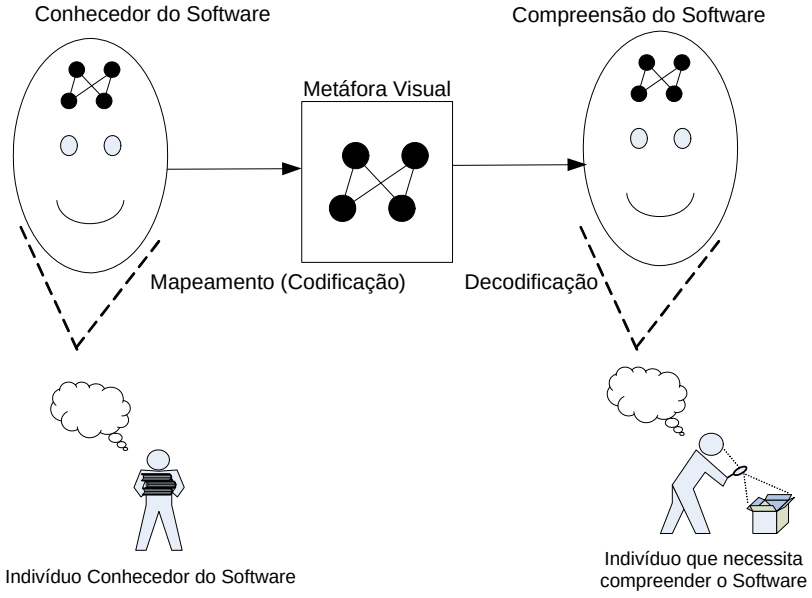

Figura 15 Transferência de Conhecimento

O terceiro fator está relacionado às diferenças individuais. Experimentos conduzidos na área de psicologia têm buscado investigar as diferenças individuais de estilos de aprendizagem e habilidades cognitivas (Conati e Maclaren, 2008). A principal contribuição desta área de pesquisa consiste não somente no conjunto de mecanismos para medir e classificar indivíduos em relação a habilidades cognitivas e estilos de aprendizagem, mas também nos resultados experimentais que conduzem a conclusões importantes relacionadas ao desempenho individual como conseqüência de sua idiossincrasia. Assim, a teoria das diferenças individuais permite concluir que diferenças nas habilidades cognitivas e estilos de aprendizagem terão como conseqüência diferenças de desempenho no uso das metáforas visuais para a compreensão de software. Como exemplo, considerando o modelo de transferência de conhecimento ilustrado na Figura 15, as diferenças individuais relacionadas ao estilo de aprendizagem poderão tornar a decodificação da metáfora visual de um indivíduo mais eficiente e robusta que a de outros.

O quarto e último fator é o construtivismo cognitivo. No lugar de considerar o conhecimento como representações estáticas da realidade que cada indivíduo tem, o construtivismo cognitivo argumenta que não existe conhecimento absoluto (Glaser e Resnick, 1989). Esta teoria defende que os indivíduos constroem de forma evolutiva seu próprio conhecimento a partir das suas experiências. Tornando-se engajados no seu ambiente, indivíduos constroem novo conhecimento a partir da interpretação de novas experiências tendo como referência o contexto que já conhecem. A ênfase dada pelo construtivismo cognitivo na aprendizagem ativa e dinâmica tem implicações importantes na efetividade da visualização de software

Esta teoria sugere ainda que os indivíduos não sejam beneficiários da metodologia proposta simplesmente por serem submetidos às metáforas visuais, independente do nível da fidelidade epistemológica que representam. Os usuários com objetivos de compreensão devem se engajar na seleção e configuração das metáforas visuais, assim como na busca do uso adequado dos seus recursos para que seja alcançada a efetividade necessária. Neste contexto, a proposta desta tese não é do uso de metáforas visuais tão somente como elemento de transporte de conhecimento, mas de uma abordagem que permita aos usuários configurar o ambiente de visualização de acordo com suas preferências e os seus objetivos de compreensão de software.

2.8 FERRAMENTAS DE VISUALIZAÇÃO DE SOFTWARE

Histórico

O papel das representações visuais na compreensão de software tem uma trajetória de mais de meio século. Já em 1947, Goldstein e von Neumann demonstraram o uso de flowcharts nas atividades de compreensão (Goldstein e Neumann, 1947). A primeira fundamentação teórica na área de visualização de informação foi apresentada em 1967 com um trabalho relacionado à semiologia dos gráficos (Weger, 1997), na qual é descrito um framework para a concepção de diagramas.

Os diagramas são as primeiras evidências do uso de visualização de software. Em 1959, Haibt (1959) desenvolveu um sistema para desenhar flowcharts e representar programas em Fortran e linguagem Assembly. Anos depois, Knuth (1963) desenvolveu um sistema para produzir flowcharts como artefato de documentação. Os diagramas de Nassi-Shneiderman foram publicados em 1973 como uma alternativa aos flowcharts (Nassi e Shneiderman, 1973).

A própria representação do código fonte também pode ser considerada a forma mais difundida, ainda que com limitações na sua efetividade, de visualização do software. A forma mais antiga de representação do código fonte é a *pretty-printing* que usou recursos de espaçamento, endentação, e layout para aumentar a legibilidade. Os primeiros *pretty-printers* surgiram na década de 1970 e representavam programas em diversas linguagens tais como LISP, PL/I (Conrow e Smith, 1970) e Pascal (Hueras e Ledgard, 1977).

Já na década de 1980 foram incluídos novos recursos tais como fonte, notações matemáticas e cabeçalhos tais como os adotados pela comunidade Xerox Cedar (Teitelman, 1984). Knuth propôs o uso combinado de código fonte e documentação usando linguagem de marcação, enquanto que um visualizador proposto por Baecker e Marcus (1989) imprimia programas em C de acordo com o estilo gráfico configurado.

Na década de 1980, recursos de visualização de software foram bastante utilizados para a visualização do comportamento do programa para finalidade educacional. A aplicação *Sorting Out Sorting* foi utilizada para a visualização de algoritmos de ordenação tais como *shell-sort*, *bubblesort* e *quick-sort* (Stasko, Domingue, *et al.*, 1998). O recurso de visualização

de software mais conhecido desta década foi o Balsa (Brown e Sedgewick, 1984), uma aplicação para a visualização da execução de algoritmos como apoio educacional e seu sucessor o Balsa-II (Brown, 1988).

Neste histórico, um evento importante para o uso de múltiplas visões ocorreu em 1984 com a disponibilidade de computadores pessoais para o uso de sistemas de visualização interativos para um público mais abrangente. Reiss foi o pioneiro no uso de múltiplas visões em um ambiente integrado de desenvolvimento chamado Pecan (Reiss, 1984).

Müller e colegas em 1986 propuseram o uso do Rigi para a visualização da estrutura do software em termos de componentes e relacionamentos (Muller e Klashinsky, 1988), tornando-se uma ferramenta bastante conhecida para visualização de software.

O avanço no uso de linguagens de programação orientadas a objeto intensificou o interesse na compreensão não somente da estrutura do código, mas também no seu comportamento associado às propriedades da linguagem e do seu paradigma. Kleyn e colegas propuseram o uso do GraphTrace (Haarslev e Möller, 1992), uma ferramenta para visualização dinâmica para a compreensão de sistemas orientados a objeto.

Em 1992, Eick e colegas apresentaram SeeSoft (Eick, Steffen e Sumner, 1992), uma ferramenta de visualização de software para a representação de propriedades relacionadas às linhas de código de sistemas. Em seguida, Ball e Eick (Ball e Eick, 1996) utilizaram SeeSoft em várias situações bem sucedidas. Em 1993, de Pauw e colegas apresentaram JInsight com o uso de novas metáforas visuais para representar o comportamento de sistemas orientados a objeto (Pauw, Helm, *et al.*, 1993).

Outra importante contribuição para a visualização de software industrial é o uso da ferramenta GASE (Holt e Pak, 1996). Gall, Jazayeri e Riva (1999) adotaram o uso combinado de visualizações 2D e 3D para analisar a evolução de sistemas industriais através de dados obtidos da história de releases.

Em 1995, Storey e Muller (1995) apresentaram SHriMP, uma ferramenta para a visualização de software a partir de uma extensão do ambiente Rigi. Em 1999, Lanza apresentou uma proposta para visualização de software chamada visão polimétrica implementada na ferramenta CodeCrawler (Lanza, 2003). Na visão polimétrica, as visualizações são enriquecidas com informações de métricas para que sejam revelados padrões e situações de desvios em sistemas orientados a objetos.

Apesar de não ser este o seu principal objetivo, a *Unified Modeling Language* (UML) tem forte relacionamento com a visualização de software. UML é uma linguagem baseada em diagramas utilizada para descrever estruturas estáticas (por exemplo, diagramas de classes, de pacotes ou de componentes) e o comportamento dinâmico (por exemplo, diagramas de seqüência e de atividades) do software. UML, como parte do *Unified Software Development Process* (USDP) (Jacobson, Booch e Rumbaugh, 1999), é o padrão de fato na indústria para modelagem de software. Os diagramas UML são muito usados para a visualização do software, sendo bastante conhecidos na representação de padrões de projeto como uma proposta de solução reutilizável para problemas recorrentes em engenharia de software (Gamma, 1995).

Cenário Atual

No início do século XXI, a pesquisa em visualização da informação já estava bastante consolidada. Os principais eventos científicos da área demonstram este estado, notadamente a *ACM Computer-Human Interaction* (*CHI*), *User Interface Software and Technology* (*UIST*), *Advanced Visual Interfaces* (*AVI*) e o *IEEE Symposium on Information Visualization* (*IV*).

In 2001, um seminário internacional em visualização de software foi organizado no Centro de Pesquisa em Ciência da Computação Dagstuhl na Alemanha para reunir os principais pesquisadores e praticantes de visualização de software. Cinqüenta pesquisadores de vários países discutiram o estado da arte em visualização de software e identificaram os desafios para a área (Diehl, 2002). A partir daí, decidiu-se pela organização de um evento a cada dois anos com foco em visualização de software: o *ACM Symposyum on Software Visualization*. Em 2002, surgiu outro evento dedicado à área: o *IEEE International Workshop on Visualizing Software for Understanding and Analysis* (VISSOFT). A partir deste momento, diversas propostas de visualização de software tiveram espaço para apresentação e discussão. No Brasil, o primeiro evento dedicado à apresentação e discussão de trabalhos em visualização de software será o I Workshop Brasileiro de Visualização de Software (WBVS) a ser realizado em 27 de setembro no Congresso Brasileiro de Software (CBSoft 2011) em São Paulo. O WBVS 2011 foi proposto e será coordenado pelo autor desta tese, juntamente com

seu orientador e com a Profa. Sandra Fabbri da Universidade Federal de São Carlos (UFSCar).

Pesquisadores vêm investindo também na construção de *frameworks* para visualização de software. Um exemplo interessante é o GSEE (Favre, 2001), um framework orientado a objetos que oferece um conjunto de recursos e funcionalidades para a construção de visualizações a partir de dados disponibilizados do software. Outro exemplo é o Mondrian (Theus, 2002), um framework de visualização para a construção de visualizações de software de forma interativa.

Com o avanço do uso da internet na última década, alguns pesquisadores investiram na disponibilização de recursos de visualização de software na web. Mesnage e Lanza desenvolveram o White Coats (Mesnage e Lanza, 2005), uma aplicação web que usa recursos de 3D para a visualização de evolução de software baseada nos dados de repositórios de versões de software. Anslow e colegas também utilizaram a web como um meio para a visualização de software 3D com recursos de animação (Anslow, Marshall, *et al.*, 2006). Lungu, Lanza e colegas (2007) propuseram o *Small Project Observatory*, uma aplicação direcionada para a disponibilização de dados para apoio à engenharia reversa de ecossistemas de software (Lungu, Lanza, *et al.*, 2010).

Muitos pesquisadores também levaram suas ferramentas de visualização de software para mais próximo dos usuários, integrando-as com ambientes de desenvolvimento de software. Exemplos desta integração são o Creole (Lintern, Michaud, *et al.*, 2003), um plug-in do Eclipse para apresentar as visões do SHriMP (Storey e Muller, 1995), e o X-Ray (Malnati, 2007), um plug-in do Eclipse que implementa a metáfora de visão polimétrica (Lanza e Ducasse, 2003).

Houve ainda um significativo avanço na área de experimentação em compreensão e visualização de software. Neste tema, podem ser citados os estudos seminais conduzidos por Storey e colegas para caracterizar como se dá o apoio de ferramentas à compreensão de software (Storey, 2006) (Storey, Fracchia e Muller, 1999) (Storey, Wong e Muller, 1997).

Pode-se verificar ainda um grande número de propostas de metáforas visuais e ambientes de visualização na última década como uma evidência de que os recursos de visualização têm conquistado cada vez mais espaço em pesquisas e publicações em engenharia de software, ainda que sua adoção na indústria não tenha acompanhado o mesmo

ritmo de crescimento (Koschke, 2003) (Diehl, 2002) (Storey, 2006) (Alam, Boccuzzo, *et al.*, 2009; Anslow, Marshall, *et al.*, 2006; Balzer e Deussen, 2004; Balzer, Deussen e Lewerentz, 2005; Caserta e Zendra, 2010).

Taxonomias e Classificações de Ferramentas de SoftVis

O número crescente de abordagens para a visualização de software conduziu a um número grande de metáforas visuais (a exemplo de (Diehl, 2007) (Stasko, Domingue, *et al.*, 1998) (Zhang, 2003) (Gracanin, Matkovic e Eltoweissy, 2005)) que, combinadas a recursos de interatividade, podem ser classificadas por diferentes taxonomias (Myers, 1990) (Price, 1993) (Roman e Cox, 1993) (Maletic, Marcus e Collard, 2002).

A primeira taxonomia para sistemas de visualização de software foi proposta por Myers (Myers, 1986) (Myers, 1990). Nesta taxonomia os sistemas para visualização de software foram classificados em dois tipos. O primeiro tipo determinava qual parte do programa seria visualizada (por exemplo, código, dados ou algoritmos). O segundo tipo determinava se a informação a ser apresentada seria estática ou dinâmica. Posteriormente, Price (1993) propôs uma taxonomia mais detalhada na qual a visualização é categorizada em seis dimensões: escopo, conteúdo, forma, método, interação e efetividade. A ortogonalidade entre estas taxonomias revela a dificuldade na categorização da visualização de software.

Pacione, Roper e Wood (2004) propuseram uma escala de abstração para classificar ferramentas de visualização conforme ilustra a Figura 16. Nesta figura, a abstração refere-se ao processo de produção de representação simplificada que enfatiza a apresentação de informação importante enquanto retira detalhes que não são relevantes, com o objetivo de reduzir a complexidade e aumentar a compreensibilidade (Berard, 1993). Assim, uma representação abstrata é produzida a partir de uma representação mais detalhada. Neste contexto, é possível gerar representações de um sistema de software em vários níveis de abstração (Pacione, Roper e Wood, 2004). Na Figura 16, uma escala ordinal é utilizada com valores de 1 a 5 baseada na posição relativa dos cinco pontos de referência estabelecidos. No nível microscópico da escala, nível 1, tem-se os depuradores com o menor nível de abstração que uma ferramenta de visualização para análise de software pode produzir. No outro extremo da escala, tem-se a visão geral do sistema com um alto nível de abstração, nível 5.

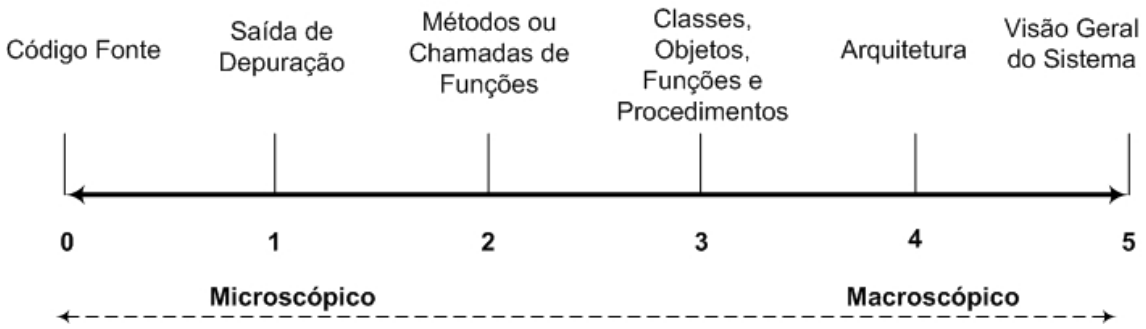

Figura 16 Escala de Abstração dos Objetivos de Compreensão

A Tabela 1 sintetiza uma classificação proposta por Caserta e Zendra (2010) para comparação de ferramentas de visualização estática de software. Nesta classificação, as ferramentas com foco na visualização estática oferecem três níveis diferentes de abstração na visualização: linhas de código fonte, nível intermediário (pacotes, classes e métodos) e arquitetural. Além do nível de abstração, a tabela também informa o foco da visualização: propriedades das linhas de código, nas métricas, na organização da aplicação, ou no relacionamento das entidades de software. A terceira coluna apresenta o nome da metáfora visual, enquanto que a quarta coluna descreve sucintamente a forma como a metáfora é representada. A quinta coluna apresenta as referências da metáfora visual enquanto que a sexta coluna informa o ano de publicação.

Considerando a Tabela 1 e seguindo a classificação de ferramentas para visualização estática de software proposta em (Caserta e Zendra, 2010), as versões do SourceMiner utilizadas nesta tese atendem aos níveis de classe com foco em métricas, assim como também o nível arquitetural com foco na organização da aplicação analisada (objeto de estudo). O SourceMiner não trata o nível/foco de visualização de linhas de código fonte. Por ser um ambiente baseado em múltiplas visões, cada metáfora visual do SourceMiner pode se focar em um nível diferente do software.

Tabela 1 Visualização Estática de Software

Nível	Foco	Metáfora Visual	Representação	Referência	Ano
Linhas	Propriedades da Linha de Código	Seesoft	Pixel colorido 2D	(Eick, Steffen e Sumner, 1992) (Ball e Eick, 1996)	1992
		Sv3d	Cubos colorido 2D	(Maletic, Marcus e Feng, 2003) (Marcus, Feng e Maletic, 2003)	2003
Classes	Métricas	Class BluePrint	Camadas e grafos 2D	(Lanza e Ducasse, 2001) (Ducasse e Lanza, 2005)	1999
Arquitetura	Estrutura e Organização	Mapa em árvores	Retângulos aninhados coloridos 2D	(Shneiderman, 1992) (Turo e Johnson, 1992)	1991
		Mapa em árvores Circular	Círculos aninhados coloridos 2D	(Wang *et al.*, 2006)	1991
		City	Metáfora de Cidade 3D	(Dieberger e Frank, 1998) (Knight e Munro, 2000) (Panas, Berrigan e Grundy, 2003)	1993
		Sunburst	Disposição Radial colorida 2D	(Andrews e Heidegger, 1998) (Chuah, 1998) (Stasko e Zhang, 2000)	1998
		Solar System	Metáfora do Sistema Solar 3D	(Graham, Yang e Berrigan, 2004)	2003
		Voronoi Mapa em árvores	Formas irregulares coloridas 2D	(Balzer, Deussen e Lewerentz, 2005)	2005
	Relacionamentos	Dependency Structure Matrix	Tabela 2D	(Sangal, Jordan, *et al.*, 2005) (Browning, 2001) (Sangal, Jordan, *et al.*, 2005)	1981
		Geon	Diagramas 3D	(Gil e Kent, 1998) (Gogolla, Radfelder e Richters, 1999) (Radfelder e Gogolla, 2000)	1998
		Solar System	Metáfora do Sistema Solar 3D	(Graham, Yang e Berrigan, 2004)	2003
		Landscape	Metáfora Landscape 3D	(Balzer, Noack, *et al.*, 2004) (Balzer e Deussen, 2004)	2004
		Hierarchical Edge Bundles	Grafos com Bundle Edges 2D	(Holten, 2006)	2006
		City	Metáfora de Cidade 3D	(Alam, Boccuzzo, *et al.*, 2009) (Panas, Epperly, et al., 2007)	2007
		3D Clustered Graph	Grafo Clusterizado 3D	(Balzer e Deussen, 2007)	2007
	Métricas	Polymetric	Grafos 2D	(Lanza e Ducasse, 2003)	1999

		Views (Visão Polimétrica)		(Demeyer, Ducasse e Lanza, 1999)	
		Solar System	Metáfora do Sistema Solar 3D	(Graham, Yang e Berrigan, 2004)	2003
		UML Metric View	Diagramas UML com charts	(Termeer, J., *et al.*, 2005)	2005
		Mapa em árvores	Retângulos aminhados 2D com cor e textura	(Holten, Vliegen e van Wijk, 2005)	2005
		City	Metáfora de cidades 3D	(Langelier, Sahraoui e Poulin, 2005) (Wettel e Lanza, 2008) (Wettel e Lanza, 2007) (Dhambri, Sahraoui e Poulin, 2008)	2005
		UML Area of Interest	Diagramas 2D com áreas de interesse	(Byelas e Telea, 2009) (Byelas e Telea, 2006)	2006

2.9 ANÁLISE VISUAL DE INTERESSES (*CONCERNS*)

Um recurso importante oferecido pelo SourceMiner é a representação visual de interesses. Recursos para apoiar a análise de interesses e de sua modularidade foram implementados pelo SourceMiner a partir da **Etapa 2H** do trabalho (vide Figura 3). Esta subseção discute o tema “Análise e Visualização de Interesses” para que ele seja referenciado mais a frente nesta tese.

Um interesse pode ser um conceito, incluindo funcionalidades específicas de domínios, requisitos não funcionais, e padrões de projeto (Robillard e Murphy, 2007). Distribuição, persistência, gerenciamento de transações, e segurança são alguns exemplos de interesses encontrados em muitos programas. Tem-se argumentado que muitas anomalias de modularidade são causadas pela forma como os interesses são estruturados no código fonte (Robillard e Murphy, 2007).

A separação de interesses no desenvolvimento de software foi inicialmente mencionada por Dijkstra (1997) e Parnas (1972). Desde então, o termo interesse tem sido usado para descrever unidades conceituais em um programa. Idealmente, os interesses devem

estar encapsulados em módulos com interfaces bem definidas (Glaser e Resnick, 1989). Entretanto, boa parte do esforço nas atividades de manutenção decorre da forma não apropriada da modularidade dos interesses. Daí a importância em se manter os interesses com boa modularidade ao longo do desenvolvimento e manutenção das aplicações.

Representação Visual de Interesses

Diferentes ferramentas têm sido desenvolvidas para a identificação de interesses no código fonte de aplicações, assim como para a documentação, visualização e gerenciamento de interesses. O reconhecimento da importância tanto da identificação como da análise de interesses em aplicações é grande. Conforme exemplificado nos parágrafos a seguir, a comunidade de programação orientada a aspectos tem publicado vários trabalhos abordando o tema.

Por exemplo, a ferramenta *Feature Exploration and Analysis Tool* (FEAT) (Robillard e Murphy, 2007) apóia a documentação da implementação de interesses em uma representação visual baseada em grafos chamada de grafos de interesse. *SoQueT* (Marin, Moonen e Deursen, 2007) é outra ferramenta que apóia a descrição e documentação de interesses em código fonte através de consultas. *ConcernMapper* (Robillard e Weigand-Warr, 2005) e *ConcernTagger* (Eaddy, Aho e Murphy, 2007) permitem o mapeamento manual dos interesses para o código fonte Java.

As abordagens e ferramentas propostas, principalmente para sistemas de software de tamanho médio a grande, não proporcionam uma visão abrangente dos interesses e de suas propriedades tais como espalhamento (como as linhas de código relacionadas a um interesse são distribuídas por vários módulos (Moha, Guhneuc e Leduc, 2006) e entrelaçamento (grau em que um único módulo contém as linhas de código que cuidam de vários interesses (Tarr, Ossher, *et al.*, 1999). Na maioria das vezes, os interesses são visualizados diretamente no código fonte ou por representações visuais que não se mostram adequadas e efetivas para sua análise. Além disso, em algumas destas ferramentas não é trivial identificar e navegar ao longo dos elementos de software afetados pelos interesses. Por causa destas dificuldades, ferramentas com recursos para visualização de interesses vêm sendo propostas.

Ducasse, Girba e Kuhn (2006) propuseram uma técnica genérica que pode ser usada para visualizar e analisar interesses. Entretanto, esta abordagem permite apenas a visualização de interesses baseada em uma única metáfora visual. Além disso, ela não oferece recursos de interação através de filtros e não permite aos usuários navegar entre as representações visuais e o código fonte.

O *Aspect Browser* (Griswold, Yuan e Kato, 2001) é outra ferramenta proposta para auxiliar na identificação de interesses através de consultas visuais ao código fonte. No Aspect Browser o usuário pode visualizar como os interesses estão espalhados pelas classes. As classes são representadas como retângulos coloridos por interesses, sem que qualquer outra informação seja adicionada a visão. Neste contexto, a análise do espalhamento e entrelaçamento de interesses fica limitada, pois a informação de realização dos interesses é incompleta quando não combinada com visões de estrutura, acoplamento, e outras propriedades do código.

Análise de Modularidade de Interesses

Anomalias de modularidade de interesses são geralmente causadas pela forma como os interesses são implementados no código fonte. A sua identificação pode depender de propriedades que governam a estrutura individual de interesses assim como a forma como dependem entre si. Apesar de ferramentas de visualização de software terem aumentado o suporte à detecção de anomalias, elas ainda estão limitadas a representar estruturas modulares tradicionais tais como métodos, classes e pacotes.

Anomalias de modularidade contribuem para a degeneração do software (Fowler, 1999). Apesar das definições tradicionais de anomalias de modularidade não serem explicitamente baseadas no conceito de interesses, trabalhos recentes têm questionado se a forma como foram realizados está diretamente relacionada com baixa modularidade de interesses (Monteiro e Fernandes, 2005) (Figueiredo, Silva, *et al.*, 2009). Neste contexto, a representação de interesses no SourceMiner tem o objetivo de apoiar na identificação de anomalias de modularidade sensíveis às propriedades dos interesses.

Como exemplos clássicos de anomalias de modularidade podem ser citados o *Feature Envy* (**FE**), o *God Class* (**GC**) e o *Divergent Change* (**DC**). O *Feature Envy* (**FE**)

ocorre quando uma parte do código parece mais interessada em outra classe do que naquela na qual está contido (Fowler, 1999). É comum que apenas parte dos métodos de uma classe sofra desta anomalia. Esta anomalia de modularidade pode ser vista como uma parte mal posicionada do código relacionado a um interesse, isto é, um trecho de código que não implementa o interesse principal da sua própria classe. Assim, o interesse realizado por esta parte de código mal posicionada provavelmente está localizado em uma classe diferente.

God Class (**GC**) é caracterizado pela não coesão de comportamento e pela tendência da classe em atrair para si cada vez mais funcionalidades (Riel, 1996). Em uma perspectiva diferente, podemos analisar o **GC** como classes que implementam muitos interesses e, por este motivo, têm muitas responsabilidades. Isto viola a idéia de que uma classe deve tratar somente de uma abstração principal. Além disso, revela quebra do princípio de separação de interesses.

Divergent Change (**DC**) ocorre quando uma classe é freqüentemente modificada de diferentes formas e por diferentes razões (Fowler, 1999). Dependendo do número de responsabilidades de uma classe, ela pode sofrer mudanças não relacionadas entre si. O fato de uma classe sofrer vários tipos de mudanças pode estar associado com o sintoma de entrelaçamento de interesses. Em outras palavras, uma classe que apresenta vários interesses está propensa a sofrer modificações por diferentes razões.

A inspeção exaustiva pela busca de anomalias de modularidade é uma atividade não trivial que demanda esforço e tempo (Moha, Guhneuc e Leduc, 2006). Um dos motivos para isto é a difícil visualização da existência de interesses e as suas interdependências. Os interesses estão espalhados através de vários módulos, tais como métodos e classes (Figueiredo, da, *et al.*, 2009). Existe um número crescente de ferramentas de visualização que apóiam a identificação de anomalias de modularidade (Parnin, G e Nnadi, 2008; Dhambri, Sahraoui e Poulin, 2008). Mas a maioria delas está limitada a representar visualmente estruturas como pacotes, classes, e métodos em uma única visão e com limitado ou nenhum suporte a anomalias de modularidade.

A forma como a representação visual de interesses suporta a identificação de anomalias de modularidade ainda é uma questão em aberto. As propriedades de interesses tais como entrelaçamento e espalhamento são informações importantes no processo de detecção de anomalias de modularidade. Entretanto, de acordo com estudos experimentais recentes

(Cacho, Sant'Anna, *et al.*, 2006; Garcia, Sant'Anna, *et al.*, 2005), entrelaçamento e espalhamento não parecem ser os únicos fatores decisivos para a identificação de anomalias. Na realidade, contradizendo estudos anteriores (Kiczales, Lamping, *et al.*, 1997), nem todas as formas de espalhamento e entrelaçamento tiveram influência direta na redução da manutenibilidade do código (Figueiredo, da, *et al.*, 2009). Outras propriedades dos interesses podem também indicar a existência de anomalias de modularidade: (i) como um interesse atravessa uma estrutura de hierarquia de herança de um programa (Figueiredo, da, *et al.*, 2009), (ii) como um interesse contribui para as propriedades do módulo, tais como o tamanho da sua interface (Boulanger e Robillard, 2006), acoplamento e coesão (Boulanger e Robillard, 2006), e (iii) como dois ou mais interesses interagem no código fonte (Figueiredo, Cacho, *et al.*, 2008), assim como a forma como dependem entre si e como se sobrepõem. Infelizmente, não há investigação analisando de que forma as propriedades dos interesses apóiam os programadores na identificação de anomalias de modularidade.

2.10 AVALIAÇÃO DE AMBIENTES DE VISUALIZAÇÃO

Estudos experimentais representam o caminho mais natural para a avaliação de diferentes tecnologias utilizadas em engenharia de software. Desta forma, pode-se avaliar a eficácia de métodos, técnicas e ferramentas em diferentes ambientes de desenvolvimento, assim como identificar o melhor contexto de uso e limitações de novas tecnologias propostas (Shull, Carver e Travassos, 2001). Além disso, a execução de conjuntos estudos experimentais possibilitam a construção de bases de conhecimento que podem ser consultadas para a identificação de pontos positivos e oportunidades de melhorias de diferentes técnicas e ferramentas de apoio à engenharia de software (Shull, Mendonça, *et al.*, 2004). Neste contexto, pesquisadores em compreensão e visualização de software vêm conduzindo e relatando estudos experimentais com o objetivo de caracterizar, avaliar, prever, controlar e melhorar os produtos, processos e modelos utilizados (Wohlin, Runeson, *et al.*, 2000; Penta, Kurt e Kraemer, 2007; Lucca e Penta, 2006). Seguindo o exemplo de outras subáreas em

engenharia de software, a apresentação de resultados de estudos experimentais tem sido pré-requisito para o reconhecimento de pesquisas desenvolvidas tanto em compreensão como em visualização de software.

Estudos Experimentais em Engenharia de Software

Um estudo experimental típico em engenharia de software requer a realização de diferentes atividades, cujas características podem variar de acordo com o tipo do estudo (Wohlin, Runeson, *et al.*, 2000; Basili, 1996). Estas atividades compõem as fases de um experimento e são geralmente descritas na literatura como: definição, planejamento, execução, análise e interpretação dos resultados e, finalmente, apresentação das lições aprendidas e empacotamento dos resultados (Kitchenham, Pfleeger, *et al.*, 2002; Lucca e Penta, 2006; Perry, Porter e Votta, 2000; Shull, Singer e Sjoberg, 2007; Sjoberg, Dyba e Jorgensen, 2007).

A abordagem *Goal Question Metric* (Basili e Rombach, 1988) pode ser utilizada na definição dos objetivos de um estudo experimental. Esta abordagem considera cinco parâmetros para o planejamento do experimento: o objeto de estudo, o objetivo, o foco da qualidade, a perspectiva e o contexto. O objeto de estudo especifica as entidades que serão estudadas no experimento, podendo ser um processo, um produto, uma tecnologia, um método, entre outros. O objetivo descreve a finalidade de um estudo conforme opções descritas a seguir: caracterizar (o que é); predizer (o que poderá ocorrer em determinadas condições); controlar (como eventos podem ser manipulados); melhorar (como um processo, técnica ou metodologia pode ser melhorada) ou avaliar (isto é bom?). O foco é o aspecto de interesse no objeto de estudo como, por exemplo, sua confiabilidade, acurácia ou eficiência. A perspectiva é o ponto de vista, ou seja, é a pessoa ou grupo interessado na informação experimental. Finalmente, o contexto descreve o ambiente no qual o experimento está sendo executado (Basili, 1996). Em resumo, a seguinte estrutura pode ser utilizada:

Analisar..... *<Objeto de estudo>*
Com o propósito de..... *<Objetivo>*
Com respeito a..... *<O Foco da qualidade>*
Do ponto de vista de..... *<Perspectiva >*
No contexto de..... *<Contexto>*

A fase de planejamento consiste na seleção do contexto, na formulação das hipóteses, na seleção das variáveis e dos participantes, no projeto do experimento, na preparação da instrumentação, e no estabelecimento da validade do experimento. O resultado desta fase é a elaboração do roteiro do experimento que pode ser registrado em um documentado chamado *protocolo do experimento* para uso durante a sua execução e também na replicação do mesmo (Wohlin, Runeson, *et al.*, 2000; Shull, Singer e Sjoberg, 2007).

A fase de execução deve atentar para o que foi especificado no planejamento do estudo, incluindo a forma de interação com os participantes do experimento para que seja garantida a execução das atividades de acordo com o estabelecido no protocolo do experimento (Wohlin, Runeson, *et al.*, 2000; Shull, Singer e Sjoberg, 2007).

Na fase de análise e interpretação dos resultados será feita a verificação da(s) hipótese(s) para que sejam aceitas ou rejeitadas. Os resultados serão analisados considerando as possíveis ameaças à validade dos resultados (Wohlin, Runeson, *et al.*, 2000; Shull, Singer e Sjoberg, 2007).

Combinando Estudos Experimentais para Avaliação de Novas Práticas

Um estudo não é suficiente para avaliar uma tecnologia, metodologia, abordagem ou processo. Por este motivo, Shull, Carver e Travassos (2001) propuseram uma abordagem para avaliar novas práticas em engenharia de software. A abordagem se inicia com estudos de viabilidade, seguido por estudos observacionais, estudos de caso de atividades reais em engenharia de software e, por último, estudos de caso na indústria.

Pelo fato de ser iterativa e incremental, a abordagem proposta em (Shull, Carver e Travassos, 2001) tem a vantagem de permitir o planejamento e configuração dos estudos de forma alinhada com as oportunidades de melhorias identificadas ao longo do processo e

baseado nos estudos anteriores. Além disso, a abordagem possibilita amadurecimento em relação ao tema estudado por parte dos pesquisadores.

Esta abordagem foi usada como referência para a avaliação do SourceMiner. Esta avaliação é descrita em seis estudos, sendo três estudos preliminares descritos no Capítulo 3, um estudo observacional *in vitro* descrito no Capítulo 6 e dois estudos de caso conduzidos na indústria descritos no Capítulo 7. Na descrição de cada estudo foi adotada a seguinte seqüência: definição, planejamento, execução, resultados obtidos e lições aprendidas do estudo realizado.

2.11 CONCLUSÃO DO CAPÍTULO

A maioria das atividades em engenharia de software tem como pré-requisito a compreensão do software. Diversos estudos apresentam evidências de que o software (marcadamente intangível, complexo e susceptível a mudanças ao longo do tempo), demanda grande esforço para ser compreendido. E isto, sem dúvida, exerce uma grande influência e impacto nas atividades de engenharia de software.

A visualização de software apresenta-se como uma promissora abordagem para tornar a compreensão mais efetiva. Para alcançar este objetivo, as metáforas e técnicas para visualização devem ser combinadas a técnicas de interação exploratória em um ambiente integrado para análise de software.

Este capítulo apresentou conceitos de compreensão e visualização de software para melhor contextualizar o Ambiente Interativo baseado em Múltiplas Visões (**AIMV**) cuja definição e evolução são descritas no próximo capítulo. E cujo modelo conceitual, como um ambiente extensível, interativo e com visões coordenadas, é descrito em detalhe no Capítulo 4 desta tese.

Capítulo

3

Este capítulo apresenta os estudos preliminares realizados para definir, avaliar e evoluir o SourceMiner. Primeiro é apresentada a estratégia adotada na avaliação e, em seguida, os três estudos realizados na evolução do seu modelo. Os resultados dos estudos indicaram a viabilidade do uso do ambiente no apoio a atividades de compreensão de software e resultaram no modelo de Ambiente Interativo baseado em Múltiplas Visões (AIMV) descrito no Capítulo 4 desta tese.

3 DEFININDO E EVOLUINDO O AMBIENTE

Com o objetivo de criar um ambiente para visualização de software adotou-se uma abordagem iterativa e incremental. O ambiente foi construído, avaliado e refinado em uma série de estudos, denominados de *estudos preliminares*. O termo *preliminar* é utilizado pelo fato de representar etapas de proposição e avaliação empírica do modelo proposto para o SourceMiner, momento no qual ainda estavam sendo coletadas evidências sobre as funcionalidades a serem oferecidas pelo ambiente. O objetivo destes estudos era avaliá-lo e aperfeiçoá-lo para que pudesse, de forma mais consistente e efetiva, apoiar as atividades de compreensão de software. A estratégia adotada é mostrada na Figura 17. Cada ciclo de estudos busca identificar ajustes para o refinamento do ambiente visual proposto e também a geração de novas hipóteses a serem investigadas nos estudos seguintes (Mafra, Barcelos e Travassos, 2006).

A abordagem iterativa e incremental adotada teve o objetivo de permitir que o planejamento e configuração dos estudos estejam alinhados com as oportunidades de melhorias identificadas. Além disso, possibilitou o amadurecimento em relação ao modelo proposto e ao tema estudado por parte dos pesquisadores (Shull, Carver e Travassos, 2001).

Utilizando o gabarito *Goal Question Metric* (GQM) (Basili e Rombach, 1988), o objetivo que norteou o conjunto de estudos realizados nesta tese de doutorado foi:

Analisar o *SourceMiner, um ambiente interativo baseado em múltiplas visões*
Com o propósito de *caracterizá-lo*
Em relação à efetividade *no apoio às atividades de compreensão de software*
Do ponto de vista *dos programadores*
No contexto de *desenvolvimento e manutenção de software.*

Tendo como referência o objetivo apresentado acima, as seguintes perguntas foram investigadas nos estudos realizados:

a) Qual a efetividade de um AIMV no apoio às atividades de compreensão de software?
b) Quais as principais funcionalidades que um AIMV deve oferecer para apoiar as atividades de compreensão de software?
c) Qual conjunto de informação deve ser disponibilizado pelo AIMV para apoiar de forma efetiva os usuários na compreensão de software?

As perguntas acima nortearam o planejamento e a execução não somente dos estudos preliminares reportados neste capítulo, mas também do estudo de viabilidade e dos estudos de caso apresentados nos Capítulos 6 e 7 desta tese. O formato adotado no desenvolvimento de cada estudo foi:

- Descrição da Versão do SourceMiner Utilizada no Estudo
- Planejamento do Estudo
- Execução do Estudo
- Análise e Discussão dos Resultados
- Ameaças à validade do estudo
- Lições Aprendidas

A Figura 17, derivada a partir da Figura 3 apresentada no Capítulo 1 desta tese, ilustra através de um diagrama os estudos preliminares conduzidos e o relacionamento existente entre eles. O ponto de partida para os estudos foram o objetivo e as perguntas apresentadas anteriormente. Estas perguntas foram elaboradas a partir de revisão bibliográfica

na literatura e também considerando as ferramentas de visualização de software desenvolvidas por diversos grupos de pesquisa.

As linhas tracejadas em azul indicam a etapa de cada estudo. As linhas tracejadas pretas indicam a transição e evolução entre os estudos. Em cada transição foi possível identificar, a partir da análise e discussão dos resultados do estudo anterior, as lições aprendidas a serem consideradas para o estudo seguinte. As lições aprendidas que motivaram a evolução do ambiente estavam relacionadas a um ou mais dos seguintes itens: (i) **tipos de atividades** de compreensão de software, (ii) **propriedades relevantes** para a execução das atividades, (iii) **metáforas visuais**, e (iv) **recursos interativos** disponibilizados pelo ambiente de visualização. Estes quatro itens serão abordados e comentados ao longo da descrição e apresentação dos resultados de cada um dos estudos.

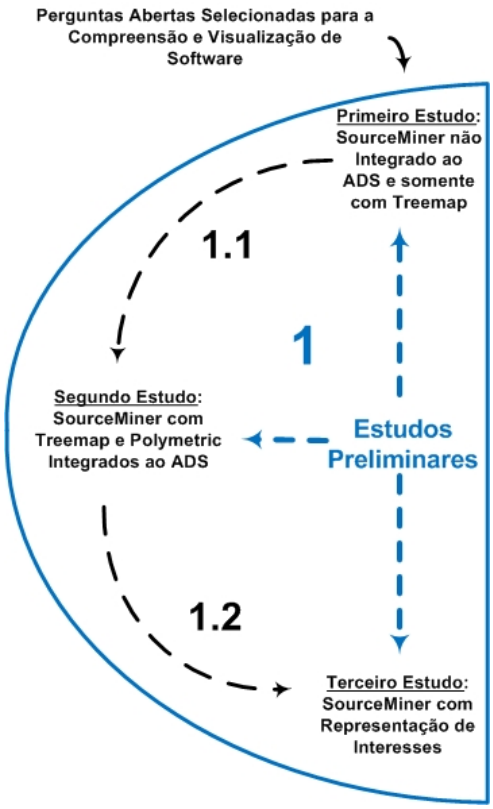

Figura 17 Estudos Preliminares com o SourceMiner

3.1 PRIMEIRO ESTUDO PRELIMINAR

O objetivo dos estudos preliminares é analisar a viabilidade das práticas propostas. Neste primeiro estudo o objetivo foi avaliar a potencialidade do ambiente proposto em apoiar as atividades de compreensão de software. Por se tratar de um estudo de viabilidade, o resultado foi um parecer em relação à continuação ou não da pesquisa. Caso o ambiente se mostrasse viável, propostas de ajustes tanto do modelo como da arquitetura seriam executadas no SourceMiner. Desta forma, o primeiro estudo mostrou a viabilidade do SourceMiner no apoio às atividade de compreensão de software e identificou um conjunto de mudanças para a sua evolução (vide Figura 18).

As atividades solicitadas aos participantes no primeiro estudo preliminar tiveram como foco a análise de como o SourceMiner auxiliava na identificação de informações relacionadas à organização dos pacotes, classes e métodos de uma aplicação, e se estas informações eram relevantes no contexto de atividades de compreensão de software. O primeiro estudo preliminar foi publicado em (Carneiro, Orrico e Mendonça, 2007) representado como **P1** na Figura 3. Este estudo foi executado no primeiro trimestre de 2007.

3.1.1 Descrição da Versão do SourceMiner Utilizada no Estudo

Neste estudo foi utilizada a versão do SourceMiner representada na Figura 18. Nesta versão, a única metáfora visual disponível era o mapa em árvores para apresentar a estrutura pacote, classe, método da aplicação. Esta versão do SourceMiner não era integrada ao ambiente de desenvolvimento de software (ADS). Para que as informações do código fonte Java fossem representadas visualmente, foi desenvolvido um analisador sintático utilizando o *JavaCC* (JavaCC, 2010). Como ferramental de apoio para a implementação do mapa em árvores foi utilizado o *Prefuse Java Toolkit* (Prefuse, 2010).

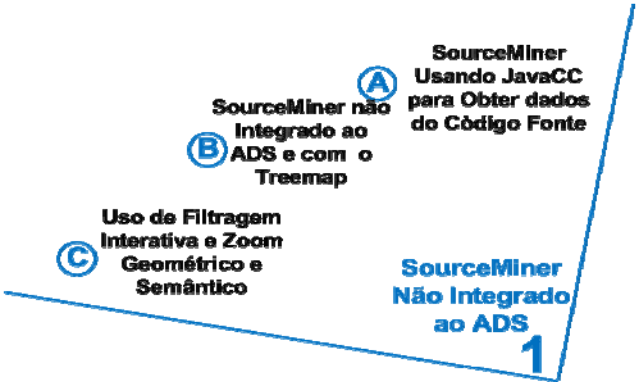

Figura 18 Versão adotada no Primeiro Estudo Preliminar

Atributos visuais tais como cor e tamanho dos retângulos do mapa em árvores foram mapeados para complexidade ciclomática e linhas de código. O mapeamento entre atributos visuais e atributos do software desta versão já permitia a configuração pelo usuário do cenário visual. Os módulos também já podiam ser filtrados de acordo com valores selecionados das métricas e atributos da estrutura do projeto disponíveis tais como pacotes, classes e métodos conforme apresentado na parte direita da Figura 19. Estes mecanismos de interação já apresentavam resposta imediata na tela.

De acordo com a Tabela 1 do Capítulo 2, a metáfora visual de *mapa em árvores* é apropriada para a representação da estrutura e organização da aplicação (Caserta e Zendra, 2010). Esta característica possibilita o uso do *mapa em árvores* para apoiar as atividades de compreensão de software relacionadas aos itens a seguir, sem que haja necessariamente acesso ao código fonte:

- Informações estruturais descritivas como, por exemplo, quais classes estão em determinados pacotes, e quais métodos estão em determinadas classes;
- Informações estruturais quantitativas como, por exemplo, quantas classes existem em um pacote, e qual classe tem o maior número de métodos;
- Informações de tamanho do código fonte como, por exemplo, qual o maior pacote, classe ou método;

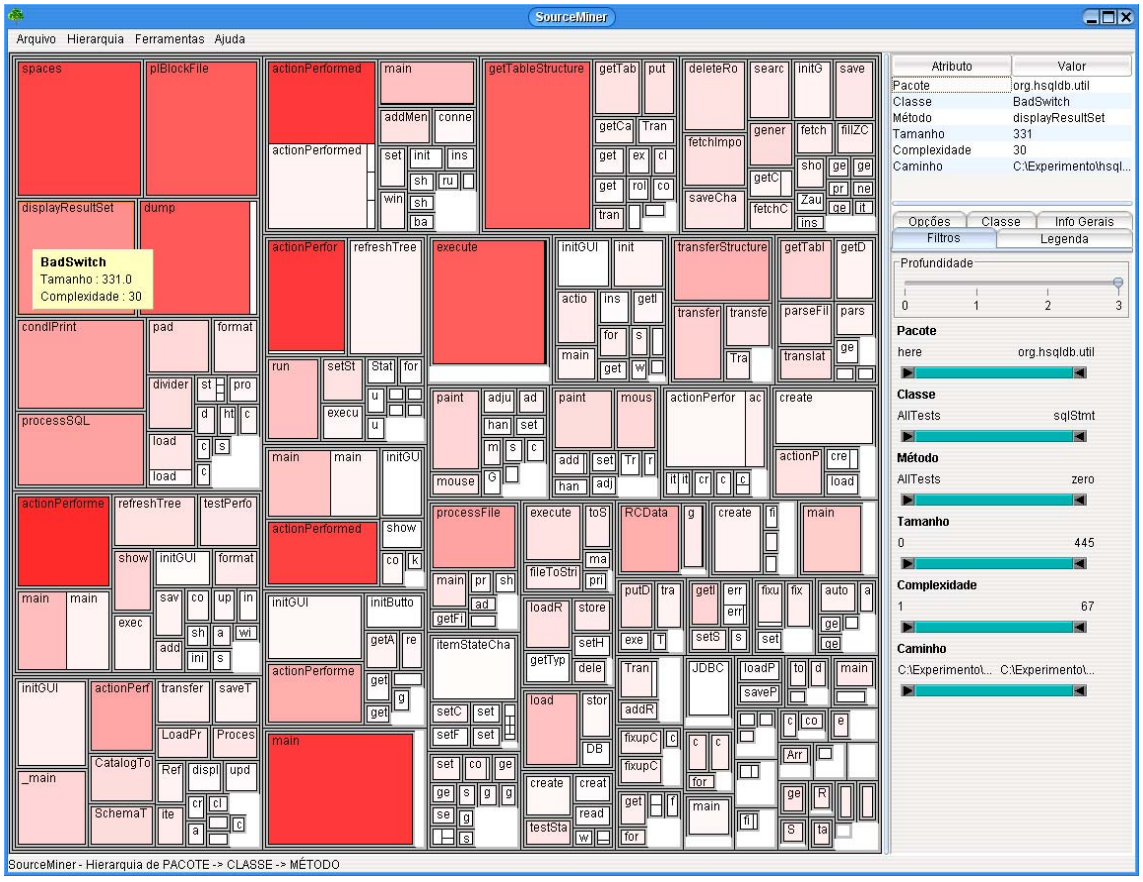

Figura 19 A Primeira Versão com o Mapa em Árvore

3.1.2 Planejamento do Estudo

O propósito deste primeiro estudo de viabilidade foi responder à questão: "O SourceMiner é um ambiente viável para a apoiar execução de atividades de compreensão de software, em especial aquelas que dependem de informações da estrutura pacote classe método da aplicação?"

O objetivo deste estudo seguindo a abordagem GQM (Basili e Rombach, 1988):

Analisar o *SourceMiner, um ambiente interativo*

Com o propósito de *caracterizá-lo*

Em relação à *viabilidade no apoio às atividades de compreensão de software através da metáfora visual de mapas em árvores*

Do ponto de vista *dos pesquisadores do ambiente SourceMiner*

No contexto de *atividades de compreensão de aplicações desenvolvidas usando a linguagem Java em um ambiente acadêmico in-vitro.*

Considerando as características da versão do SourceMiner disponíveis à época e descritas anteriormente, este estudo preliminar teve o propósito de responder às perguntas de pesquisa (**PP**) a seguir.

Pergunta de Pesquisa 1 (PP1): Analisar a metáfora visual mapa em árvores com o objetivo de avaliá-la com respeito à sua efetividade em apoiar a identificação de informações estruturais de software.

Pergunta de Pesquisa 2 (PP2): Analisar a metáfora visual mapa em árvores com o objetivo de avaliá-la com respeito à sua efetividade em apoiar a identificação de informações sobre o número de módulos e sub-módulos do software.

Pergunta de Pesquisa 3 (PP3): Analisar a metáfora visual mapa em árvores com o objetivo de avaliá-lo com respeito à sua efetividade em apoiar a identificação informações de tamanho e complexidade ciclomática da aplicação analisada.

Pergunta de Pesquisa 4 (PP4): Analisar a metáfora visual mapa em árvores com o objetivo de avaliá-la com respeito à sua efetividade em apoiar na identificação de anomalias de modularidade (code smells) (Fowler, 1999) e oportunidades de reestruturação na aplicação.

As perguntas PP1, PP2 e PP3 estavam diretamente relacionadas às características da metáfora visual (mapa em árvores) disponível no momento. As perguntas PP1, PP2 e PP3 devem ser utilizadas de forma combinada para que seja analisado de forma abstrata até que ponto o ambiente apóia a resolução de problemas de compreensão de software. A pergunta PP4, ao contrário das três anteriores, teve o objetivo de analisar até que ponto o SourceMiner seria efetivo no apoio a uma necessidade bastante comum e real de compreensão de software.

O estudo trabalhou com duas variáveis independentes (aquelas que influenciam na aplicação do tratamento e conseqüentemente nos resultados do estudo):

- Apoio à execução das atividades. As atividades foram realizadas com o SourceMiner e com o ADS Eclipse ou somente com o ADS.

- O grau de experiência dos participantes. Três deles tinham experiência na indústria enquanto que os outros tinham somente experiência com atividades da graduação (o estudo foi realizado em ambiente acadêmico *in-vitro*).

A variável dependente deste estudo foi a efetividade na identificação da:

- Estrutura de pacotes, classes e métodos da aplicação;
- Número de pacotes, classes e métodos;
- Pacotes, classes e métodos que se destacam na aplicação em relação ao seu tamanho;
- Métodos da aplicação que se destacam em relação ao seu valor de complexidade ciclomática;
- Identificação de oportunidades de reestruturação e anomalias de modularidade na aplicação.

Todas as aplicações utilizadas no tutorial e no estudo foram do tipo software livre. O tutorial utilizou a aplicação BlackJack, um jogo composto por cinco classes e 2075 linhas de código. Durante a execução das atividades, os participantes utilizaram as seguintes aplicações desenvolvidas com a linguagem Java: HSQLDB, versão 1.7.3.3, e Tyrant, versão 0.3.3.4. A primeira é um banco de dados relacional com 15 pacotes, 323 classes, 4141 métodos e mais de 75000 linhas de código. A segunda aplicação é um jogo com 13 pacotes, 273 classes, 2292 métodos e mais de 38000 linhas de código.

Tabela 2 Projeto Experimental do Estudo

Participante	**Aplicações**	
	Eclipse	**Eclipse + SourceMiner**
Participante 1	HSQLDB	Tyrant
Participante 2	Tyrant	HSQLDB
...		
Participante 9	HSQLDB	Tyrant
Participante 10	Tyrant	HSQLDB

A alocação dos participantes aos grupos de tratamento ocorreu de acordo com a Tabela 2. Antes do tutorial, os participantes assistiram uma apresentação com os objetivos do estudo. No tutorial, os participantes utilizaram os recursos do SourceMiner para analisar a aplicação BlackJack e também receberam orientações de como preencher os formulários do experimento.

Atividades do Estudo. Os participantes receberam um questionário com as seguintes atividades:

a) Identifique e apresente a estrutura de pacotes das aplicações sob análise;
b) Identifique e apresente os pacotes, classes e métodos que se destacam em relação ao seu tamanho na aplicação;
c) Identifique e apresente os métodos que se destacam em relação a sua complexidade na aplicação;
d) Identifique oportunidades de reestruturação e anomalias de modularidade na aplicação.

Estas atividades foram previamente analisadas e executadas pelo autor desta tese para compor uma lista de referência das atividades do estudo. A lista de referência foi utilizada na análise de dados e na comparação com as repostas fornecidas pelos participantes. Para a elaboração das listas de referência, as perguntas de “a” a “c” foram respondidas pelo autor da tese usando o SourceMiner, e também dos recursos oferecidos pelo ambiente de desenvolvimento de software. A pergunta “d” foi respondida por intermédio de consulta a listas de discussão e repositórios das aplicações HSQLDB e Tyrant utilizadas no estudo.

3.1.3 Execução do Estudo

As atividades foram executadas em laboratório por oito estudantes de graduação e dois graduados. Todos estavam envolvidos de alguma forma com atividades do nosso grupo de pesquisa e tinham interesse em engenharia de software experimental (Wohlin, Runeson, *et al.*, 2000). Os participantes foram solicitados a executar as atividades individualmente por um

período de duas horas. A participação foi voluntária e os participantes foram informados de que tanto a qualidade das respostas fornecidas como as estratégias aplicadas eram importantes para o estudo.

No início do estudo, os participantes preencheram questionário a respeito de atividades de programação (incluindo a linguagem de programação Java), experiência profissional e acadêmica em atividades de desenvolvimento e manutenção de software. Ao final do estudo, os participantes completaram outro questionário para registro de sugestões e comentários, incluindo pontos positivos, oportunidades de melhorias tanto no estudo como no SourceMiner. O questionário também reservou um espaço para qualquer outro comentário dos participantes.

3.1.4 Análise e Discussão dos Resultados

Pergunta de Pesquisa 1

Hipótese Nula 1: A identificação e o reconhecimento da estrutura de pacotes da aplicação usando o SourceMiner e o ADS em termos de efetividade é igual se comparada com o uso apenas do Eclipse.

Hipótese Alternativa 1: A identificação e o reconhecimento da estrutura de pacotes da aplicação através do SourceMiner e do ADS em termos de efetividade é maior se comparada com o uso apenas do Eclipse.

Em relação à pergunta “a” usando somente o Eclipse, verificou-se que todos os participantes identificaram corretamente a estrutura das aplicações. De acordo com as informações registradas nos questionários, verificou-se que o Package Explorer foi a visão mais utilizada para esta atividade. Usando-se o Eclipse e o SourceMiner, também foram obtidos os mesmos resultados com a diferença que neste caso o mapa em árvores foi utilizado de forma combinada com o Package Explorer para navegação na estrutura de pacotes da

aplicação. Comparando-se o intervalo de tempo médio decorrido para a identificação da estrutura de pacotes, classes e métodos, verificou-se que com o uso do SourceMiner houve uma redução média de aproximadamente 20% do tempo se comparado com o uso somente do ADS. Apesar das respostas terem sido as mesmas para a análise da estrutura com os dois tratamentos (mesma eficácia), o uso do SourceMiner resultou na realização da atividade de forma mais eficiente.

Os participantes relataram que "*a navegação na estrutura da aplicação foi facilitada pelo mapa em árvores, pois nesta visão não foi necessário o uso intensivo da barra de rolagem nem de cliques necessários para expandir ou reduzir a estrutura de pacotes, classes e métodos como ocorre no Package Explorer*". Um dos participantes ainda complementou que, "*apesar da atividade não ter solicitado informações de classes e métodos, eu naturalmente quis conhecer algumas classes e alguns métodos da aplicação. Com o Package Explorer esta atividade demanda mais tempo em função dos recursos de interatividade do que aquele correspondente ao uso do mapa em árvores*."

As respostas dos participantes apresentaram evidências iniciais de que a hipótese é verdadeira, pois apesar da eficácia ter sido a mesma para os dois tratamentos, a comparação entre os tempos de execução apresenta diferença de eficiência entre os tratamentos.

Pergunta de Pesquisa 2

Hipótese Nula 2: A identificação de pacotes, classes e métodos que se destacaram em relação ao tamanho usando o SourceMiner e o ADS em termos de efetividade é igual se comparada com o uso apenas do Eclipse.

Hipótese Alternativa 2: A identificação de pacotes, classes e métodos que se destacaram em relação ao tamanho com o SourceMiner é mais efetiva se comparada com o uso apenas do Eclipse.

Para responder à pergunta "b" usando somente o Eclipse os participantes relataram dificuldade para encontrar as respostas e classificaram o trabalho como cansativo. O comentário a seguir de um dos participantes ilustra este caso: "*apesar do Eclipse oferecer as informações de tamanho solicitadas, elas não são apresentadas de forma intuitiva e usando recursos visuais como o SourceMiner. Além disso, a forma como é apresentada originalmente*

não favorece a comparação das entidades em relação ao tamanho". Somente com o Eclipse, nenhum participante identificou o conjunto completo de pacotes, classes e métodos que se destacavam em relação ao tamanho. Eles identificaram somente parte das entidades registradas na lista de referência do estudo.

Usando-se o Eclipse e o SourceMiner, todos os participantes identificaram a maioria dos pacotes, classes e métodos que se destacavam em relação ao tamanho. O comentário de um participante destaca o uso dos recursos interativos: "*a possibilidade do uso do filtro por tamanho de métodos serve de apoio para imediatamente encontrar as respostas solicitadas*".

O estudo, portanto, apresentou evidências iniciais de que a hipótese é verdadeira, pois houve diferença significativa na identificação das disparidades de tamanhos entre os métodos da aplicação.

Pergunta de Pesquisa 3

Hipótese Nula 3: A identificação de métodos que se destacaram em relação à complexidade ciclomática usando o SourceMiner e o ADS em termos de efetividade é igual se comparada com o uso apenas do Eclipse.

Hipótese Alternativa 3: A identificação de métodos que se destacaram em relação à complexidade ciclomática é mais efetiva com o SourceMiner se comparada com o uso apenas do Eclipse.

Na realização desta atividade usando somente o Eclipse nenhum participante respondeu os métodos que se destacavam em relação à complexidade ciclomática. Usando o Eclipse e o SourceMiner todos os participantes identificaram os métodos que se destacavam em relação à complexidade ciclomática. Todos os participantes comentaram a dificuldade em ter que calcular manualmente a complexidade ciclomática, enquanto que por outro lado estas informações já são apresentadas visualmente pelo SourceMiner.

Pergunta de Pesquisa 4

Hipótese Nula 4: A identificação de anomalias de modularidade e oportunidades de reestruturação com o SourceMiner é mais efetiva se comparada com o uso apenas do Eclipse.

Hipótese Alternativa 4: A identificação de anomalias de modularidade e oportunidades de reestruturação com o SourceMiner é mais efetiva se comparada com o uso apenas do Eclipse.

Para os dois tratamentos, nenhum participante apresentou sugestões de anomalias de modularidade (*code smells*) ou de reestruturação consistentes com as ocorrências coletadas das listas de discussão das aplicações. Por este motivo não foi possível comprovar a hipótese. Entretanto, dois dos participantes fizeram as seguintes observações: “não tenho certeza em relação a qual anomalia estas classes representam, mas de qualquer forma as classes Monster, PrintFormat e ConversionSpecification da aplicação Tyrant (todas com mais de com mais de 2200 linhas) devem caracterizar alguma anomalia de modularidade. Isto ficou evidente através da análise visual do mapa em árvores”.

Ameaças à Validade do Estudo. Todos os participantes do estudo foram estudantes ou recém graduados em Ciência da Computação, mas a maioria deles relatou experiência na participação de projetos na indústria. As perguntas utilizadas no questionário (neste caso de “a” até “c”) foram elaboradas de acordo com as funcionalidades oferecidas pelo mapa em árvores. Por este motivo foi incluída também a pergunta “d” para analisar até que ponto a metáfora visual disponibilizada oferece suporte a atividades comuns de compreensão de software tais como a identificação oportunidades de reestruturação e anomalias de modularidade em uma aplicação. Os próximos estudos poderiam replicar o protocolo elaborado para este experimento para permitir a generalização dos resultados. Entretanto, foi dada prioridade ao endereçamento das oportunidades de melhorias identificadas e planejamento de um novo estudo para analisar a efetividade da nova versão do ambiente com as funcionalidades incluídas.

3.1.5 Lições Aprendidas

Os resultados do primeiro estudo mostraram que o SourceMiner era viável. Entretanto, diversos ajustes e refinamentos identificados ao longo do estudo seriam necessários para que o ambiente apoiasse de forma efetiva as atividades de compreensão de software.

A maioria dos participantes do primeiro estudo relatou que os resultados poderiam ser mais promissores se o SourceMiner fosse integrado ao ambiente de desenvolvimento de software (ADS). Importantes novas funcionalidades poderiam ser oferecidas como resultado desta integração. A primeira delas seria acessar o código fonte no próprio editor do ADS a partir da representação visual de uma entidade de software. A segunda seria a possibilidade de utilizar recursos do próprio Eclipse em substituição ao *JavaCC* para obter informações do código fonte para a representação visual do software analisado.

Outra questão evidenciada durante o experimento foi que o mapa em árvores não se mostrou suficiente para a identificação das anomalias de modularidade. O que foi disponibilizado estava limitado somente a informações relacionadas à estrutura hierárquica de pacotes, classes e métodos do projeto. Mesmo tendo enriquecido a metáfora visual com informações de tamanho e complexidade através das cores e tamanho dos retângulos, tais recursos foram considerados úteis, mas limitados, ainda que úteis. A respeito desta questão os participantes relataram duas importantes observações. A primeira foi que eles sentiram a necessidade de algum tipo de informação das funcionalidades da aplicação nas representações visuais. Desta forma poderiam ser feitas associações entre as entidades do software e qual a responsabilidade destas entidades no contexto dos requisitos funcionais e não funcionais da aplicação. A segunda observação foi que somente as propriedades representadas pelo mapa em árvores não seriam suficiente para a execução de atividades reais em engenharia de software tais como a identificação de anomalias de modularidade. Para isto seria necessária a inclusão da representação de outras propriedades tais como relacionamentos de acoplamento entre as entidades de software e hierarquia de herança existente entre elas.

A análise do perfil de uso do ambiente foi limitada e, de certa forma, dificultada pela inexistência de registros da seqüência dos recursos utilizados pelos participantes. Ainda

que os participantes tenham tentado registrar estas informações nos questionários, verificou-se que a gravação das ações dos usuários facilitaria a análise do perfil de uso do ambiente.

Os recursos disponíveis de navegação, zoom semântico e filtros foram bastante utilizados pelos participantes.

Em resumo, os resultados deste primeiro estudo mostraram que:

1. A não integração do SourceMiner aos recursos nativos da ADS dificultou a realização de parte das atividades solicitadas;
2. A metáfora visual do mapa em árvores não foi suficiente para a execução da última atividade solicitada;
3. A análise do perfil de uso do ambiente foi limitada pela não existência de monitoramento automático;
4. A utilização dos recursos de interação derivados da área de InfoVis foram bastantes úteis aos participantes.

3.2 SEGUNDO ESTUDO PRELIMINAR

No primeiro estudo não foi possível analisar a viabilidade do SourceMiner para o apoio efetivo a atividades não relacionadas à estrutura pacote classe método da aplicação. A atividade de identificação de anomalias de modularidade não foi executada por nenhum dos participantes do primeiro estudo. Por este motivo, foram realizadas modificações no SourceMiner conforme descrito a seguir e, em março de 2008, realizou-se um outro estudo de viabilidade da nova versão da ferramenta.

As atividades solicitadas aos participantes no segundo estudo preliminar tiveram como foco a verificação da forma como o SourceMiner auxiliava na execução de cinco atividades de manutenção originalmente executadas em (Ko, Myers, *et al.*, 2006). O objeto de estudo foi uma aplicação cujo objetivo é oferecer recursos para desenhar e editar figuras com cores em um editor gráfico. O segundo estudo preliminar foi publicado em (Carneiro, Magnavita, *et al.*, 2008) e é representado como **P5** na Figura 3.

3.2.1 Descrição da Versão da Ferramenta Utilizada no Estudo

Neste estudo foram feitas diversas modificações no SourceMiner, incluindo o seu uso como um plug-in do Eclipse. Esta mudança possibilitou o uso de recursos já disponibilizados pelo ADS no próprio **AIMV**. Por exemplo, não foi mais necessário o uso do *JavaCC* para a obtenção de dados necessários para a representação visual do código fonte. Para esta finalidade, o *Java Development Tool* (JDT) do próprio Eclipse foi utilizado. Além disso, nesta versão foi introduzido o conceito de perspectivas. Também houve a integração dos recursos de filtragem interativa, zoom semântico e geométrico para que fossem aplicados tanto no mapa em árvores como na visão polimétrica. Outra novidade desta versão foi a inclusão da funcionalidade de monitoração automática de operações primitivas com o objetivo de apoiar a análise mais detalhada das atividades executadas pelos usuários. A versão da ferramenta deste estudo corresponde à **etapa 2G** da Figura 3.

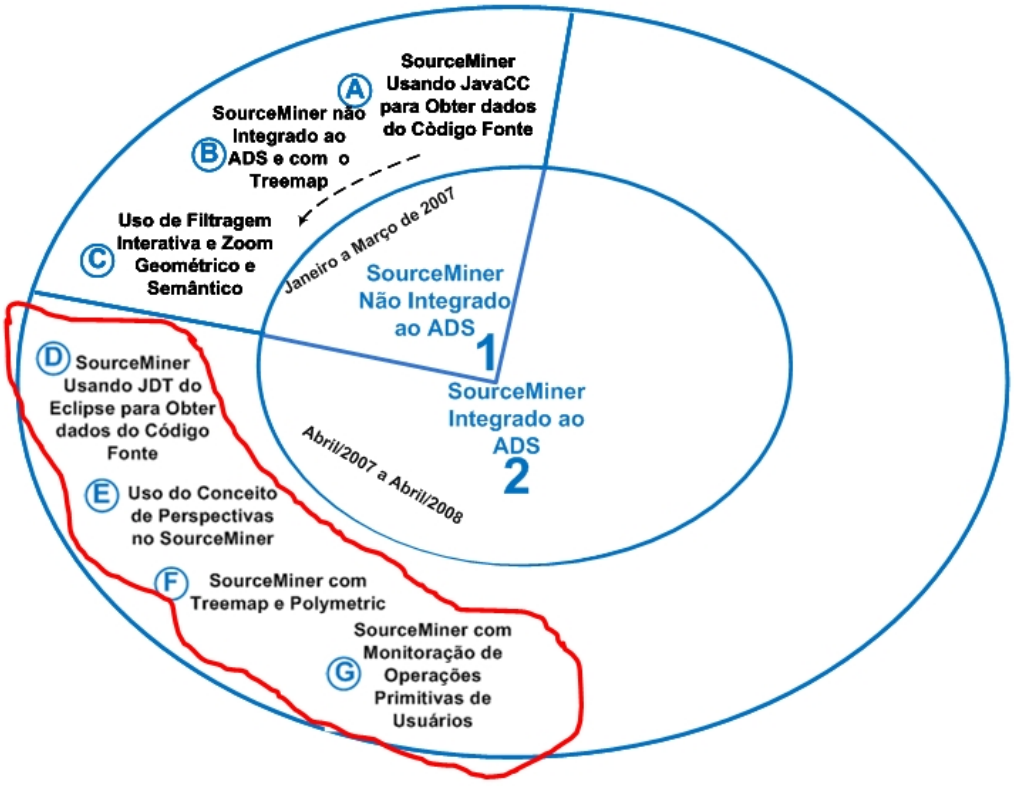

Figura 20 Versão Adotada no Segundo Estudo Preliminar

3.2.2 Planejamento do Estudo

O propósito deste estudo de viabilidade foi responder à questão: “O SourceMiner é um ambiente viável para apoiar a execução de atividades de compreensão de software, em especial aquelas que dependem de informações da estrutura pacote classe método e da hierarquia de herança da aplicação?”.

O objetivo deste estudo expresso segundo o gabarito GQM é:

Analisar o *SourceMiner, um ambiente de visualização interativo*

Com o propósito de *caracterizá-lo*

Em relação à *viabilidade no apoio às atividades de compreensão de software através das visões mapas em árvores e polimétrica*

Do ponto de vista *dos pesquisadores do ambiente SourceMiner*

No contexto de *atividades de compreensão de aplicações desenvolvidas em linguagem Java em um ambiente acadêmico in-vitro com participantes da indústria.*

Perguntas de Pesquisa. As perguntas de pesquisa (**PP**) selecionadas para este estudo são apresentadas a seguir.

Pergunta de Pesquisa 1 (PP1): Analisar o uso combinado do mapa em árvores, da visão polimétrica e de recursos de interatividade com o objetivo de avaliá-los com respeito à sua efetividade no apoio a atividades de manutenção de software.

Pergunta de Pesquisa 2 (PP2): Analisar até que ponto o uso das informações disponibilizadas pela monitoração automática das ações dos usuários apóia a identificação de estratégias e perfis de uso do ambiente.

O Objeto de Estudo. A aplicação objeto de estudo é uma aplicação chamada *Paint, a mesma utilizada por* Ko, Myers e colegas (2006). Ela utiliza Java *Swing* (Walrath, Campione, *et al.*, 2004) e é composta por nove classes com um total aproximado de 4700 linhas de código fonte. A aplicação oferece aos usuários uma janela para desenho e edição de figuras com cores.

Atividades do Estudo. As atividades de manutenção solicitadas aos participantes foram as mesmas utilizadas por Ko, Myers e colegas (2006) conforme descrito a seguir:

a) A barra de rolagem nem sempre aparece depois que se pinta fora da área reservada, mas quando aparece, desregula a aparência da tela da aplicação. Os participantes devem modificar a aplicação de tal forma que a barra de rolagem apareça imediatamente ao se pintar fora da área reservada da aplicação;
b) Os usuários não podem selecionar a cor amarela. Os participantes devem modificar o programa de tal forma que esta cor possa ser utilizada pelos usuários;
c) O botão de desfazer nem sempre funciona. Os participantes devem modificar a aplicação de tal forma que o botão desfaça a última ação ou limpe a tela da aplicação;
d) Há um botão disponível para desenhar com linha, mas não funciona. Os participantes devem incluir uma funcionalidade que possibilite aos usuários desenhar uma linha entre dois pontos;
e) Os participantes devem incluir um slider que regule a espessura da linha utilizada nos desenhos;

Foi solicitado que os participantes informassem nos questionários a experiência no uso de recursos Swing na linguagem Java, o horário de início e término de cada atividade, as estratégias adotadas para a resolução e os recursos do SourceMiner utilizados para cada uma. Estas informações foram utilizadas na etapa de análise juntamente com os registros obtidos do serviço de monitoração do SourceMiner. Pelo fato da atividade “a” ter sido simples, esperava-se que a maioria dos participantes estivesse apta a resolvê-la. Caso algum participante não executasse com sucesso esta atividade, ele seria excluído do experimento pelo fato de provavelmente não ter o perfil adequado para participação no estudo. As atividades de “b” até “e” foram consideradas como de complexidade média. Após o término das atividades, os participantes preencheram um formulário para registrar opinião a respeito do estudo, do uso do SourceMiner e da execução das atividades.

Participantes do Estudo. Os participantes do estudo foram profissionais da indústria matriculados em um curso de especialização em Engenharia de Software. As

atividades foram realizadas em equipes de três pessoas que trabalharam juntas para a execução das atividades. Nenhuma forma de compensação ou de atribuição de notas foi utilizada para participação no estudo.

3.2.3 Execução do Estudo

Todos os participantes utilizaram a aplicação *Paint* para executar as atividades e responder às perguntas do questionário. Todas as orientações foram realizadas pelo autor desta tese no laboratório antes do início das atividades.

O treinamento e a execução das atividades do estudo foram realizadas no intervalo de 3.5 horas. A sessão de treinamento durou 30 minutos com o objetivo de apresentar aos participantes as funcionalidades do SourceMiner e também descrever as atividades a serem executadas. As três horas restantes foram reservadas para a execução das atividades.

Este estudo foi composto por oito equipes de três pessoas. Todos informaram ter desenvolvido ou ter dado manutenção em aplicações Java com *Swing.*

3.2.4 Análise e Discussão dos Resultados

A análise das atividades executadas foi realizada através da leitura do código fonte fornecido por cada equipe. Além disso, a aplicação resultante de cada equipe foi executada para a verificação das funcionalidades solicitadas nas questões de “a” até “e”. O resultado foi que duas equipes não executaram a primeira atividade e, portanto, não tiveram seus dados considerados no estudo. Quatro equipes executaram somente as atividades “a”, “b” e “c” e duas equipes executaram todas as atividades. O próximo passo consistiu na análise das perguntas de pesquisa.

Pergunta de Pesquisa 1 (PP1)

Hipótese Nula 1: O uso combinado do mapa em árvores, da visão polimétrica e dos recursos de interatividade não são efetivos no apoio às atividades de manutenção de software.

Hipótese Alternativa 1: O uso combinado do mapa em árvores, da visão polimétrica e dos recursos de interatividade são efetivos no apoio às atividades de manutenção de software.

Neste estudo, o recurso de monitoração automática de atividades possibilitou a identificação tanto das visões como das seqüências de visões mais utilizadas. De forma similar ao registrado por Sillito, Volder e colegas (2005), constatou-se que os participantes dedicaram esforços na busca de informações no código fonte da aplicação analisada. Este esforço foi preponderante nas primeiras duas horas de execução das atividades. Neste período, o Package Explorer foi o mais utilizado com média aproximada de 63,83% de uso, seguido pelo Editor (21,28%), Search (6,38%), Outline (4,26%), Mapa em árvore (1,06%), Type Hierarchy (1,06%), Call Hierarchy (1,06%) e visão polimétrica (1,06%). Uma das duas equipes que executou as atividades de "a" até "e" de forma bem sucedida relatou o seguinte: "*inicialmente fizemos uma pesquisa no código fonte da aplicação para identificar os trechos que seriam de nosso interesse para executar as atividades.*" Os recursos de interatividade (filtros e zoom) foram pouco representativos para a execução das atividades, pois tiveram uso inferior a 0,5%.

O editor do ADS foi o recurso mais utilizado a partir da segunda hora do estudo. A justificativa para este fato foi a necessidade de modificação dos trechos de código fonte. O uso relativamente baixo do mapa em árvores, da visão polimétrica e dos recursos de interatividade oferecidos pelo SourceMiner não possibilitou a comprovação da **hipótese alternativa 1**. Segundo os participantes, "*apesar do mapa em árvores e da visão polimétrica representarem visualmente informações importantes da aplicação Paint, a equipe não conseguiu associar o seu uso com a execução das atividades solicitadas*". E ainda complementaram que "*na realidade procedemos como se estivéssemos utilizando o ADS em sua versão original*".

Pergunta de Pesquisa 2 (PP2)

Hipótese Nula 2: O uso das informações disponibilizadas pela monitoração automática das ações dos usuários não apóia a identificação de estratégias e perfis de uso do ambiente.

Hipótese Alternativa 2: O uso das informações disponibilizadas pela monitoração automática das ações dos usuários apóia a identificação de estratégias e perfis de uso do ambiente.

O registro de monitoração do SourceMiner disponibilizou fonte detalhada de informação a respeito das atividades executadas pelos participantes. Com ele podemos comprovar o que foi relatado pelos participantes em relação ao uso do SourceMiner. A análise dos registros não identificou nenhuma ligação direta entre o uso das visões do SourceMiner e a execução das atividades solicitadas no estudo, confirmando os relatos dos participantes apresentados anteriormente.

Ainda como resultado da análise dos registros, foi constatado o uso de busca intensa por trechos de código fonte. Conforme já apresentado na análise da pergunta de pesquisa 1 (PP1), as visões do ADS Eclipse utilizadas na busca são apresentadas a seguir com os respectivos valores aproximados de uso: Package Explorer (30%), Search (3%), Outline (2%), Type Hierarchy (0,5%) e Call Hierarchy (0,5%). Através dos resultados obtidos da análise dos registros foi possível constatar que a hipótese alternativa 2 é verdadeira.

Ameaças à Validade do Estudo. Pelo fato de se tratar de um estudo *in vitro*, existem limitações em relação à sua validade externa. O tamanho e a complexidade do objeto de estudo (aplicação *Paint*) são menores do que aplicações comerciais típicas. Entretanto, esta aplicação é compatível com o tempo de atividade disponível para uma sessão in-vitro. Além disso, ela já tinha sido utilizada por Ko, Myers e colegas (2006) em um estudo similar.

A característica das atividades é outra questão a ser analisada, tendo em vista que a maioria estava relacionada com funcionalidades relacionadas a interfaces gráficas, um cenário não abrangente de manutenções corretivas. Apesar disto, o conhecimento para a realização destas atividades é típico de muitas outras atividades de manutenção corretivas em sistemas orientados a objetos. Portanto, as estratégias adotadas pelos participantes podem ser generalizadas para outros tipos de atividades que não sejam necessariamente aquelas relacionadas a interfaces gráficas.

3.2.5 Lições Aprendidas

Os resultados deste segundo estudo preliminar indicaram que os recursos de monitoramento do SourceMiner apoiaram a análise das atividades executadas pelos participantes. O registro fornecido pelo SourceMiner das ações dos usuários foi útil para identificar a sequência de procedimentos adotada por cada equipe. Em compensação, a efetividade do apoio oferecido pelas visões do SourceMiner não foi constatada nem nos relatos dos participantes nem na análise dos registros. Os participantes relataram dificuldades em executar as atividades solicitadas através das informações representadas visualmente pelo SourceMiner. Foi identificada a necessidade de enriquecimento das metáforas visuais para que as mesmas tivessem algum tipo de vínculo com os requisitos da aplicação analisada. Esta necessidade ficou evidente na análise da hipótese alternativa 1. Um dos participantes mencionou que "*de alguma forma poderia ser incluída representação das funcionalidades da aplicação nas visões do SourceMiner*". Este mesmo participante complementou que "*durante a análise visual das classes não se sabia quais funcionalidades estas classes ofereciam*".

Na análise da hipótese 2 ficou evidente a necessidade de uma metodologia para a análise dos registros de eventos dos usuários. Em função do grande volume de dados, a inspeção manual dos registros requer parcela considerável de esforço e tempo. Uma solução encontrada foi de localizar nos registros o momento em que os participantes identificaram as classes e os métodos a serem modificados e a partir daí dividir em registros menores para análise. Apesar do esforço demandado, o registro automático realizado pelo SourceMiner mostrou-se como um recurso de baixo custo para a coleta de informações de uso da aplicação a exemplo do que tem sido feito na análise de perfil de uso de aplicações web (Kaushik, 2009)

Por último, os participantes relataram a necessidade de uma ou mais visões que representassem visualmente as relações de acoplamento entre os elementos de software. O argumento utilizado era a necessidade de poder ter uma visão abrangente de como classes identificadas como estratégicas para uma determinada atividade se relacionavam com as demais na aplicação. Também foi argumentado pelos participantes que os recursos de filtro poderiam ser disponibilizados para analisar somente um determinado tipo de relacionamento.

Em resumo, os resultados deste estudo mostraram que ainda seriam necessárias adaptações no SourceMiner e no seu conjunto de visões para dar um apoio mais efetivo às atividades de compreensão. Por outro lado, a análise dos registros de monitoração mostrou a possibilidade de identificação de estratégias de uso do ADS, fator importante para a caracterização do ambiente e definição de estratégias de sua utilização para compreensão de software.

3.3 TERCEIRO ESTUDO PRELIMINAR

O segundo estudo levantou a necessidade da representação de funcionalidades nas representações visuais. O uso de interesses mostrou-se uma idéia promissora para esta finalidade. Não somente pelo fato de ser uma inovação em relação a outras abordagens conhecidas na literatura (Robillard e Murphy, 2007) (Tarr, Ossher, *et al.*, 1999) (Cacho, Sant'Anna, *et al.*, 2006) (Garcia, Sant'Anna, *et al.*, 2005), mas por possibilitar a análise simultânea de espalhamento, dedicação e entrelaçamento dos interesses, tema de grande importância na atualidade.

Modificações foram implementadas no SourceMiner para suportar a análise visual de interesses, facilitando a configuração simultânea em todas as visões com diferentes decorações (métricas, características de entidades, e interesses), ficando a cargo do usuário a escolha e configuração do melhor cenário para a análise.

O último estudo da fase preliminar teve o objetivo de caracterizar e analisar o uso da representação visual de interesses (*concerns*) no SourceMiner. Ao longo do estudo foi analisado até que ponto o SourceMiner era efetivo para a análise de propriedades como espalhamento, entrelaçamento e dedicação e como estas propriedades poderiam apoiar atividades de compreensão de software. Este estudo foi publicado em (Carneiro, Sant´Anna, *et al.*, 2009), representado como **P8** na Figura 3.

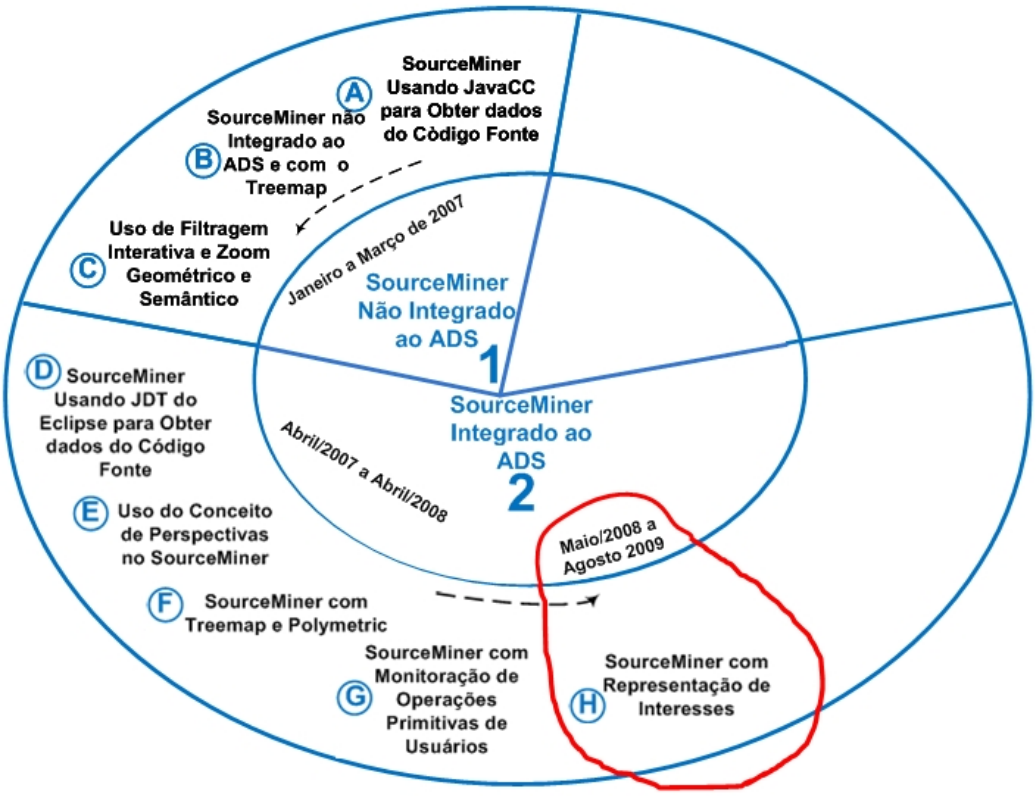

Figura 21 Versão Adotada no Terceiro Estudo Preliminar

3.3.1 Descrição da Versão do SourceMiner Utilizada no Estudo

A versão do SourceMiner utilizada neste estudo foi a da **etapa 2H** da Figura 3. Nesta versão, duas metáforas visuais (mapa em árvores e visão polimétrica) estavam disponíveis para representar os interesses de uma aplicação de forma simultânea, incluindo a possibilidade de seleção dos interesses a serem representados visualmente.

Conforme descrito na subseção 3.4.4, utilizamos o conceito de decoração para tratar da representação visual dos interesses. Neste caso, um elemento visual é preenchido com uma cor se ele implementar um dado interesse. Por exemplo, elementos visuais tais como retângulos nos mapas em árvores ou retângulos arredondados na visão polimétrica são

coloridos se aqueles métodos ou classes, respectivamente, forem afetados por um dado interesse.

Cada interesse mapeado é associado a uma cor diferente. Programadores podem selecionar tanto a cor para representar um interesse, como o(s) interesses a serem representados visualmente, conforme exemplificou a Figura 12 da subseção 2.4.1.

3.3.2 Planejamento do Estudo

O propósito deste estudo de viabilidade foi responder à seguinte questão: “O SourceMiner é um ambiente viável para apoiar a execução de atividades de compreensão de software, em especial aquelas que dependem da representação visual de interesses nas visões do ambiente?”

O objetivo deste estudo expresso segundo o gabarito GQM é
Analisar o SourceMiner, um ambiente interativo
Com o propósito de caracterizá-lo
Em relação à viabilidade no apoio às atividades de compreensão de software através da representação visual de interesses
Do ponto de vista dos pesquisadores do ambiente SourceMiner
No contexto de atividades de compreensão de aplicações desenvolvidas usando a linguagem Java em um ambiente acadêmico in-vitro.

Perguntas de Pesquisa. As perguntas de pesquisa (**PP**) selecionadas para este estudo são apresentadas a seguir.

Pergunta de Pesquisa 1 (PP1): Analisar o uso das visões polimétricas com a representação visual de interesses, com o objetivo de avaliá-los com respeito à sua efetividade no apoio a atividades de avaliação de modularidade de interesses.

Pergunta de Pesquisa 2 (PP2): Analisar o uso dos mapas em árvores com a representação visual de interesses, com o objetivo de avaliá-los com respeito à sua efetividade no apoio a atividades de avaliação de modularidade de interesses.

O Objeto de Estudo. A aplicação objeto do estudo selecionada foi a MobileMedia (MobileMedia, 2006). Ela pode ser utilizada para manipular fotos, áudio e vídeo em dispositivos móveis tais como telefones celulares. Os interesses identificados foram *photo*, *music*, *video*, *album*, *create/delete media*, *label media*, *view/play media*, *sort media*, *copy media* e *set favorites*. Pelo fato da aplicação ter sido implementada usando o padrão *Model-View-Controller* (**MVC**) (Krasner e Pope, 1988), o interesse controller também foi identificado no sistema. Outros interesses relacionados a requisitos não funcionais tais como tratamento de exceções e persistência também foram identificados.

Várias foram as razões para a escolha do MobileMedia. A primeira delas é que a aplicação foi desenvolvida na linguagem Java e seu código fonte está disponível na web. A segunda é que a aplicação tem servido de objeto de estudo para a avaliação de várias técnicas de avaliação de modularidade (Figueiredo, Silva, *et al.*, 2009; Figueiredo, Cacho, *et al.*, 2008). A terceira razão é que o mapeamento de interesses relevantes da aplicação já tinha sido realizado previamente por outros pesquisadores (Figueiredo, Cacho, *et al.*, 2008) e estava disponível para o autor desta tese. E, finalmente, a modularidade dos seus interesses também já tinha sido avaliada através de métricas sensíveis a interesses e outras estratégias de detecção (Storey, 2006; Shneiderman e Plaisant, 2010).

Atividades do Estudo. Os participantes deveriam visualmente executar a análise de modularidade dos interesses previamente mapeados na aplicação selecionada para avaliar suas propriedades de espalhamento, dedicação e entrelaçamento. A análise da modularidade dos interesses seria realizada em conjunto pelos participantes utilizando o SourceMiner, sem acesso ao código fonte. O objetivo não era avaliar, mas caracterizar como os participantes executariam as atividades solicitadas (Wohlin, Runeson, *et al.*, 2000).

Participantes do Estudo. Os participantes deste estudo foram o autor desta tese juntamente com um especialista em métricas orientadas a aspecto e interesses.

3.3.3 Execução do Estudo

Para que fossem representados, os interesses foram primeiro assinalados manualmente no código fonte por especialistas da aplicação sob análise usando o plug-in *ConcernMapper* do Eclipse (Robillard e Weigand-Warr, 2005). O resultado da atividade foi um arquivo *XML* que foi então lido pelo SourceMiner para a análise. Após esta leitura, o ambiente ficou pronto para apresentar visualmente os interesses mapeados na aplicação.

Três interesses da aplicação MobileMedia foram selecionados para análise visual: *controller*, *album* e *persistence*. Estes interesses foram selecionados com base na sua representatividade: (i) *album* está relacionado ao domínio do negócio da aplicação, (ii) *persistence* representa um requisito não funcional, e (iii) *controller* é um dos três pilares do padrão arquitetural sob o qual a aplicação foi projetada.

Cada interesse foi analisado através da decoração das visões mapa em árvores e polimétrica do SourceMiner com base nos seguintes atributos: (i) espalhamento – o grau com que o interesse está espalhado ao longo dos diferentes módulos da aplicação; (ii) dedicação – quanto de cada unidade de modularidade está afetado pelo interesse, e (iii) entrelaçamento – o grau com que um interesse coexiste com outros em uma unidade de modularidade. Os participantes executaram as atividades deste estudo no intervalo de duas horas. Os resultados do estudo foram registrados em um formulário utilizado para descrever como ocorreu a análise visual da modularidade dos interesses.

3.3.4 Análise e Discussão dos Resultados

Pergunta de Pesquisa 1 (PP1)

Hipótese Nula 1: O uso da representação de interesses na visão polimétrica não apóia a execução de atividades de avaliação de modularidade de interesses.

Hipótese Alternativa 1: O uso da representação de interesses na visão polimétrica apóia a execução de atividades de avaliação de modularidade de interesses.

A parte A da Figura 22, Figura 23 e Figura 24 ilustram como os interesses podem ser representados na visão polimétrica. Os retângulos foram coloridos em verde claro, azul escuro e vermelho para tal finalidade. Neste caso, tem-se a representação de classes ou interfaces que são afetadas pelo interesse selecionado naquele instante. Usando esta visão é possível analisar de forma interativa as propriedades de **espalhamento**, **entrelaçamento** e **dedicação** em termos na hierarquia de herança da aplicação. Os parágrafos a seguir descreverão exemplos da análise de cada uma destas propriedades.

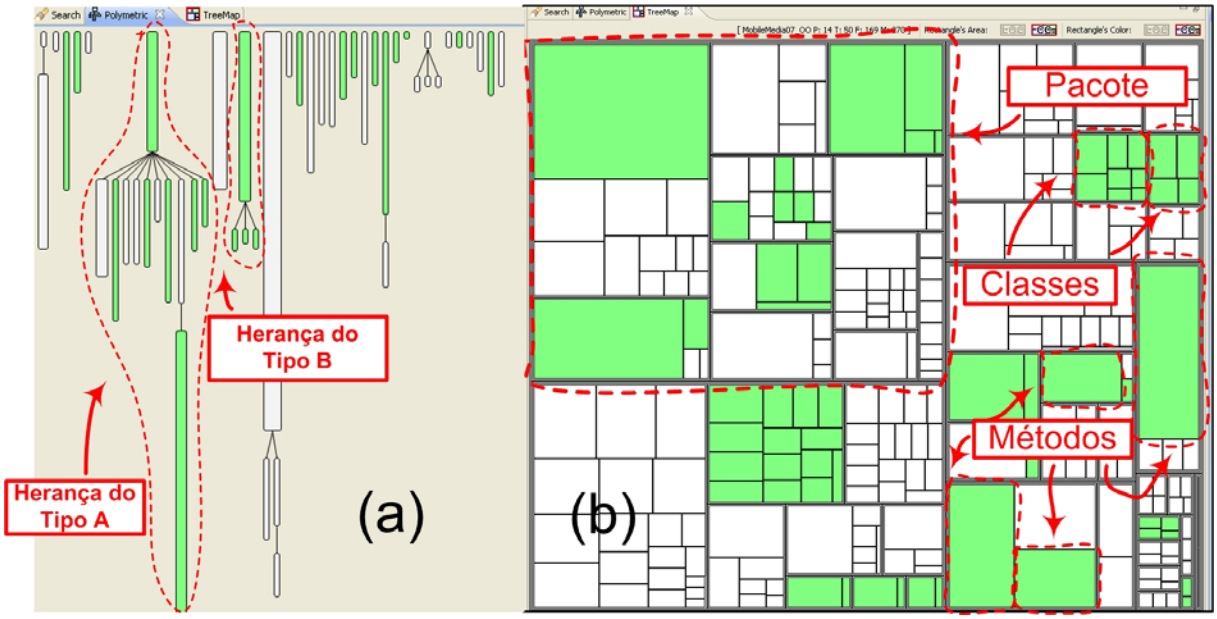

Figura 22 Visão Polimétrica e Mapa em Árvores (Interesse I)

O grau de **espalhamento** de um interesse está relacionado ao número de árvores de herança que apresentam a cor selecionada para o interesse. Uma cor manifestada em várias árvores indica que o mesmo interesse afetou pelo menos uma classe ou interface em cada uma destas árvores. A Figura 22 ilustra que o interesse representado em verde claro está espalhado em 13 árvores. Um interesse espalhado por um grande número de árvores pode ser um indicativo de problemas de modularidade (Greenwood, Bartolomei, *et al.*, 2007; Conejero, Figueiredo, *et al.*, 2009).

A **dedicação** de uma árvore de herança a um interesse é obtida pela proporção da árvore que é colorida pela cor correspondente ao interesse selecionado. A árvore que tem todos os retângulos preenchidos por uma determinada cor indica que no mínimo um método de todas as classes e interfaces que compõem a árvore está dedicado ao interesse. Por outro lado, árvores com baixa dedicação a um interesse específico têm poucos retângulos coloridos. Isto pode ser uma indicação de que a implementação deste interesse não faz parte do objetivo principal desta árvore (Figueiredo, Silva, *et al.,* 2009). Comparando-se a herança do tipo A com a do tipo B na Figura 22, é possível verificar como duas árvores podem ter diferentes graus de dedicação ao mesmo interesse. De forma similar, comparando-se a herança do tipo A da Figura 22 e da Figura 23, pode-se verificar como a mesma árvore pode ter diferentes graus de dedicação a interesses distintos.

O número de cores distintas manifestadas em uma árvore de herança revela ainda o nível de **entrelaçamento** entre os interesses. Uma árvore que apresente várias cores indica que ela lida com vários interesses. Isto pode ser uma indicação de que a árvore pode estar susceptível a diferentes tipos de mudanças em função de qualquer um destes interesses.

A facilidade com que estas análises foram feitas é uma evidência que o uso da visão polimétrica, associada à representação visual de interesses, apóia a execução de atividades de avaliação de modularidade de interesses. A análise do espalhamento, dedicação e entrelaçamento dos interesses descrita nos parágrafos acima suportam qualitativamente a aprovação da hipótese alternativa 1.

Hipótese Nula 2: O uso de mapas em árvores associado à representação visual de interesses não apóia a execução de atividades de avaliação de modularidade de interesses.

Hipótese Alternativa 2: O uso de mapas em árvores associado à representação visual de interesses apóia a execução de atividades de avaliação de modularidade de interesses.

As partes B da Figura 22, Figura 23 e Figura 24 ilustram como os interesses podem ser representados na visão de mapa em árvores. Usando esta visão, os usuários podem analisar de forma interativa as ocorrências de **espalhamento**, **entrelaçamento** e **dedicação** em termos da estrutura pacote-classe-método.

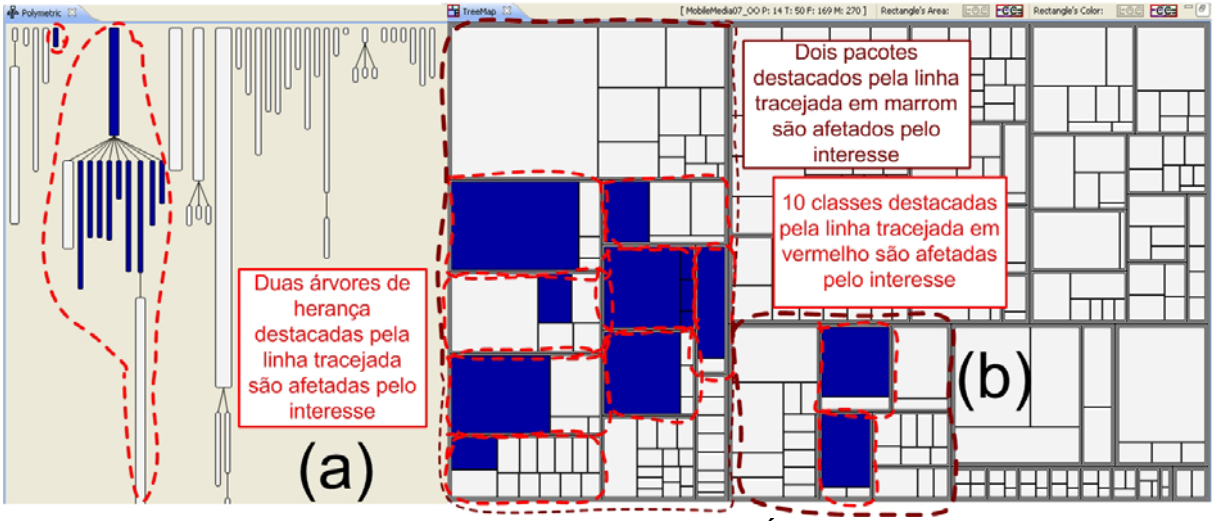

Figura 23 Visão Polimétrica e Mapa em Árvores (Interesse II)

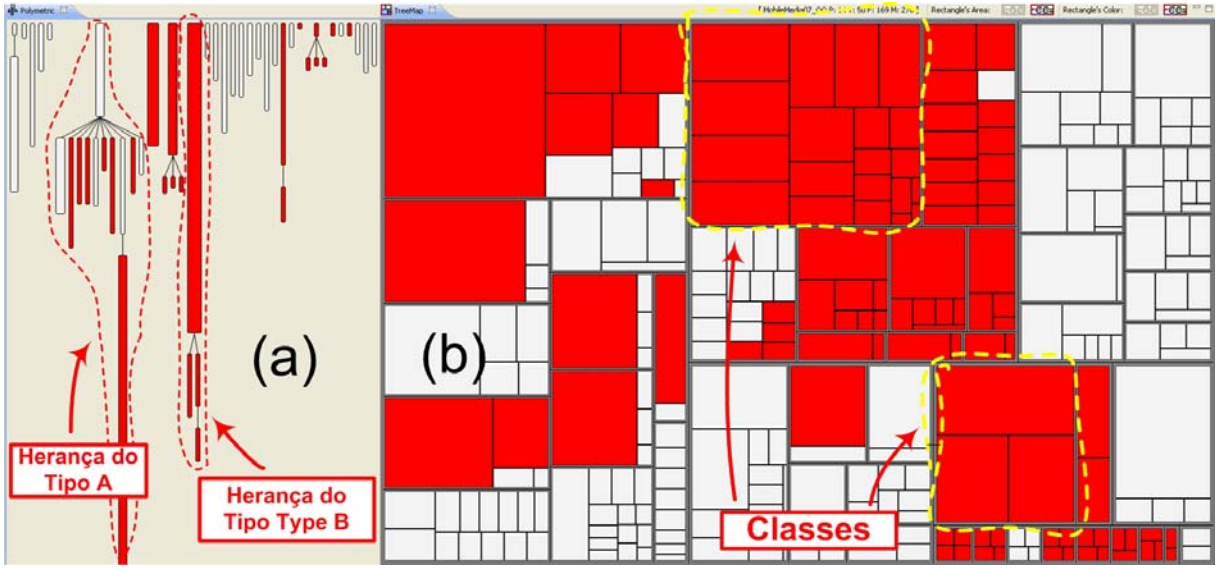

Figura 24 Visão Polimétrica e Mapa em Árvores (Interesse III)

Nos mapas em árvores, o grau de espalhamento de um interesse é relacionado ao número de pacotes que apresentam a cor do interesse. A cor quando manifestada em vários pacotes indica que o mesmo interesse afetou pelo menos uma classe em cada pacote. Um interesse quando espalhado por vários pacotes pode ser um indicador de a sua modularidade deve ser revista.

A **dedicação** de um pacote a um interesse é manifestada quanto todas, ou quase todas as suas classes, são afetadas pelo interesse. Um pacote tem todas as classes dedicadas a

um interesse significa que pelo menos um método da classe é preenchido pela cor do interesse. De forma análoga, uma classe que tem todos os seus métodos dedicados a um interesse tem todos os seus métodos preenchidos com a cor correspondente do interesse.

De modo inverso, pacotes ou classes apresentam baixa dedicação a um interesse se têm poucas classes, ou métodos coloridos pela cor do interesse. Esta pode ser uma indicação de que a realização deste interesse não é o objetivo principal deste pacote ou classe.

O número de cores distintas manifestada em um pacote revela o grau de **entrelaçamento** de interesses. Um pacote que é decorado por várias cores revela que lida com vários interesses. Isto pode ser uma indicação de que este pacote pode estar susceptível a mudanças de forma proporcional ao número de interesses que o afetam. De forma análoga, uma classe que apresenta várias cores indica que ela lida com vários interesses. Isto pode ser uma indicação de que a classe pode ser susceptível a várias mudanças em função de suas múltiplas dedicações.

A Figura 22 mostra o interesse *album*. Através da visão Mapa em árvores (parte B) foi possível verificar que este interesse está **espalhado** por vários métodos, classes e pacotes. Algumas classes têm inclusive todos os métodos afetados por ele. Pesquisando o nome destas classes através das visualizações (posicionando-se o mouse sobre os elementos visuais são apresentados os nomes dos elementos de software representados), foi possível observar que o objetivo principal destas classes era a implementação da funcionalidade relacionada ao interesse *album*. A visão mapa em árvores também é efetiva para indicar que algumas classes têm alguns de seus métodos dedicados ao interesse *album*. Estes métodos e conseqüentemente suas classes utilizam serviços relacionados ao interesse *album*. Nestes casos, este interesse está geralmente entrelaçado com outros.

As visões, e análises visuais, também podem ser facilmente combinadas. Considerando o mesmo interesse *album* através da visão polimétrica, contida na parte A da Figura 22, foi possível observar, por exemplo, que a informação disponibilizada visualmente pela visão polimétrica corrobora aquela apresentada pela visão mapa em árvores. O interesse é espalhado por várias classes. A questão é que agora se vê que estas classes são elementos de diferentes árvores de herança.

Verificou-se que a estrutura de herança da aplicação MobileMedia foi projetada de forma compatível com o padrão **MVC**. Como resultado, existem três árvores de herança,

cada uma implementando um dos papéis do padrão **MVC**. Com a análise visual, pode-se verificar que o interesse *album* estava espalhado por diferentes árvores, pois uma funcionalidade relacionada ao negócio estava afetando os três papéis do **MVC**. Pode-se verificar também que existe uma árvore inteiramente dedicada a este interesse. Isto não foi surpresa, tendo em vista que os três tipos de mídia com os quais a aplicação trabalha (foto, áudio e vídeo) são organizadas em álbuns.

A visualização do interesse *persistence* é apresentada na Figura 24. De forma análoga ao interesse *album*, a visão mapa em árvores mostra o interesse *persistence* espalhado por vários pacotes e classes – parte B da Figura 24. A diferença é que existe maior número de classes com todos os métodos dedicados ao interesse *persistence*. Ao analisar a visão polimétrica, parte A da Figura 24, observa-se que o interesse *persistence* afeta várias árvores, incluindo aquelas relacionadas ao "modelo", "visão" e "controle" do padrão **MVC**. Para analisar melhor esta questão, acessou-se o código fonte e verificou-se que estas classes tratavam exceções específicas de persistência que eram propagadas das classes relacionadas ao "modelo" do padrão **MVC**.

Finalmente, a Figura 23 mostra as visões mapa em árvores e visão polimétrica apresentando o interesse *controller*. Diferentemente dos outros dois interesses, o interesse *controller* está com boa modularidade, concentrando-se em alguns poucos pacotes e classes. A análise da visão polimétrica indica que o interesse está quase que restrito a uma árvore de herança que está dedicada ao papel de controle no escopo do padrão **MVC**.

Pode-se concluir que estes resultados apóiam a hipótese alternativa 2 de que o uso da visão mapas em árvores associada à representação visual de interesses apóia a execução de atividades de avaliação de modularidade de interesses e conseqüentemente rejeitam a hipótese nula 2.

Ameaças à Validade do Estudo. Da mesma forma que mencionado no estudo anterior, o fato de se tratar de um estudo *in vitro* traz limitações em relação à validade externa. O tamanho e a complexidade do objeto de estudo (aplicação *MobileMedia*) são mais realísticos, mas o estudo foi realizado apenas com uma dupla. Além disso, há um risco de viés, pois o autor da tese participou da atividade de análise. Esta ameaça de conclusão é parcialmente mitigada pelo fato da análise de modularidade de interesses ter sido compartilhada com um perito no tema, que nunca antes tinha usado visualização para esta

finalidade. Desta forma, argumenta-se que o estudo foi efetivo para ilustrar o potencial das metáforas visuais disponibilizadas pelo SourceMiner para a análise das propriedades de dedicação, entrelaçamento e espalhamento.

3.3.5 Lições Aprendidas

Este estudo mostrou que a inclusão de representações visuais de interesses no SourceMiner é útil e torna mais efetivas as atividades de compreensão da modularidade do software. Através da interação com as abstrações visuais da aplicação, possibilitou-se a análise sensível a interesses através de duas visões. Estas visões, por sua vez, representam perspectivas diferentes: a estrutura pacote-classe-método e hierarquia de herança. Este foi um primeiro passo para avaliar o uso de ambientes visuais interativos baseados em múltiplas perspectivas para a análise da modularidade de interesses e proporcionar aos usuários informações a respeito de requisitos funcionais e não funcionais da aplicação.

Apesar das vantagens oferecidas pelo ambiente, outras oportunidades de melhoria foram identificadas durante o estudo. A primeira limitação é o fato do menor nível de mapeamento de interesses serem os níveis de método e atributo. Não é possível mapear os interesses até o nível de linha de código utilizando o ConcernMapper (Robillard e Weigand-Warr, 2005), fato que poderia tornar mais detalhada a análise. Outra limitação é que a análise da modularidade é depende dos interesses previamente mapeados através do ConcernMapper.

Também foi novamente identificada a necessidade de representações visuais para apresentar relacionamentos de dependências entre os elementos de software. Estas representações visuais complementariam, através de outra perspectiva, as atividades de compreensão de software.

3.4 CONCLUSÃO DO CAPÍTULO

Os estudos preliminares foram decisivos para a definição incremental do modelo conceitual do ambiente de visualização de software proposto. Este modelo inclui vários pontos importantes derivados dos estudos realizados. O primeiro deles é o conceito de perspectiva como um conjunto de visões coordenadas que representam propriedades específicas de uma situação, contexto ou entidade, geralmente utilizando diversas metáforas visuais. No SourceMiner, cada perspectiva define uma estrutura de dados a partir da qual novas visões podem ser construídas.

Outro ponto importante é o uso intensivo de recursos de interatividade para permitir ao usuário configurar o cenário visual mais adequado às suas necessidades em um dado instante. Neste contexto, tanto o zoom semântico como o geométrico podem ser utilizados para dar foco aos elementos de software em análise em um determinado momento. Além disso, os filtros baseados em dados categóricos e numéricos também podem ser utilizados para que sejam destacados no cenário visual os elementos que atendem a um conjunto de critérios informado pelo usuário.

O terceiro ponto é a necessidade de integração transparente. A integração do SourceMiner ao ADS Eclipse é um importante avanço conceitual, pois possibilita a extração de dados diretamente da Árvore Sintática Abstrata (AST) disponibilizada pelo ADS. A integração entre as visões do SourceMiner e destas com o ADS através dos recursos disponibilizados pelo *Java Development Tool* (JDT) do Eclipse facilita sua utilização e evolução.

O quarto ponto é a representação visual de atributos de interesse dos elementos de software através do conceito de decoração consistente das visões. No caso do SourceMiner, a representação de interesses através de cores trouxe a possibilidade da análise simultânea de importantes propriedades de modularidade do software tais como entrelaçamento, espalhamento e dedicação de interesses. Rapidamente ficou claro que outros atributos, tais como *churning* ou número de ocorrências de defeitos em elementos de software, poderiam ser representadas de forma similar nas decorações das visões do SourceMiner.

O último ponto é o suporte à experimentação. O registro das ações dos usuários no uso do ambiente mostrou-se promissor para fornecer evidências de estratégias utilizadas nas atividades de compreensão de software.

Todos estes pontos foram reunidos em um modelo conceitual e um conjunto de visões que são discutidos detalhadamente no próximo capítulo.

Capítulo

4

Este capítulo apresenta o modelo conceitual do SourceMiner refinado a partir dos resultados obtidos dos estudos preliminares apresentados no capítulo anterior. Em seguida são apresentados o modelo conceitual, e as perspectivas e visões que compõem o SourceMiner.

4 O MODELO CONCEITUAL PROPOSTO E AS VISÕES DO SOURCEMINER

Os estudos preliminares anteriores apresentaram a evolução do SourceMiner como um Ambiente Interativo baseado em Múltiplas Visões (**AIMV).** Este capítulo apresenta o modelo conceitual do SourceMiner como um **AIMV** interativo, extensível e composto por visões coordenadas. O modelo é baseado em múltiplas perspectivas que oferecem um conjunto de visões que permitem a análise do software sob diferentes pontos de vista. É coordenado porque suas visões são sincronizadas e respondem às ações dos usuários de forma consistente. É interativo porque os usuários podem configurar dinamicamente o cenário visual mais adequado à construção de seus modelos mentais. É extensível porque sua arquitetura foi concebida para facilitar a inclusão de novas visões ao ambiente.

O modelo conceitual proposto reflete as características citadas no parágrafo anterior. Este modelo, apresentado na Figura 25, foi concebido e refinado ao longo dos estudos preliminares. A parte **A** da figura indica de forma metafórica que o código fonte (retângulo em azul) não é suficiente para tornar as atividades de compreensão (octógono vazado) efetivas. Por este motivo é realizada a extração de informação do modelo Java e da árvore sintática abstrata disponibilizada pelo ADS Eclipse. A partir desta extração também são computadas métricas cujos valores enriquecerão as visões disponibilizadas no ambiente (parte **C)**. A interação com os cenários visuais é viabilizada através dos filtros de controle (parte **D)**. A parte **B** indica que o modelo é baseado em perspectivas que são um conjunto de visões coordenadas que representam um grupo de propriedades específicas do software

analisado (por exemplo, as partes **E**, **F**, **G** e **H**). O cenário visual resultante é um conjunto de visões configurado em um dado instante para a execução de uma tarefa específica de compreensão de software (parte **I**). No decorrer deste capítulo este modelo será apresentado em mais detalhes.

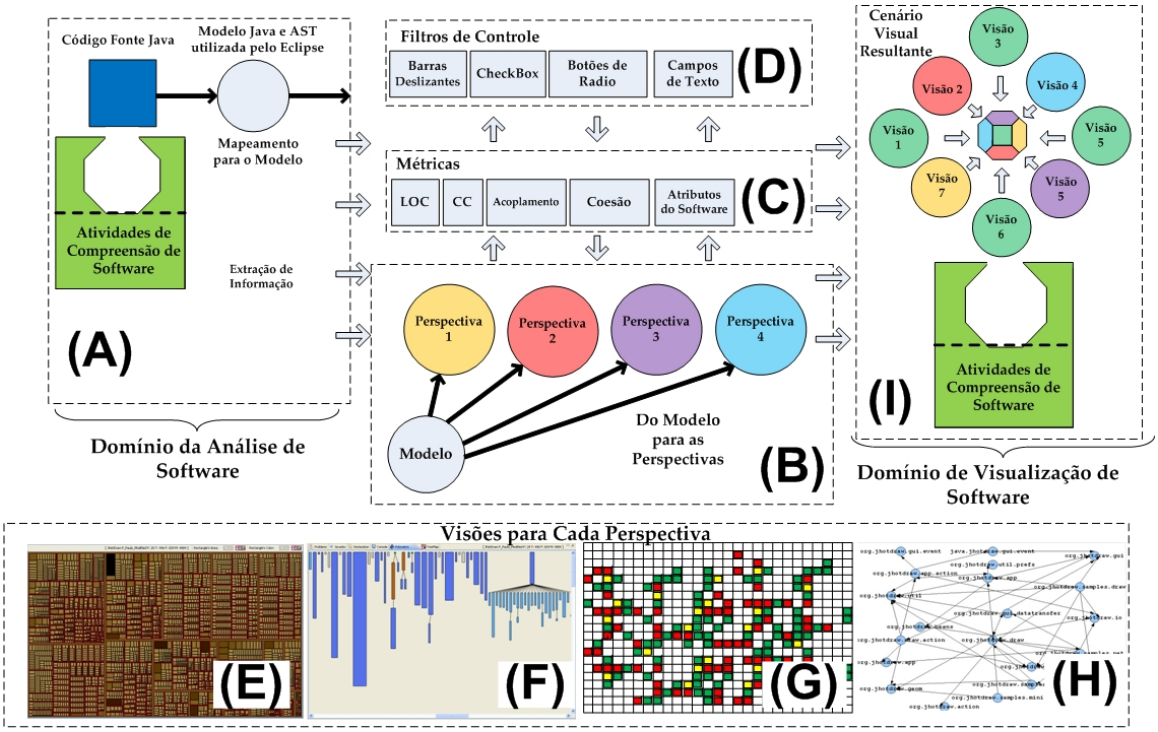

Figura 25 O Modelo Conceitual Proposto

4.1 DETALHANDO O MODELO PROPOSTO

Muitos projetos de visualização de software têm sido concebidos como sistemas isolados, não sendo, portanto, integrados a ambientes de desenvolvimento de software. Entretanto, uma premissa importante para este trabalho é o fato do ADS ser um substrato apropriado para ambientes como o SourceMiner. A integração do **AIMV** ao ADS é um caminho natural para apoiar as atividades de compreensão de software.

Os ADSs atuais já oferecem diversos recursos para apoiar a compreensão de software. A maioria deles oferece pelo menos um editor direcionado à sintaxe que utilizam recursos de *pretty printing*, como também algum tipo de representação hierárquica da estrutura do projeto sob análise. Geralmente, várias outras visões disponibilizam informações relevantes para os programadores, representando sistemas de software de diferentes formas (por exemplo, o Package Explorer e o Outline do Eclipse). Os ADSs também disponibilizam diferentes formas de pesquisa e navegação.

Uma conseqüência natural do uso de um ADS como substrato de um **AIMV** é, portanto, o fato dos programadores poderem, de forma alternada e em um mesmo ambiente, acessarem o código fonte, assim como as visões originalmente disponibilizadas pelo ADS, em conjunto com aquelas disponibilizadas pelo **AIMV**.

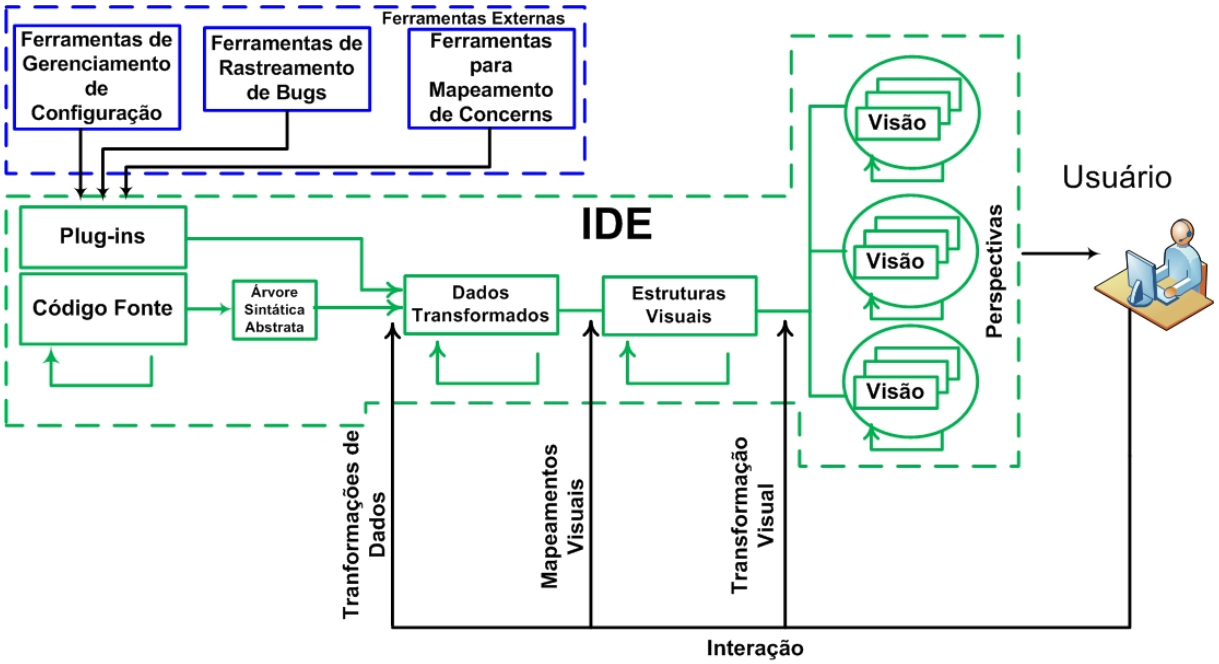

Figura 26 Um Modelo de Referência para SoftVis

A Figura 26 ilustra como o modelo de Card foi adaptado para o domínio de visualização de software. O objetivo é proporcionar um conjunto de visões, coordenadas e sincronizadas entre si, integradas ao ADS. De forma análoga ao modelo apresentado nas Figuras 13 e 14, a Figura 26 também tem três níveis de interação: transformação de dados, mapeamento visual e transformação visual.

O modelo enfatiza que o ADS é o principal fornecedor de dados da aplicação a ser analisada. Os dados disponibilizados pelo ADS são acessados, transformados, mapeados para as estruturas visuais e renderizados nas visões. Os ADSs permitem a extração de informação útil do código através de seus recursos nativos tais como a sua Árvore Sintática Abstrata (AST) (Clayberg e Rubel, 2009).

Informações adicionais, tais como mapeamento de interesses, histórico de modificações e defeitos, também podem ser capturadas de fontes externas de dados e utilizadas para enriquecer os cenários visuais (Carneiro, Sant´Anna e Mendonça, 2010; Carneiro, Silva, *et al.*, 2010). Isto abre a possibilidade de uso do modelo concebido para incluir nas metáforas visuais dados complementares àqueles oferecidos pela árvore sintática da aplicação analisada. O ambiente permite a inclusão futura de novas fontes de dados cujas representações visuais favoreçam a compreensão do software. Vide parte superior esquerda da Figura 26.

Múltiplas visões são usadas para representar diferentes propriedades do software. Por exemplo, uma visão pode ser concebida para a representação da hierarquia de herança entre os módulos enquanto outra pode ser dedicada à representação visual do acoplamento entre eles. Como discutido no Capítulo 2, diferentes metáforas visuais podem ser usadas para a mesma propriedade. Por exemplo, o acoplamento entre módulos pode ser representado através de grafos interativos ou matrizes de relacionamento. Neste caso, cada representação deve enfatizar diferentes aspectos da propriedade analisada ou deve oferecer novos mecanismos para facilitar a interpretação visual dos dados representados. Neste trabalho, será utilizado o termo *visualização baseada em múltiplas formas* para se referir a diferentes visões (formas) sendo usadas para representar mesma propriedade do software.

Conforme mencionado anteriormente, múltiplas visões devem ser coordenadas de forma que uma ação em uma visão seja refletida em todas as outras visões do ambiente. Por este motivo, algumas vezes utilizamos a expressão *múltiplas visões coordenadas* – no lugar de simplesmente múltiplas visões (Roberts, 2007) – para enfatizar este fato.

O modelo da Figura 26 realça a coordenação entre as visões pelas setas verdes de retorno no lado direito da figura.

O uso de múltiplas visões coordenadas e visualizações baseadas em múltiplas formas são recursos apropriados para apoiar a exploração de espaços complexos de dados

(Wu e Storey, 2000) (Graham e Kennedy, 2008). A idéia de se ter múltiplas visões coordenadas é um esforço para se combinar visualmente diferentes aspectos dos dados analisados em diferentes visões (Becks e Seeling, 2004). Múltiplas visões apóiam diferentes perspectivas e níveis de abstração e, quando bem configuradas, reduzem o volume de cognição e interpretação necessária para a execução de atividades recorrentes em engenharia de software, (Koschke, 2003; Storey, 2006).

4.2 PERSPECTIVAS E VISÕES NO SOURCEMINER

Os estudos conduzidos no capítulo anterior mostraram a importância do uso das perspectivas. A versão atual do SourceMiner tem três perspectivas: (i) a perspectiva pacote classe método (PCM); (ii) a perspectiva de hierarquia de herança e (iii) a perspectiva de acoplamento. As perspectivas têm o objetivo de agrupar conjuntos de visões que representam o mesmo conjunto de propriedades de software. O objetivo das perspectivas é que, quando utilizadas de forma combinada, possam proporcionar uma gama mais ampla de recursos e dados relacionados ao software se comparado ao uso de cada uma de forma isolada.

Diversas metáforas visuais foram analisadas para uso nas perspectivas mencionadas. A Tabela 3 apresenta as metáforas visuais adotadas na versão atual do SourceMiner: (i) mapa em árvores (Shneiderman, 1992) para a perspectiva pacote-classe-método; (ii) visão polimétrica (Lanza e Ducasse, 2003) para a perspectiva de hierarquia de herança; e (iii) um conjunto de três metáforas visuais para a perspectiva de acoplamento – grafos de dependências de classes e pacotes (Battista, Eades, *et al.*, 1994), visão tabular para representar classes, grafos espirais egocêntricos para representar classes, e matrizes de relacionamento para representar métodos, classes e pacotes.

Todas as visões do SourceMiner foram implementadas a partir do zero. Duas delas, a visão tabular e o grafo espiral egocêntrico são contribuições novas do nosso trabalho para a área de visualização de informações, em geral, e visualização de software, em

particular (Carneiro, Nunes, *et al.*, 2010). As demais, mapa em árvores (Shneiderman, 1992), visão polimétrica (Lanza e Ducasse, 2003), grafos (Battista, Eades, *et al.*, 1994) e matrizes de relacionamento (Sangal, Jordan, *et al.*, 2005), foram propostas por outros pesquisadores, mas foram completamente re-implementadas no contexto desta tese para as necessidades do SourceMiner.

4.2.1 A perspectiva pacote classe método (PCM)

A informação estrutural relacionada à estrutura pacote-classe-método tem um papel importante nas atividades de compreensão de software (Storey, 2006). A maioria dos ADSs oferece pelo menos uma visão que apresenta este tipo de estrutura. Um exemplo é o Package Explorer do ADS Eclipse.

Esta metáfora utiliza uma estrutura hierárquica em árvore vertical que permite esconder ou expandir cada um de seus nós. Estes nós utilizam ícones para denotar o tipo de elemento que eles representam. Como qualquer outra, esta metáfora tem limitações. Ela não escala bem, e em ADS, geralmente, se limita a apresentar somente a estrutura do projeto. Além disso, possui limitações em relação à sua efetividade de uso, conforme mencionado no capítulo 1 desta tese. Claramente, há espaço para enriquecer esta metáfora visual com novos atributos visuais tais como cores, posicionamento e tamanho dos elementos visuais. Estes atributos podem ser usados para representar importantes propriedades do software tais como tamanho, versões, modificações executadas, entre outros.

Tabela 3 Perspectivas e suas respectivas Metáforas Visuais

Grupos de Visões ou Perspectivas	**Visões**
Estrutura Pacote-Classe-Método (PCM)	Mapa em árvore (1)
Hierarquia de Herança (HH)	Visão Polimétrica (2)
Acoplamento (ACO)	Grafos de Dependência de Pacotes (3) e Classes (4); Visão Tabular (5); Grafos Espirais Egocêntricos (6); Matrizes de Relacionamento de Pacotes (7), Classes (8) e Métodos (9).

Decidimos utilizar uma metáfora visual alternativa para representar a estrutura PCM no SourceMiner. Uma visão baseada nesta metáfora deveria escalar bem, suportar a fácil inclusão dos atributos visuais mencionados anteriormente, e ajudar a contrapor a visão PCM tradicional disponibilizada pelo ADS. A metáfora visual *mapas em árvores* foi selecionada para isto.

Mapas em árvores são visualizações 2D que mapeiam uma estrutura em árvore usando retângulos aninhados de forma recursiva (Shneiderman, 1992). Mapas em árvores são bastante efetivos na representação de grandes hierarquias. Além da hierarquia propriamente dita, também representam outros atributos de dados usando o tamanho (área) do retângulo e as suas cores.

Mapas em árvores são efetivos na representação de grandes hierarquias, pois cada retângulo pode ser facilmente dimensionado para utilizar toda área que lhe é disponibilizada na tela do computador. Como cada retângulo corresponde a um nó da hierarquia PCM, um mapa em árvore pode visualmente representar simultaneamente milhares de nós de uma mesma estrutura. Pela sua simplicidade, esta estrutura também facilita a descoberta de padrões e a identificação de casos excepcionais e pontos fora da curva.

A Figura 27 e a Figura 28 mostram um software representado na perspectiva PCM utilizando os mapas em árvore como metáfora visual. Conforme mencionado anteriormente, os elementos de software são representados por retângulos aninhados. Os retângulos mais internos representam os métodos e o mais externos representam aos pacotes.

Usando esta metáfora visual, os métodos (retângulos internos) que estão contidos na mesma classe são representados dentro de um mesmo retângulo mais externo. Para facilitar a visualização, os retângulos que representam métodos têm as bordas pretas e os retângulos que representam classes têm as bordas vermelhas.

De forma análoga, as classes que estão contidas em um mesmo pacote são representadas em um mesmo retângulo mais externo. Para facilitar a visualização, os retângulos que representam pacotes têm as bordas brancas. Este processo prossegue até que se chegue ao nível do projeto que é representado pelo retângulo mais externo a todos.

Figura 27 Mapa em árvores Decorado por Interesses

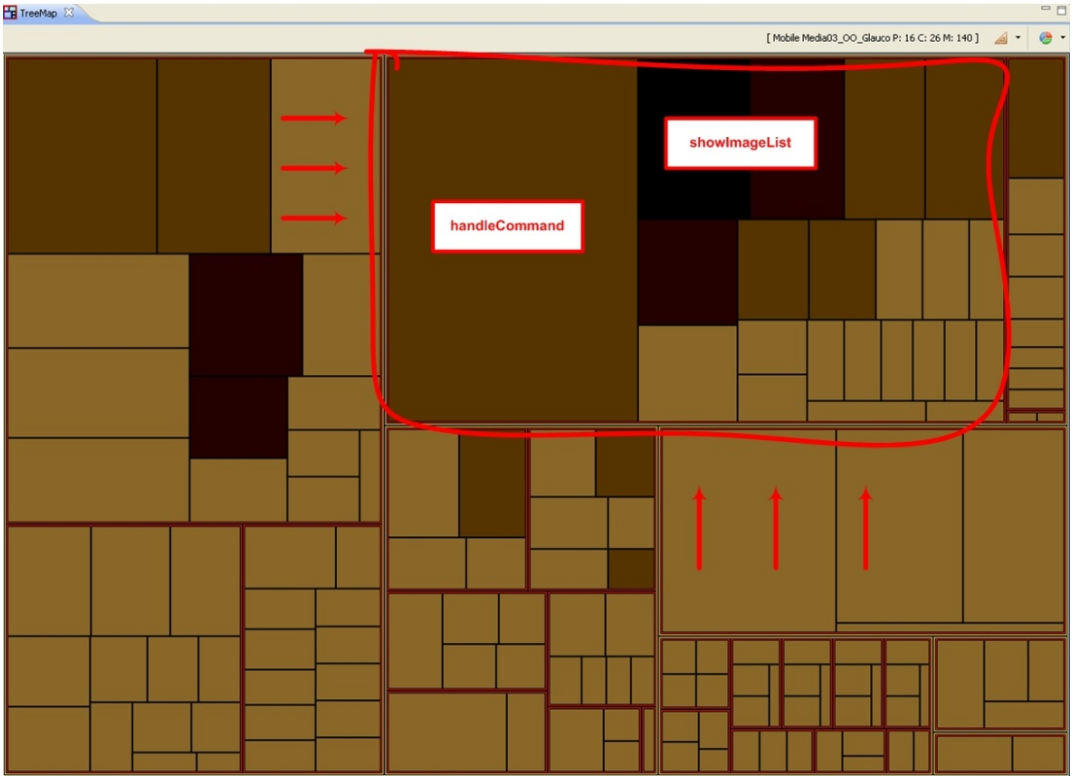

Figura 28 Mapa em árvores Decorado por Complexidade

Utilizando os mapas em árvore desta forma, pode-se representar em uma única tela todos os métodos, classes, interfaces e pacotes de acordo com sua posição na estrutura hierárquica de um projeto.

Além da forma como os retângulos estão posicionados e aninhados, a área e a cor dos retângulos são usadas para decoração. Isto é ilustrado na Figura 27, aonde a cor é utilizada para representar os interesses que afetam cada elemento, e na Figura 28, aonde a cor é utilizada para representar a complexidade ciclomática para cada método.

A nossa implementação de mapas em árvore utiliza zoom semântico para melhor apresentar na tela os pacotes, classes e métodos de um projeto. Programadores podem usar o zoom semântico para navegar do nível mais alto para o mais baixo na representação visual da abstração dos módulos do sistema. Esta navegação ocorre quando se pressiona, respectivamente, os botões esquerdo e direito do mouse sobre a representação do elemento de software na visão montada pelo ambiente.

A Figura 29 apresenta um exemplo de zoom semântico. Nesta figura, a classe marcada em vermelho corresponde à classe marcada na Figura 28 após a aplicação do zoom.

Como em qualquer visão do SourceMiner, os usuários também podem aplicar critérios de filtragem para eliminar a representação de elementos de software que não sejam de interesse da análise naquele momento no mapa em árvore. Usando os recursos de filtragem, os usuários podem se focar em elementos de interesse utilizado critérios de busca ortogonais à estrutura PCM. Pode-se, por exemplo, executar uma filtragem para somente mostrar elementos com certo tamanho ou com certa *substring* no seu nome.

Também de forma similar a outras visões, a partir dos elementos gráficos é possível acessar o código fonte correspondente ao mesmo no editor do Eclipse. Para esta finalidade, os usuários devem simultaneamente utilizar a tecla *Control* e pressionar o botão esquerdo do mouse sobre o elemento gráfico em questão. A camada de desenho gráfico do sistema então solicita ao módulo de Coordenação e Estrutura Visual da camada de coordenação para que seja ativado o Editor do Eclipse para apresentação do código fonte correspondente ao elemento de software selecionado.

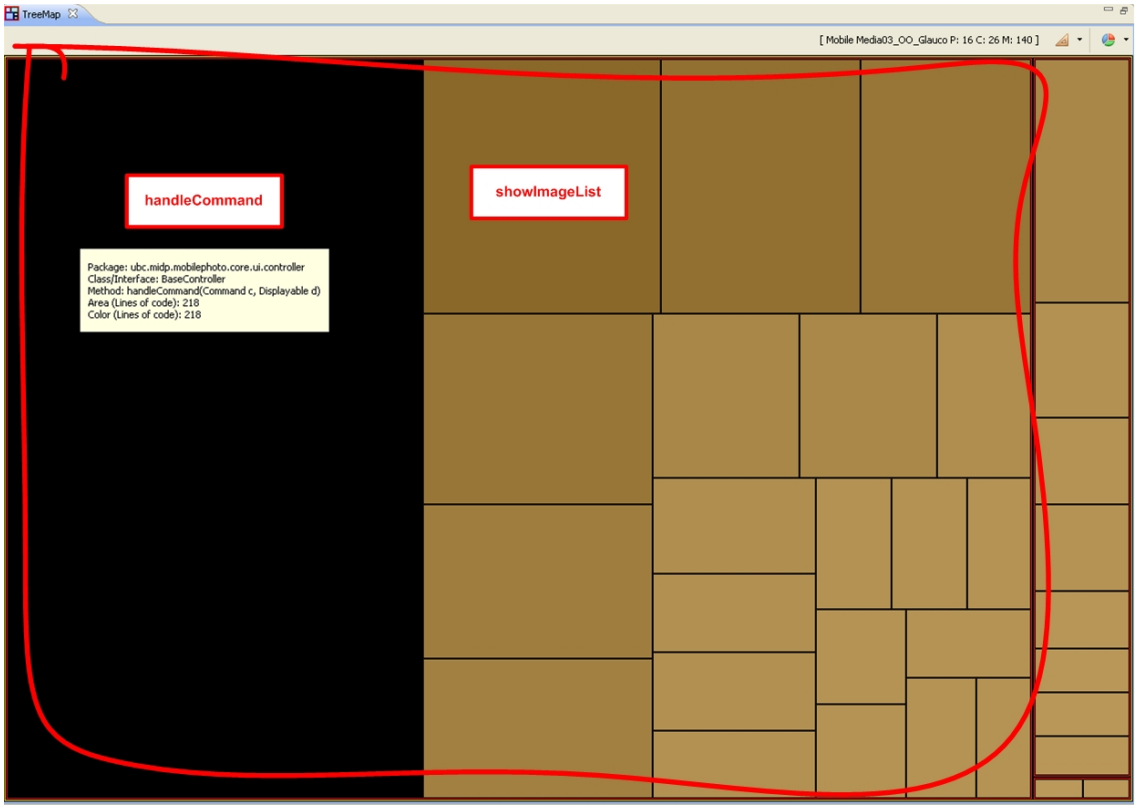

Figura 29 Zoom Semântico no Mapa em árvores

4.2.2 A perspectiva de hierarquia de herança (HH)

A visão polimétrica (Lanza e Ducasse, 2003) foi selecionada para apresentar a hierarquia de herança da aplicação analisada. As Figuras 30 e 31 apresentam exemplos desta visão. A visão polimétrica apresenta o relacionamento de herança entre as entidades de software (no caso classes e interfaces) como uma floresta de retângulos com bordas arredondadas. Cada árvore da floresta apresenta uma estrutura de herança com as arestas representando o relacionamento de herança existente entre os nós (retângulos).

Originalmente proposto para esta finalidade, a visão polimétrica auxilia na compreensão da estrutura da aplicação analisada (Lanza e Ducasse, 2003). Florestas com muitas árvores com apenas um nó indicam uso limitada a orientação a objeto, por exemplo.

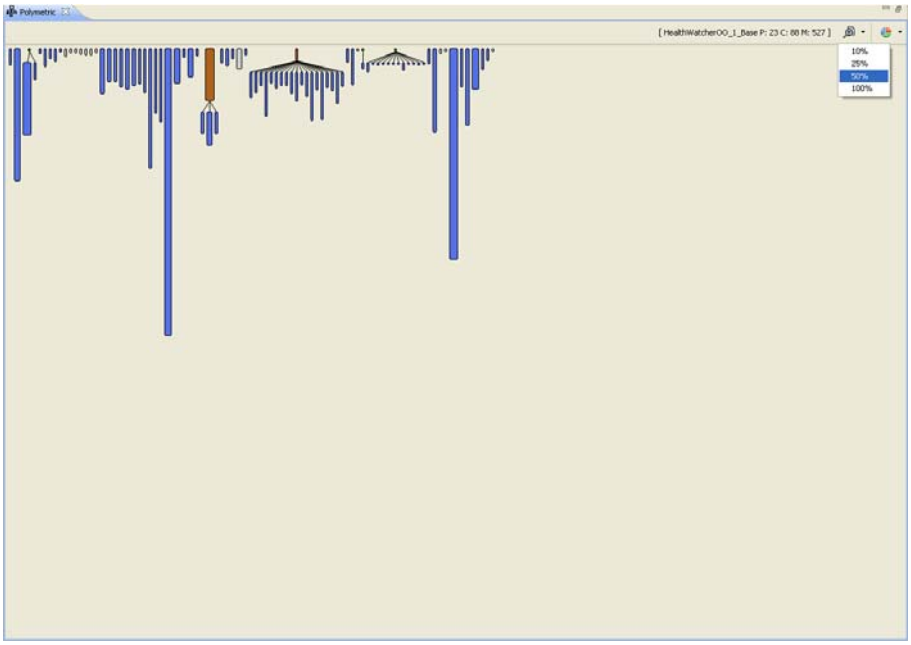

Figura 30 A Visão polimétrica com Zoom de 50%

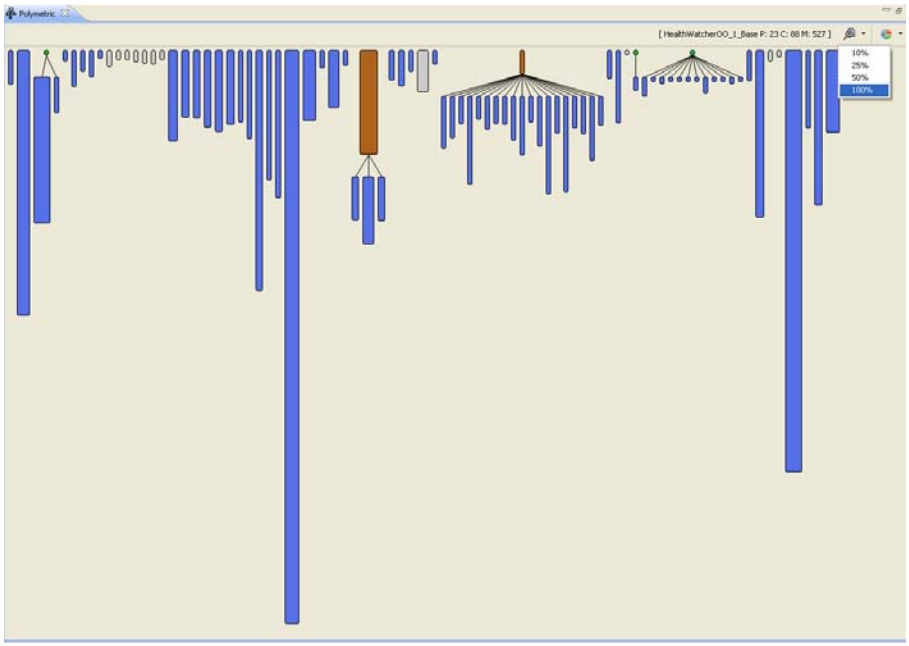

Figura 31 A Visão polimétrica com Zoom de 100%

De forma similar aos mapas em árvores, as dimensões dos retângulos na visão polimétrica são utilizadas para representar propriedades das entidades. Na versão atual do SourceMiner, a largura corresponde ao número de métodos enquanto que a altura corresponde ao número de linhas do código correspondente de uma classe ou interface. A cor é usada para

efeitos de decoração conforme mencionado na perspectiva anterior. Ela pode representar diferentes propriedades do software sob análise.

O zoom geométrico está disponível para melhor representar a visão polimétrica de acordo com o número de elementos na tela. A Figura 30 e a Figura 31 apresentam a visão polimétrica de um projeto com zoom de 50% e 100%, respectivamente.

O zoom semântico também pode ser utilizado para a navegação de sub-árvores específicas de uma determinada hierarquia. Para isto basta clicar em um elemento e a visão será redesenhada a partir deste elemento. A operação pode ser facilmente revertida com outro clique. Está operação é bastante útil para analisar árvores de herança de grande porte.

De forma similar aos mapas em árvores, o código de qualquer entidade pode ser acessado a partir dos elementos gráficos que a representa. Para isto, basta clicar sobre o elemento que o código da entidade representada por ele é aberto no editor do ADS.

Também de forma similar aos mapas em árvores, os recursos de filtragem podem ser utilizados para retirar da cena visual as classes que não são de interesse à análise de herança. Caso uma operação de filtragem remova uma super classe, mas não remova uma de suas sucessoras, toda subestrutura de herança ainda é mantida visível para enfatizar a posição de subclasses na árvore. Neste caso as entidades removidas são mostradas em tons pastéis na cena visual.

4.2.3 A perspectiva de acoplamento (ACO)

Representar visualmente relacionamentos de acoplamento não é uma atividade trivial, especialmente se comparado com as duas perspectivas apresentadas anteriormente. Existem vários tipos de acoplamento. Nós podemos mencionar a chamada de objetos, chamada de métodos, utilização de atributos, e implementação de interfaces, apenas para citar alguns deles (Briand, Daly e Wust, 1999). Além do tipo do acoplamento, há também interesse em representar o sentido do acoplamento (de quem para quem) e o número de vezes em que existem ocorrências de acoplamentos entre pares de entidades.

Devido à quantidade, tipos de acoplamentos e outros aspectos de interesses sobre os mesmos, concluímos que uma única visão não seria suficiente para apoiar sua completa visualização. Resolvemos atacar o problema através do conceito de visualização baseada em múltiplas formas, utilizando quatro visões para representar esta propriedade. São elas: visões de acoplamento baseada em grafos, matriz de relacionamentos, visão tabular e grafo egocêntrico espiral. As seções a seguir discutem cada uma dessas visões.

4.2.3.1 Visões de Acoplamento Baseadas em Grafos Radiais

O grafo é um dos modelos matemáticos mais comuns na área de Computação. Eles são facilmente representados visualmente e são apropriados para modelar a dependência entre módulos de software. Desta forma, eles podem ser utilizados para criar cenários visuais apropriados à identificação de relacionamentos de acoplamento. Muitos destes relacionamentos não são facilmente identificáveis através dos recursos tradicionais disponibilizados pelo ADS, como, por exemplo, a pesquisa de entidades por intermédio da leitura do código fonte.

Infelizmente, o desenho de grafos não é uma atividade trivial. Grafos são fortemente propensos à oclusão de nós e à superposição de arestas (Ghoniem, Fekete e Castagliola, 2004). Existem muitos algoritmos para o desenho de grafos, a maioria é ineficiente (em termos de visualização), difícil de implementar, ou consome muito tempo de processamento (Ghoniem, Fekete e Castagliola, 2004).

Nesta tese selecionamos a metáfora visual de grafos radiais como uma das formas de representar relações de acoplamento entre módulos de software. Grafos radiais são relativamente fáceis de desenhar. Neles, um nó é posicionado no centro do grafo e os outros são desenhados em curvas concêntricas a ele de acordo com a sua vizinhança em relação ao nó central, vide Figura 33.

Este tipo de grafo facilita a análise egocêntrica do nó colocado no meio do grafo, mas não necessariamente dos outros. No SourceMiner, o elemento de software com o maior

nível de acoplamento (definido como o número de arestas que incidem e que chegam ao nó) é inicialmente apresentado no centro do grafo. Qualquer outro elemento pode ser movido para o centro do grafo. Para isto, basta que se clique duas vezes sobre o elemento de interesse. Cores são utilizadas para efeito de decoração da mesma forma como foi descrito nas perspectivas anteriores. Isto, combinado a recursos de filtragem, permite configurar o cenário visual de forma mais apropriada à atividade de compreensão em mãos.

O SourceMiner utiliza grafos radiais para representar acoplamentos baseados em pacotes e em classes.

A Figura 32 mostra um grafo radial de acoplamento baseado em pacotes. Os nós quadrados são utilizados para ressaltar que os elementos representados são pacotes (classes são representadas como círculos). Conforme ilustrado na mesma figura, qualquer dos nós periféricos representando pacotes pode ser selecionado para que sejam também visualmente representadas as classes que o compõem na forma de nós circulares. Neste caso, o SourceMiner apenas mostra classes que justificam o relacionamento de acoplamento do pacote periférico selecionado com o pacote central do grafo.

A Figura 33 mostra o grafo de acoplamento de classes. Os nós circulares representam as classes. Pode-se verificar neste caso que, mesmo em uma representação radial, o grafo pode ficar com um grande número de arestas. Para lidar parcialmente com este problema, há a possibilidade de se selecionar de forma dinâmica os nós e os tipos de relacionamentos de interesse em um dado instante. Isto pode ser feito através das opções de controle disponibilizadas na parte superior da visão. A Figura 34 ilustra este caso, mostrando uma seleção com nós e tipos de acoplamentos selecionados da Figura 33.

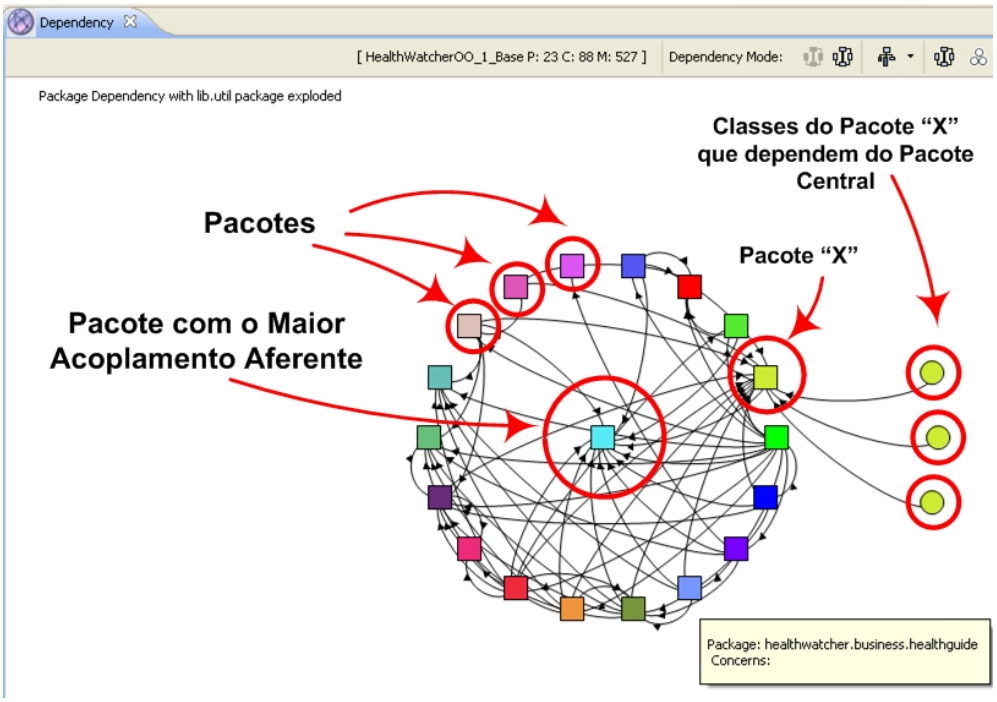

Figura 32 Grafo Radial de Acoplamento de Pacotes

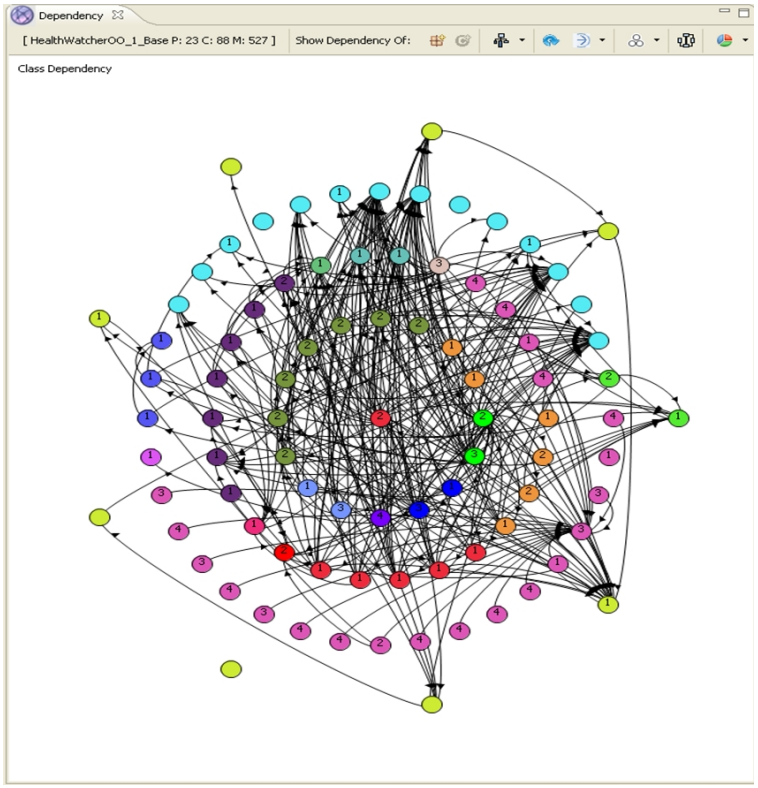

Figura 33 Grafo Radial de Acoplamento de Classes

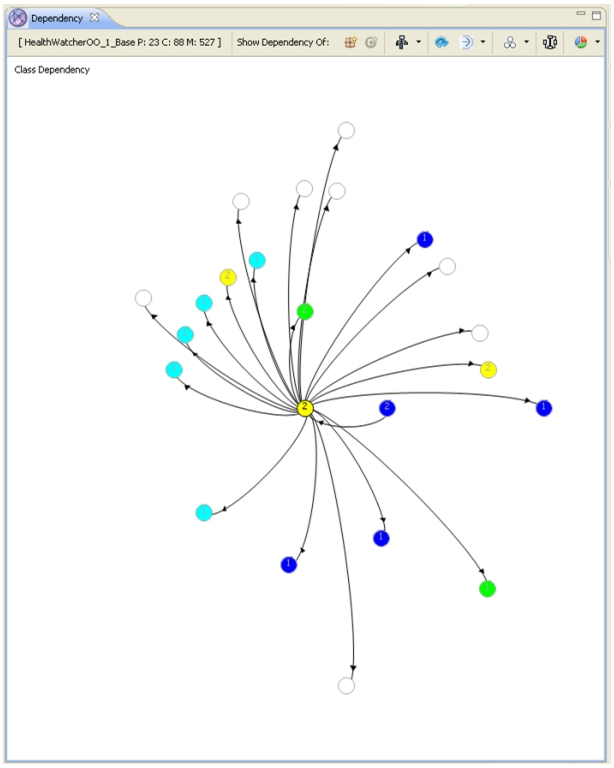

Figura 34 Grafo de Acoplamento com os Nós Selecionados

Mesmo com os recursos descritos, os grafos radiais ainda apresentam duas limitações. A primeira é que não são efetivos para apresentar a força de acoplamento através das suas arestas. Isto é, uma aresta indica que um módulo depende de outro, mas não indica o quanto. A segunda é que a sobreposição das arestas aumenta consideravelmente à medida que aumenta o número de nós no grafo. Este é um problema mitigado, mas não resolvido, através dos recursos de interação e filtragem fornecidos pelo SourceMiner. Por este motivo, o SourceMiner oferece duas outras visões para complementar o uso da visão de grafos radiais conforme descrito nas duas próximas subseções.

4.2.3.2 Visões de Acoplamento Baseadas em Matrizes de Relacionamento

O grafo radial da Figura 33 apresenta claramente poluição visual como resultado do congestionamento de arestas. As matrizes de relacionamentos têm sido utilizadas como uma alternativa interessante à visualização baseada em grafos (Keller e Eckert, 2006). As matrizes não apresentam oclusão nem sobreposição dos elementos que a compõem. O SourceMiner adota esta metáfora com uma alternativa aos grafos.

No SourceMiner, uma matriz composta por linhas e colunas é implementada para a representação de diferentes níveis de relacionamento entre os elementos de software. Ao contrário dos grafos radiais, que só apresentam relacionamento baseados em pacotes e classes, As matrizes do SourceMiner, devido a sua escalabilidade, permitem a apresentação de relacionamento baseados em pacotes, classes e *métodos*

Todas as visões (baseada em pacote, em classes e em método) adotam exatamente a mesma metáfora visual para o desenho das matrizes. A Figura 35 mostra um exemplo de matriz de dependência de pacotes, enquanto que a Figura 36 mostra o exemplo correspondente para classes. As matrizes são quadradas com uma linha e uma coluna para cada elemento sendo analisado.

As linhas representam os elementos através dos seus respectivos nomes (e de um índice numérico) com uma cor associada. Os elementos nas colunas são representados apenas pelo seu índice numérico e pela cor. Devido às limitações de espaço não há necessidade de se repetir o nome da entidade na coluna. Eles são colocados em posição equivalente aos das linhas, o elemento colocado na primeira linha também é colocado na primeira coluna e assim sucessivamente. As cores seguem o mesmo padrão de representação discutido anteriormente, podendo ser decoradas de acordo com vários atributos de interesse do usuário.

A seleção dos tipos de relacionamento segue o mesmo padrão utilizado nos grafos. Desta forma, as matrizes também podem ser simplificadas de acordo com as necessidades de análise.

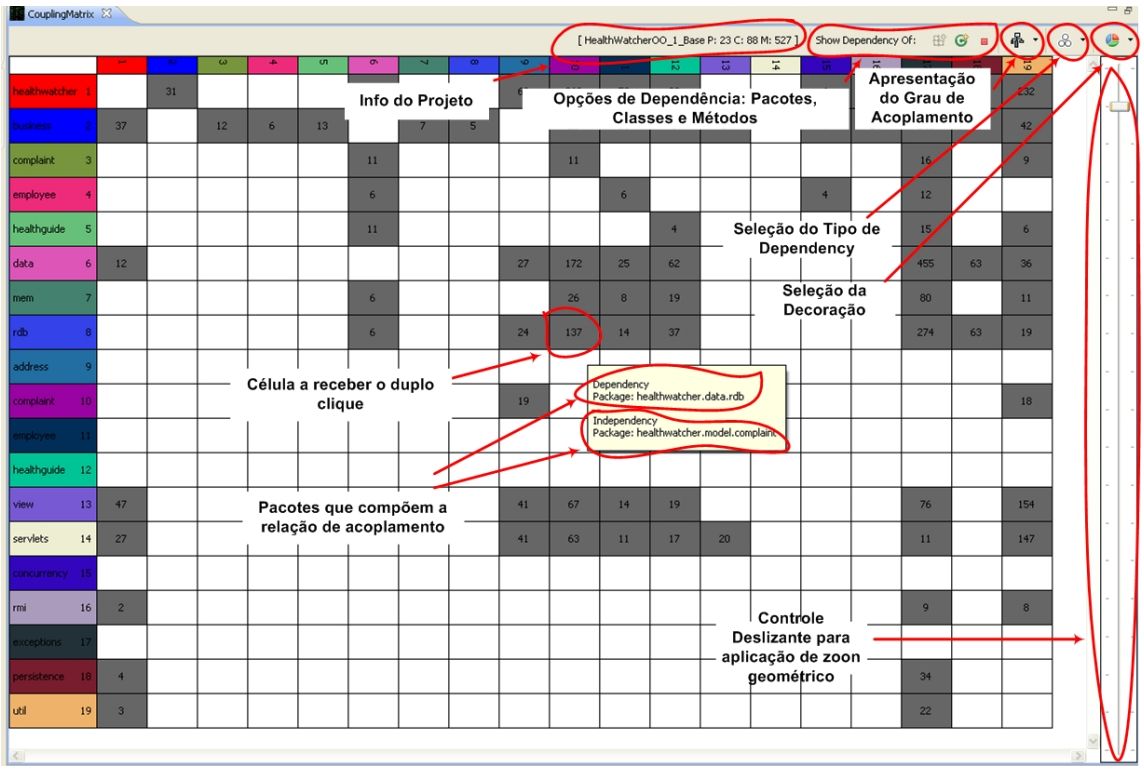

Figura 35 Matriz de Dependências de Pacotes

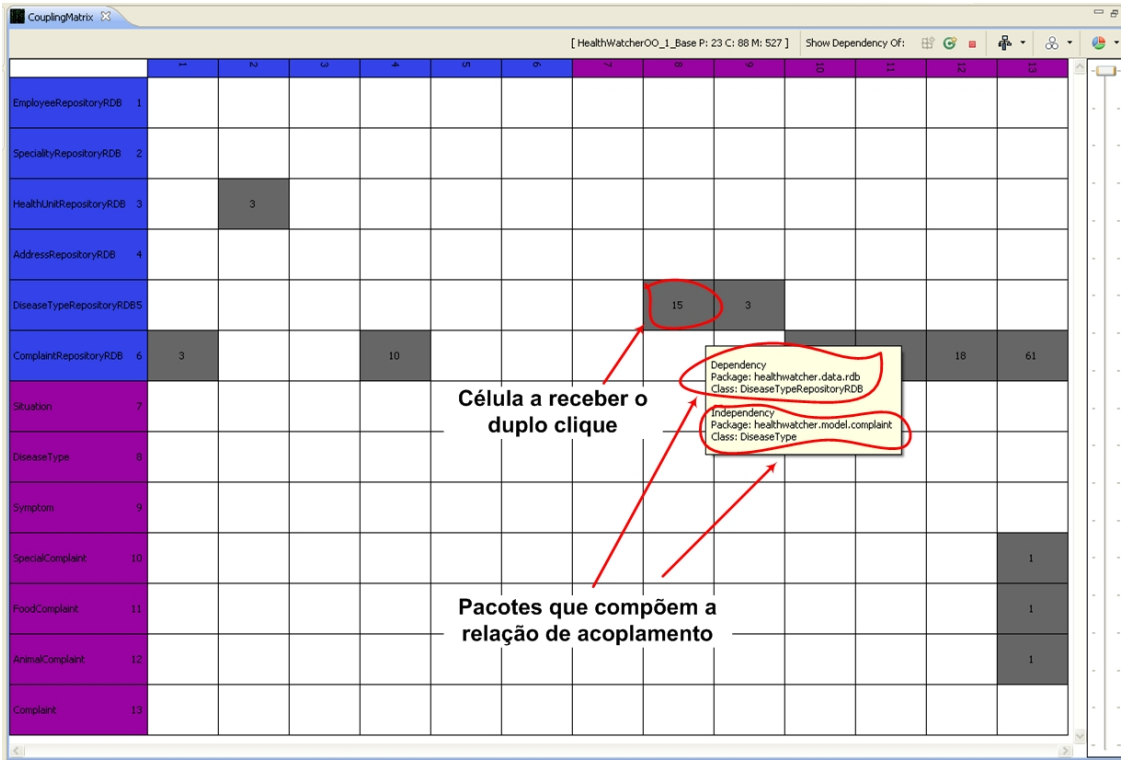

Figura 36 Matriz Gerada por Zoom Semântico

As células da matriz utilizam a cor cinza para indicar que há um relacionamento de acoplamento, enquanto que células de cor branca indicam que não há ocorrência de relacionamento. As dependências são direcionais da linha para a coluna. Isto é, as linhas apresentam os acoplamentos eferentes enquanto que as colunas apresentam acoplamentos aferentes dos elementos. O número de relacionamentos de acoplamento entre dois elementos pode ser apresentado, por decisão do usuário, em cada célula cinza.

O controle deslizante (*slider*) no lado direito da matriz permite o uso do zoom geométrico. O uso deste controle deslizante juntamente com as barras de rolagem vertical e horizontal possibilita a aplicação de zoom e visão panorâmica para identificar regiões na matriz que possuem maior concentração de células cinza. Desta forma, os usuários podem identificar pacotes, classes e métodos com relacionamentos de acoplamento representativos, mesmo em projetos de software de maior porte.

O SourceMiner também possibilita o uso do zoom semântico nas matrizes. Isto pode ser obtido clicando-se duas vezes na célula cinza. Esta ação apresenta uma nova matriz de acoplamento que detalha as dependências do acoplamento selecionado. Por exemplo, clicando-se na célula de acoplamento de pacotes mostrada na Figura 35, será apresentada a matriz de dependência de classes na Figura 36. Esta matriz detalha as classes envolvidas na dependência selecionada no nível mais alto de abstração. Este tipo de ação também é válido na navegação da matriz de classes para matrizes de dependência de métodos. As transições são bidirecionais, o que significa que o usuário pode retornar à matriz original simplesmente clicando com o botão direito do mouse na visão na visão detalhada.

4.2.3.3 Visões de Acoplamento Baseadas em Visão Tabular e Grafos Egocêntricos

A visão tabular (Parte A da Figura 37) e o grafo de acoplamento espiral egocêntrico apresentado na parte B da mesma figura foram especialmente concebidas e implementadas para o SourceMiner. O objetivo destas duas visões é representar a força de dependências entre módulos de uma aplicação, isto é, o número de referências de um módulo

para outro. Isto é totalmente diferente das duas visões apresentadas nas subseções anteriores onde o foco era o grau de dependência entre os módulos. Considere como exemplo que uma classe A declara um atributo e uma variável local que tem a classe B como tipo, e inclui duas chamadas para métodos da própria classe B. Neste caso, o valor da força de dependência entre as classes A e B é quatro enquanto que o grau de dependência entre elas é um.

A visão tabular é uma visão que usa a metáfora de um tabuleiro de xadrez para apresentar todas as classes do sistema como retângulos posicionados em relação à força de acoplamento (FA). A FA de uma classe é a soma dos valores das dependências entre esta classe e as demais com as quais se relaciona. O retângulo representando a classe com o maior valor de FA é colocado na parte superior esquerda da visão tabular. Os outros são arranjados a partir daí em ordem decrescente.

A seleção dos tipos de relacionamento segue o mesmo padrão utilizado nos grafos e matrizes. Desta forma, visões tabulares também podem ser simplificadas de acordo com as necessidades de análise.

O grafo espiral egocêntrico de acoplamento é um zoom semântico da visão tabular. Esta visão é obtida clicando-se duas vezes em um elemento da visão tabular. Ele detalha o FA da classe selecionada utilizando um grafo egocêntrico em um formato espiral.

A classe selecionada fica posicionada no centro, enquanto que outras classes relacionadas a ela são posicionadas ao seu redor. O nó mais próximo do nó central representa a classe com maior valor de FA com a classe sob análise. As outras classes são apresentadas em uma espiral crescente de acordo com a sua força de dependência decrescente em relação ao nó central. A navegação da visão tabular para a visão de grafo espiral egocêntrico é um exemplo de amarração de navegação (Keim e Kriegel, 1996). A uma ação executada na visão tabular, tem-se uma ação correspondente na visão do grafo egocêntrico espiral.

Cores são utilizadas para decoração da visão tabular e do grafo espiral egocêntrico de acoplamento da mesma forma que as visões anteriores. As filtragens também funcionam conforme descrito para as perspectivas anteriores.

Figura 37 As Visões Tabular e Grafo Espiral Egocêntrico

4.3 COMBINANDO AS VISÕES

Uma das maiores utilidades das metáforas visuais disponibilizadas nas perspectivas é o seu uso de forma combinada e coordenada. Esta seção discute o assunto.

O Eclipse, e outros ADSs, já provêem recursos para configuração das suas visões. Como o SourceMiner foi construído de forma integrada a este ADS, as visões do SourceMiner e dos Eclipse podem ser configuradas e utilizadas de forma combinada. Estas configurações podem ser nomeadas e salvas no próprio ADS. A Figura 2 é um exemplo de uma destas configurações.

O Eclipse chama este recurso de configurações de “Perspectivas”. Isto não deve ser confundido com o nosso conceito de perspectiva discutido durante esta tese. Com este recurso, os engenheiros de software podem criar e salvar conjuntos de configurações do SourceMiner-Eclipse para tarefas específicas da área (e.g., reengenharia, depuração, detecção de anomalias de projeto, etc.). Isto também possibilita a execução de estudos específicos para se desenvolver e avaliar configurações do AIMV para estas atividades.

Deve-se ressaltar que a combinação de visões ainda é uma questão de pesquisa em aberto e que não temos conhecimento de publicações que abordem este problema.

Apesar da importância do tema, por questão de tempo, esta questão não pôde ser aprofundada no decorrer do desenvolvimento desta tese. Desta forma, não vamos aqui apresentar um catálogo de combinação de visões para tarefas específicas de engenharia de software. Todavia, enfatizamos aqui a importância de se aprofundar o estudo das combinações de visões em atividades de compreensão de software, e a seguir apresentamos algumas lições aprendidas durante o uso das visões suportadas pelo nosso AIMV.

Com base na nossa experiência, a perspectiva mais usada no SourceMiner é a PCM representada através do mapa em árvores. Na maioria das vezes, o mapa em árvores é a visão de referência ao longo do ciclo de uma atividade de compreensão, tanto por parte do usuário experiente e conhecedor do projeto, como daquele recém integrado à equipe de projeto.

A PCM é utilizada desde o início para uma visão panorâmica da estrutura da aplicação e à medida que é aprofundado o conhecimento necessário para uma atividade, o usuário sempre retorna a ela para melhor contextualizar as informações fornecidas pelas outras visões. A justificativa para esta preferência é o fato desta visão fornecer uma representação abrangente da estrutura da aplicação. Desta maneira ela é utilizada como ponto de partida e referência para a execução das atividades cognitivas, realizando um papel muito similar ao do *Package Explorer* em atividades de codificação no Eclipse.

Também baseado na nossa experiência, a segunda perspectiva mais utilizada é a de acoplamento. Partindo do princípio que o usuário já teve uma metáfora visual que lhe dispusesse uma visão panorâmica da estrutura da aplicação, o usuário tende a buscar em seguida relacionamentos representativos na aplicação. Estas buscas envolvem a análise de relacionamentos específicos de um módulo, de determinados tipos de relacionamentos, ou na identificação de relacionamentos que se destacam em relação aos demais. Se, por um lado, a perspectiva PCM fornece uma visão panorâmica da aplicação, a perspectiva de acoplamento possibilita ao usuário pesquisar situações específicas de relacionamento entre módulos. Nesta perspectiva, o foco é identificar quais módulos estão mais relacionados, como se dão estes relacionamentos e quais interesses os motivam. A combinação destas duas perspectivas abre

um leque de possibilidades para a construção de modelos mentais para a compreensão de software.

Os grafos radiais são os mais utilizados nesta perspectiva. Eles são geralmente combinados aos filtros e utilizados na navegação com zoom semântico para execução de atividades como a identificação de anomalias de modularidade, de desvios arquiteturais e de caracterização de arcabouços (*frameworks*) de software. Para situações específicas, onde se precisa conhecer a força de acoplamento dos módulos na aplicação, passa-se para o uso da visão tabular e do grafo espiral egocêntrico.

A visão tabular é utilizada de forma complementar aos grafos radiais para dar uma visão panorâmica da força de acoplamento dos módulos na aplicação. A partir desta visão panorâmica parte-se para a visão específica da força de acoplamento fornecida pelo grafo espiral egocêntrico. Esta visão é utilizada para conhecer a "comunidade de relacionamento" da classe selecionada e permite a identificação da intensidade dos seus relacionamentos e quais interesses os motivam. Ela é utilizada tanto para identificar e analisar situações não previstas no projeto da aplicação (e.g., desvios arquiteturais), como também para entender melhor o objetivo de cada módulo (e.g., se ele se destaca como provedor ou consumidor de recursos e funcionalidades a outros módulos). Ela auxilia a reflexão se de fato a implementação do módulo corresponde ao seu projeto e pode ser usada para analisar a sua importância real (de acordo com o que foi implementado) na aplicação.

As matrizes de relacionamento fornecem uma visão alternativa de acoplamento entre os módulos de software. Elas não são tão utilizadas quanto os grafos. Todavia, como os grafos apresentam problemas de oclusão para projetos maiores, elas são utilizadas em combinação com os grafos para proporcionar uma visão panorâmica do projeto em relação ao acoplamento nas aplicações de maior porte.

4.4 CONCLUSÃO DO CAPÍTULO

Este capítulo descreveu o modelo conceitual do SourceMiner. Ele adapta o modelo de Card do domínio de visualização de informação, e adiciona a ele vários conceitos de integração, extensibilidade, experimentação e consistência de decoração, para a visualização de software.

Este capítulo também apresentou cada uma das perspectivas e visões que compõem o SourceMiner e os recursos por ele oferecidos para apoiar as atividades de compreensão de software. Foram exemplificados importantes conceitos como o zoom geométrico e semântico. Foram apresentadas as peculiaridades dos mapas em árvores para a visualização da estrutura da aplicação, da visão polimétrica para a representação visual da hierarquia de herança, e das visões de grafos radiais, matrizes de relacionamento, visão tabular e grafo espiral egocêntrico para a representação do acoplamento entre módulos de uma aplicação.

Também foram apresentadas possibilidades do uso combinado das visões baseada na experiência do seu uso pelo autor desta tese.

Capítulo

5

Este capítulo apresenta a arquitetura adotada para a implementação do SourceMiner a partir do modelo conceitual do AIMV proposto no capítulo anterior. O SourceMiner foi desenvolvido sobre o ADS Eclipse e é composto pela Camada do Núcleo do Ambiente (CNA) e pela Camada de Renderização e Visualização (CRV). Também são apresentados os detalhes arquiteturais que permitem estender o SourceMiner através da inclusão de novas funcionalidades e recursos.

5 DESENVOLVENDO UM AIMV PARA O ECLIPSE

O desafio na construção de múltiplas visões coordenadas e com múltiplas formas excede aquela que corresponde à construção de ambientes com uma única visão. Este capítulo apresenta as decisões tomadas durante o desenvolvimento do SourceMiner para a ADS Eclipse. Neste contexto, é apresentada a estrutura concebida para o ambiente, seus módulos e a forma como estão organizados. Em seguida é apresentada a seqüência de funcionamento do ambiente na ADS.

A Figura 38 apresenta a estrutura do ambiente composta por camadas e seus respectivos módulos. A Camada de Visualização e Renderização (CVR) é responsável pela renderização das visões no SourceMiner. A camada do Camada do Núcleo do Ambiente (CNA) é responsável pela captura de dados do ADS para que seja preparado e disponibilizado para a CVR. A camada CNA também coordena a comunicação entre as visões e entre o ADS. As seguintes subseções descrevem as funcionalidades oferecidas por cada uma das camadas.

5.1 A CAMADA DO NÚCLEO DO AMBIENTE (CNA)

A Camada do Núcleo do Ambiente (*CNA*) é a camada principal do SourceMiner. Ela é responsável pela extração dos dados do código fonte. Ela utiliza os recursos disponibilizados pelo Eclipse *Java Development Tool* (*JDT*) para esta finalidade (Eclipse.org, 2011).

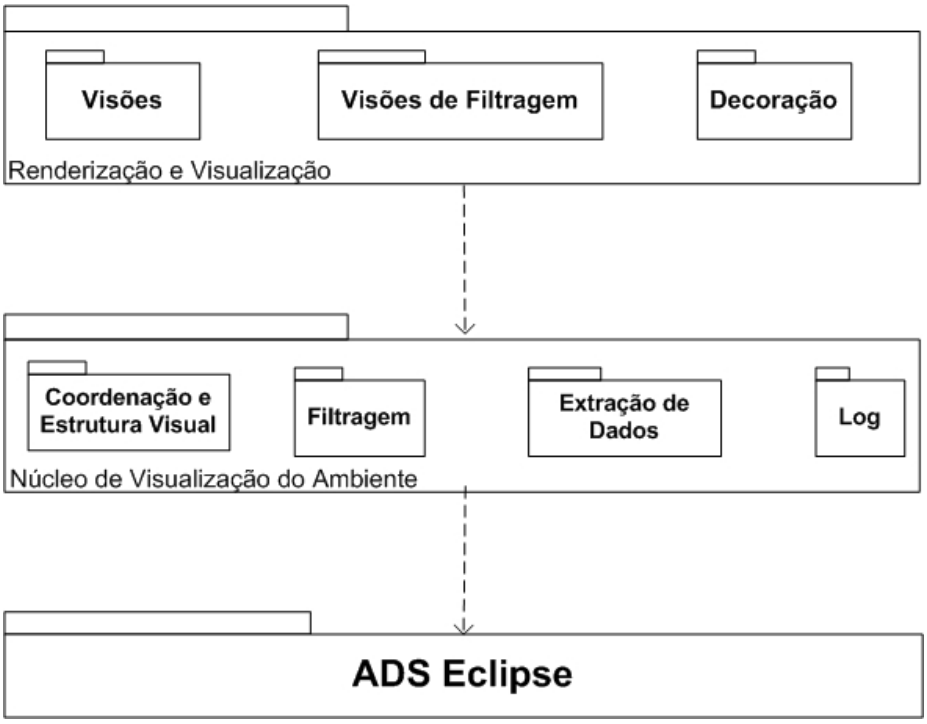

Figura 38 Estrutura de Camadas do SourceMiner

O JDT acessa a árvore abstrata de sintaxe (AST) do projeto e disponibiliza dados relacionados às entidades do software sob análise sem que seja necessário obtê-los a partir do zero. Isto torna o desenvolvimento do AIMV independente da análise sintática da aplicação feita pelo ADS. Com isto, pode-se manter o foco do trabalho na extensão do ADS para a inclusão das visões, e seus mecanismos de coordenação e interação. Este é um forte ponto a favor do uso de um ADS de código aberto como substrato para o **AIMV**.

Além de extrair a estrutura dos dados da aplicação analisada, a CNA também deve oferecer serviços de coordenação entre as visões, filtragem dos dados a serem apresentados

nas visões e registro (*logs*) de operações primitivas executadas pelos usuários enquanto usam o ambiente. A CNA também utiliza recursos do ADS para coordenar seus elementos e elementos do próprio ADS. A próxima subseção apresenta em maior detalhe os módulos da CNA.

5.1.1 O Módulo de Extração de Dados

O **Módulo de Extração de Dados** é responsável pela extração dos dados do código fonte. O Eclipse utiliza uma AST para armazenar uma descrição detalhada e sempre atualizada do software. O ADS dá acesso a estes dados através do JDT. O diagrama apresentado na Figura 39 indica que a extração destes dados pelo SourceMiner ocorre quando um usuário seleciona o projeto Java a ser representado visualmente no **AIMV**.

Na AST, os elementos do programa são organizados hierarquicamente em uma árvore e decompostos em elementos filhos que os compõem. Na ADS Eclipse, a árvore de elementos Java define uma forma para conhecer o projeto baseado nas características da linguagem Java. Ela é composta por elementos como *package fragments*, *compilation units*, *types* (representando classes, interfaces e enumerações), métodos e atributos. No Eclipse, os dados originais da AST Java estão disponíveis na forma de *Java Element Tree* e são disponibilizados pelo *Java Model* API (Eclipse.org, 2011).

A parte esquerda da Figura 39 indica a extração de dados da árvore sintática abstrata correspondente ao código fonte da aplicação a ser analisada. A extração de dados é realizada em duas etapas. A primeira etapa identifica os elementos na árvore Java usando o modelo Java (Eclipse.org, 2011). A segunda etapa obtém dados relevantes dos elementos identificados. É importante ressaltar que: a) a segunda etapa depende dos dados obtidos na primeira etapa; b) cada tipo de informação, conforme indicado na mesma figura, extrai dados relacionados às propriedades de seu interesse.

Conforme apresentado no modelo conceitual proposto no capítulo anterior, a primeira etapa consiste na identificação de objetos que representam elementos tais como

pacotes, classes e métodos da aplicação sob análise. Estes objetos são utilizados na segunda etapa para transformar os dados em estruturas de dados apropriadas. Estas estruturas devem suportar a construção de estruturas visuais representando propriedades como acoplamento, herança e estrutura pacote-classe-método do projeto analisado.

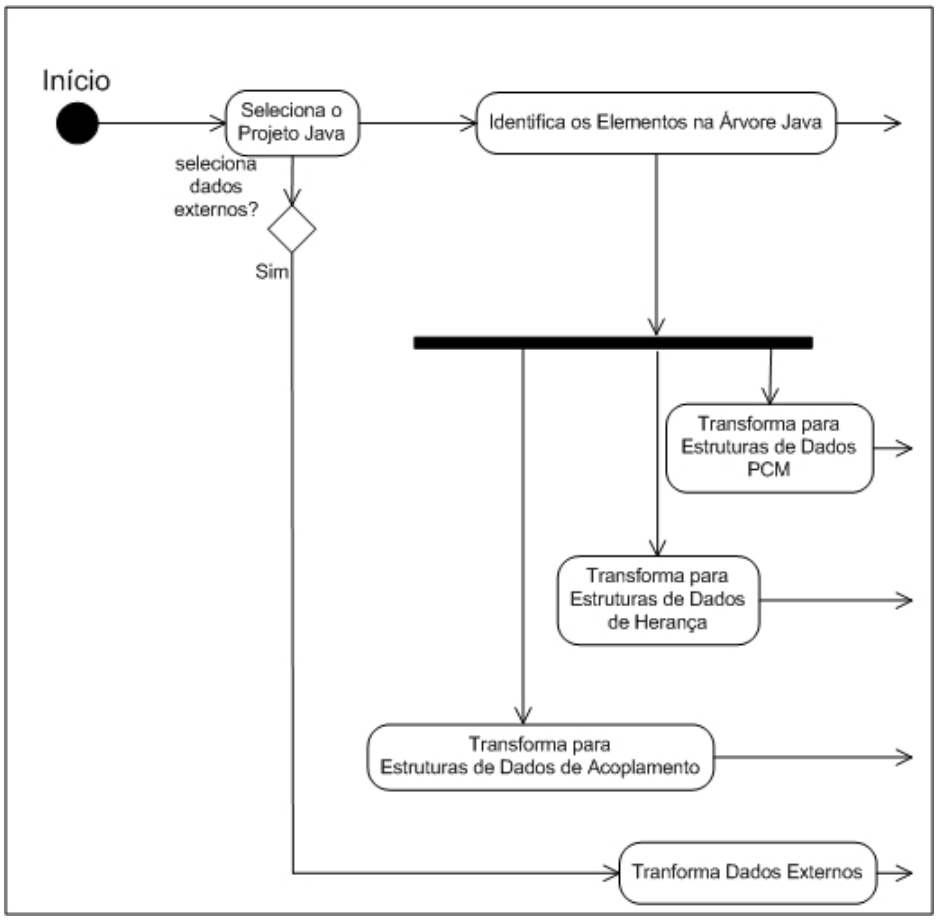

Figura 39 Selecionando o Projeto e Extraindo os Dados

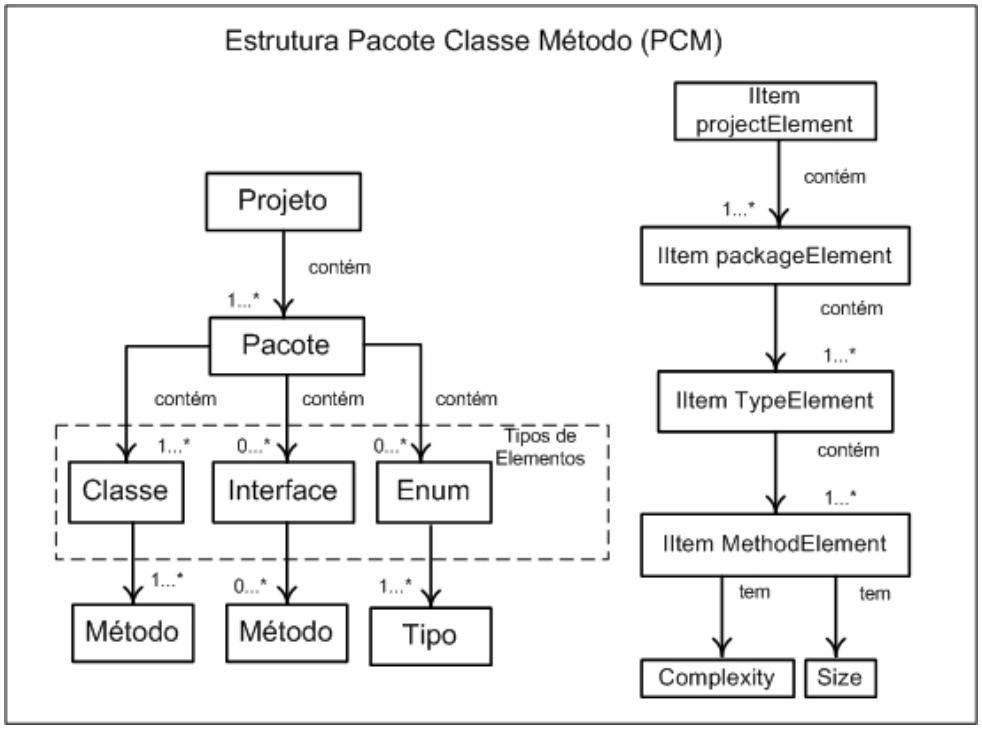

Figura 40 A Estrutura Pacote-Classe-Método

O objetivo da transformação Pacote-Classe-Método (PCM) é a obtenção de dados a respeito da forma como os pacotes, classes e métodos estão organizados em um projeto. A Figura 40 apresenta esta estrutura de dados para esta perspectiva. No modelo, propriedades relevantes de software como tamanho e complexidade também podem ser introduzidas para enriquecer o conjunto de dados a ser apresentado posteriormente como atributos visuais.

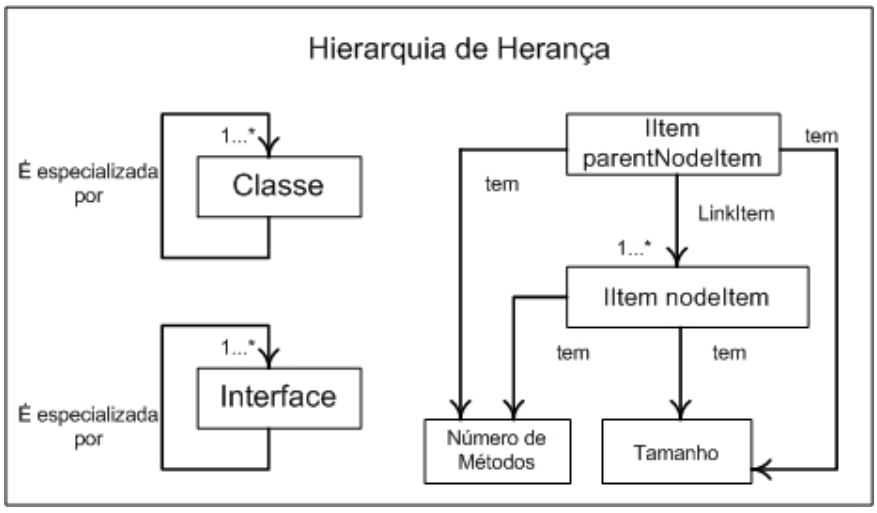

Figura 41 A Estrutura de Herança

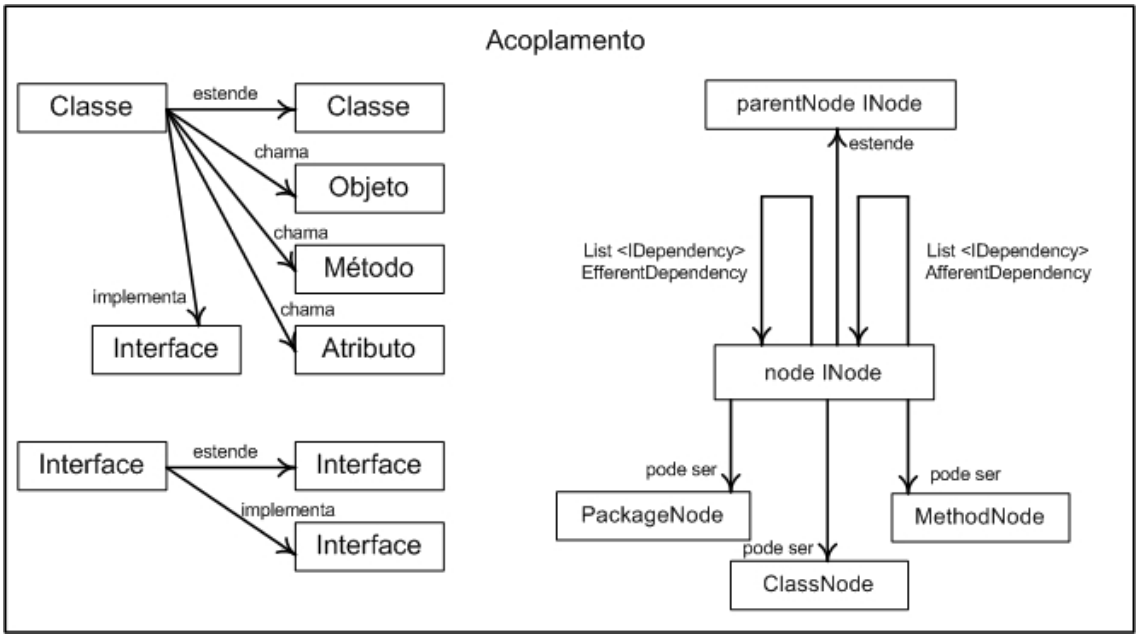

Figura 42 A Estrutura de Acoplamento

O objetivo da transformação de herança é obter dados que descrevam os relacionamentos de herança no projeto. No desenvolvimento do SourceMiner, classes e interfaces são analisadas para que seja conhecido como são especializadas. A Figura 41 descreve esta estrutura.

O objetivo da transformação de acoplamento é a obtenção de dados para descrever os relacionamentos de acoplamento em um projeto. A versão atual do SourceMiner considera os relacionamentos de acoplamento descritos em (Briand, Daly e Wust, 1999). A Figura 42 descreve esta estrutura.

A Figura 39 mostra ainda que dados externos também podem ser incorporados às visões. O ambiente carrega arquivos do tipo XML (*Extensible Markup Language*) com mapeamento das características do software capturados de repositórios externos para elementos de software da árvore Java (Robillard e Weigand-Warr, 2005; Robillard e Murphy, 2007). Estes dados podem ser capturados de repositórios externos de informações tais como sistemas de controles de bugs e ocorrências, sistemas de mapeamento de interesses, e sistemas de controles de versões. A partir dos dados destes arquivos são identificadas propriedades

associadas a cada entidade de software para que seja possível o posterior mapeamento para as estruturas visuais do ambiente.

5.1.2 O Módulo de Filtragem

A Figura 43 é a continuação do diagrama de atividades da Figura 39. Ela ilustra as atividades de filtragem, indicando que fica a cargo do usuário a utilização ou não do filtro. A filtragem atua tanto na transformação de dados como nos dados externos previamente discutidos. As ações de filtragem disparam de imediato a atualização das estruturas visuais para que as visões sejam renderizadas de forma consistente e coordenada. Elas são propagadas automaticamente para todas as visões.

O modulo de filtragem fornece, portanto, meios aos usuários para que eles possam dinamicamente ajustar os conteúdos das cenas visuais de forma consistente e sincronizada.

Por exemplo, pode-se ter a necessidade de se apresentar visualmente as classes que atendam aos critérios configurados pelo usuário como exemplificado na Parte C da Figura 2. Através do uso do mecanismo de filtragem é possível informar os critérios a serem utilizados para que, logo em seguida, seja possível visualizá-los simultaneamente em todas as visões. Isto garante cenários visuais consistentes, proporciona foco na análise e facilita a exploração de um dado conjunto de elementos de software usando diferentes perspectivas.

5.1.3 O Módulo de Coordenação e Estruturação Visual

O **módulo de coordenação e estruturação visual** é responsável por definir quais estruturas visuais serão usadas para o mapeamento dos dados para sua representação na visão. Estas estruturas visuais definem as dimensões e a região onde a representação visual será

criada. A representação gráfica é composta pelos seguintes elementos gráficos que aparecem na visão: pontos, linhas e superfícies.

Propriedades gráficas (tamanho, orientação, cor, e forma) são aplicadas aos elementos gráficos e determinam as propriedades na metáfora visual que será definida para a visão.

Este módulo também trata e provê mecanismos de amarração de navegação (Shneiderman e Plaisant, 2010) (Keim e Kriegel, 1996), vinculação (Shneiderman e Plaisant, 2010) (Keim e Kriegel, 1996) e pincelamento (Shneiderman e Plaisant, 2010) (Keim e Kriegel, 1996). Um dos objetivos do uso de múltiplas visões coordenadas é o de possibilitar a composição visual de diferentes representações de dados em diferentes visões. Conforme mostrado na Figura 44, este modelo permite o uso conjugado da amarração de navegação, da vinculação e do pincelamento para que as visões do SourceMiner sejam coordenadas e sincronizadas entre si. Estes mecanismos asseguram que as ações dos usuários em uma visão sejam imediatamente refletidas em todas as outras. Por exemplo, selecionando-se uma classe em uma visão implica que esta classe será simultaneamente selecionada em todas as outras visões.

Outra funcionalidade oferecida por este módulo é a integração com os recursos nativos do ADS como o editor de código. Através deste último, pode-se selecionar uma classe específica diretamente da sua representação visual para que se tenha acesso através do editor do Eclipse ao código fonte correspondente do elemento selecionado.

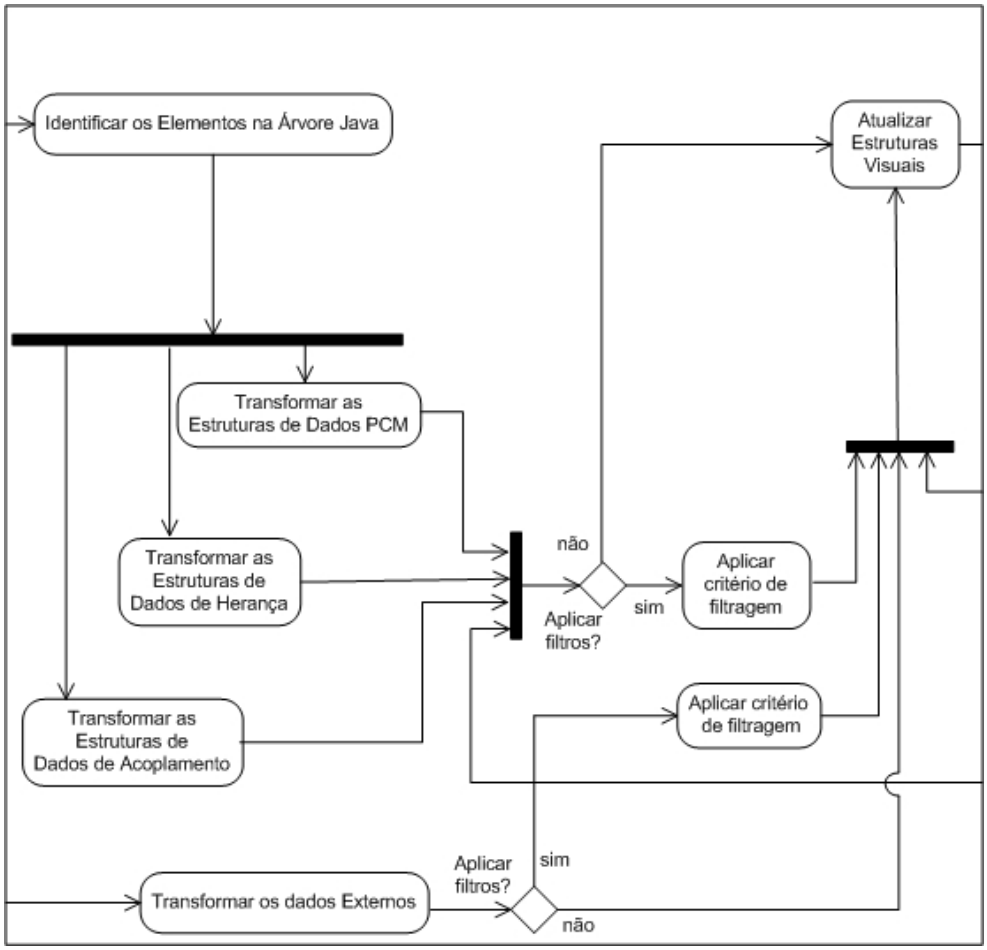

Figura 43 Filtragem e Estruturas Visuais no SourceMiner

O mecanismo de coordenação é implementado através do padrão *publisher-subscriber* (Buschmann, Meunier, *et al.*, 1996). As estruturas visuais se registram como interessadas nos eventos de interação dos usuários. A existência de qualquer interação do usuário com o ambiente, como por exemplo, a seleção de um novo projeto, ou a execução de ações de filtragem ou zoom, dispara notificações para as estruturas visuais registradas. Uma vez notificadas, as estruturas visuais são automaticamente atualizadas para que possam refletir as modificações correspondentes.

O uso do padrão *publisher-subscriber* é um importante ponto de extensão no SourceMiner. Novas estruturas visuais podem ser tanto incluídas no ambiente a qualquer momento como também registradas para interação com outros componentes do AIMV.

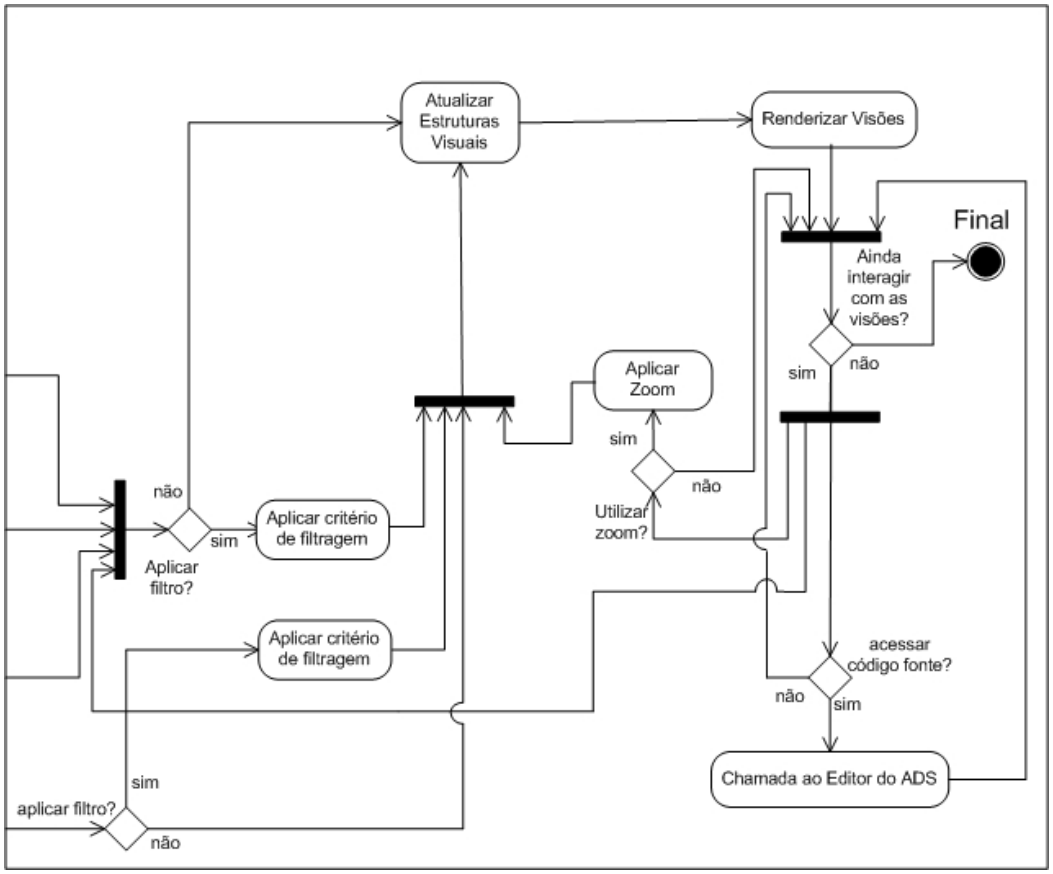

Figura 44 Renderização e Visualização no SourceMiner

5.1.4 O Módulo de Monitoramento

No modelo desenvolvido foi concebida e implementada uma funcionalidade para registrar as operações primitivas executadas pelo usuário durante a utilização tanto do ADS como do SourceMiner. Esta funcionalidade pode ser ativada e desativada a qualquer momento. A idéia é que sejam registradas ações do usuário durante o uso do ADS para a análise da execução de atividades no ambiente. Os registros são compostos por quatro campos. O primeiro campo contém a data e horário da ação. O segundo campo contém a visão a partir da qual o evento foi disparado. O terceiro campo apresenta a ação executada pelo participante na visão. E o quarto campo é dedicado a informações adicionais, tais como o objeto alvo da ação juntamente com dados adicionais a respeito da ação.

A funcionalidade de monitoração registra as ações dos usuários em um nível refinado de detalhes. O alto nível de detalhe e baixo nível de abstração dificultam a análise dos registros de uso coletados. As abordagens para exploração desta informação, tais como mineração de dados e técnicas de visualização estão sendo desenvolvidas agora, conforme será descrito na seção trabalhos em andamento no capítulo 6 desta tese. Estas abordagens estão sendo usadas para se conhecer a freqüência com que uma visão ou funcionalidade foi utilizada, quais são as transições mais comuns entre as visões, e em qual momento de determinada atividade de engenharia de software uma ação ocorreu.

O objetivo do monitoramento é oferecer mecanismos para que seja possível melhor compreender as estratégias adotadas pelos usuários, identificar perfis de uso do ambiente e, principalmente, permitir a exploração interativa para a busca temporal de dados categóricos registrados nos *logs* (Wang, Plaisant, *et al.*, 2009). Isto possibilita a identificação de oportunidades de melhorias no ambiente, o que pode levar a ajustes e inclusão de funcionalidades para melhor apoiar o usuário. Esta é uma novidade em ambientes de visualização de software, embora este recurso já seja aplicado em outras áreas a exemplo do que se conhece como *web analytics* para a avaliação de aplicações web (Kaushik, 2009).

5.2 A CAMADA DE RENDERIZAÇÃO E VISUALIZAÇÃO (CRV)

A camada de Renderização e Visualização (CRV) é responsável pela renderização das visões no SourceMiner e depende dos serviços prestados pela Camada do Núcleo do Ambiente (CNA). Os módulos que as compõem são os seguintes: Visões, Configuração Gráfica, Visões de Filtragem e Decoração das Visões. As subseções a seguir descrevem estes módulos.

5.2.1 O Módulo das Visões

As visões são recursos importantes do SourceMiner. Existem atualmente seis visões no SourceMiner. Elas serão discutidas em detalhes no próximo capítulo desta tese. Nesta seção serão abordados os conceitos relacionados à composição e renderização destas visões.

As visões são resultados da aplicação de uma metáfora visual para apresentar os dados armazenados em estruturas visuais conforme indicado na Figura 44. As visões serão utilizadas pelos usuários para instanciar cenários visuais. Um cenário visual é instanciado a partir de dados da aplicação analisada, de recursos de decoração e da configuração gráfica selecionada pelo usuário. A seleção dos dados é baseada no critério de filtragem. A decoração aplicada é baseada nos atributos de software escolhidos tanto para cor como para o tamanho dos elementos gráficos na visão. A configuração gráfica é definida pelas transformações gráficas tais como os recursos disponibilizados para zoom e visão panorâmica selecionados pelo usuário para melhor visualizar os elementos gráficos. Todas estas três operações são altamente interativas. As possibilidades de configurações gráficas, filtros e decorações serão abordados em detalhes nas subseções seguintes.

Todas as visões devem ser registradas como interessadas nos dados disponibilizados pelas estruturas visuais através do padrão *publisher-subscriber* (Buschmann, Meunier, *et al.*, 1996). Este é outro ponto de extensão do SourceMiner. Novas visões podem ser incluídas no ambiente através da codificação de uma nova metáfora visual. Em seguida, esta nova visão deverá ser registrada como interessada nos dados disponibilizados por uma determinada estrutura visual.

Vale ressaltar que quando duas visões utilizam a mesma estrutura visual elas criam uma visualização com múltiplas formas. Isto ocorre pelo fato de utilizarmos duas metáforas (formas) diferentes para representar o software sob o mesmo ponto de vista (perspectiva).

5.2.2 Configuração Gráfica

Todas as visões têm seus parâmetros de ajustes de renderização subordinados aos controles de interação disponibilizados aos usuários. A maioria destes controles, tais como os controles deslizantes (*sliders*) utilizados para ajuste da visão panorâmica são familiares para a maioria dos usuários. Entretanto, existe um conceito importante que necessita ser abordado. As visões do SourceMiner utilizam tanto o zoom geométrico como o zoom semântico. O zoom geométrico é utilizado para aumentar e diminuir a escala dos elementos na visão. A aproximação do zoom tem como efeito o aumento do número de pixels para a representação de cada elemento visual. O distanciamento do zoom tem efeito oposto.

O zoom semântico, por outro lado, refere-se à mudança do nível de detalhes no qual um conjunto de elementos é apresentado. Neste caso, tanto a aproximação como o distanciamento do zoom poderão ter como efeito, por exemplo, a navegação nas árvores estrutural e de herança, assim como a expansão e a redução das informações de dependência representadas nos grafos e nas matrizes. A informação apresentada em um cenário visual resultante da utilização de um zoom semântico é diferente daquela apresentada pela visão antes da aplicação dos ajustes (Spence, 2007). O próximo capítulo fornecerá exemplos destas operações ao comentar as visões do SourceMiner.

5.2.3 Visões de Filtragem

As visões de filtragem fornecem conjuntos de controles (*widgets*) para a execução das operações de filtragem. Estes controles permitem a exclusão de determinados elementos de software das visões de acordo com um conjunto de critérios específicos. As operações de filtragem são baseadas em valores numéricos e categóricos conforme apresentado pelas Partes A e B da Figura 45. O critério resultante de filtragem é a conjunção de todos os critérios

parciais de filtragem. As visões de filtragem disponibilizam ainda botões de retorno às configurações iniciais para todos os valores e também para valores específicos de cada filtro.

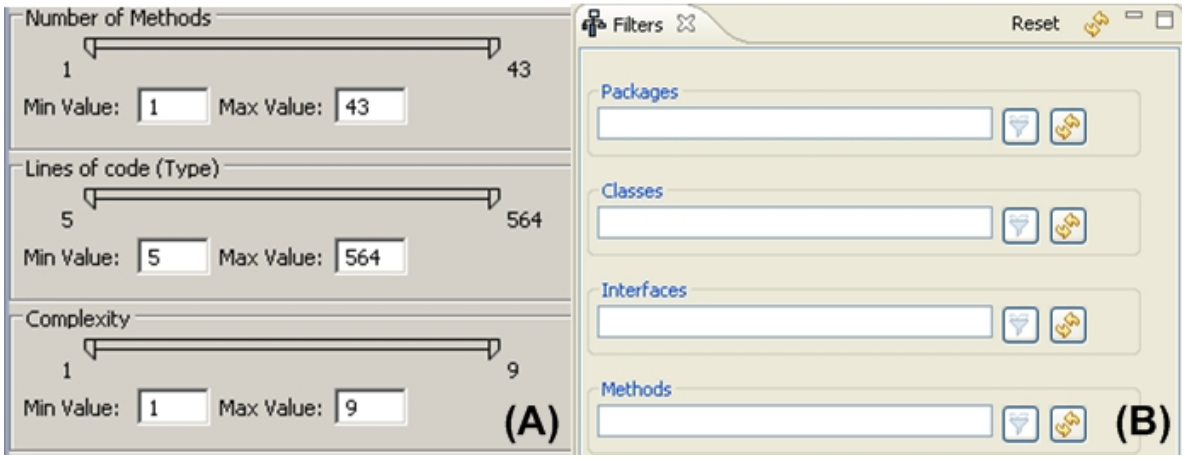

Figura 45 Controles Deslizantes (A) e Caixas de Texto (B)

5.2.4 Decoração das Visões

Os elementos visuais podem ser decorados com atributos visuais tais como tamanho e cor. Alguns destes atributos podem ser utilizados para decoração consistente. Isto é, eles podem ser utilizados como elemento de amarração visual entre todas as visões do ambiente. O SourceMiner utiliza a cor como principal elemento de amarração visual.

A cor é um importante atributo visual em ambientes interativos baseados em múltiplas visões, pois é um atributo de destaque e fácil modificação na maioria, senão todas, as metáforas visuais. SourceMiner utiliza um único módulo para colorir (decorar) os elementos de representação visual de forma consistente em todas as visões. Este módulo de decoração utiliza cores para representar propriedades de software tais como tamanho do elemento, complexidade ciclomática de métodos, e o tipo do elemento de software (por exemplo, classes concretas, abstratas, internas ou externas). As Figura 46 e Figura 47 apresentam a mesma visão decorada por tipo de elemento e interesses implementados,

respectivamente. Na Figura 47, o número em cada retângulo indica o número de interesses que afetam uma determinada classe.

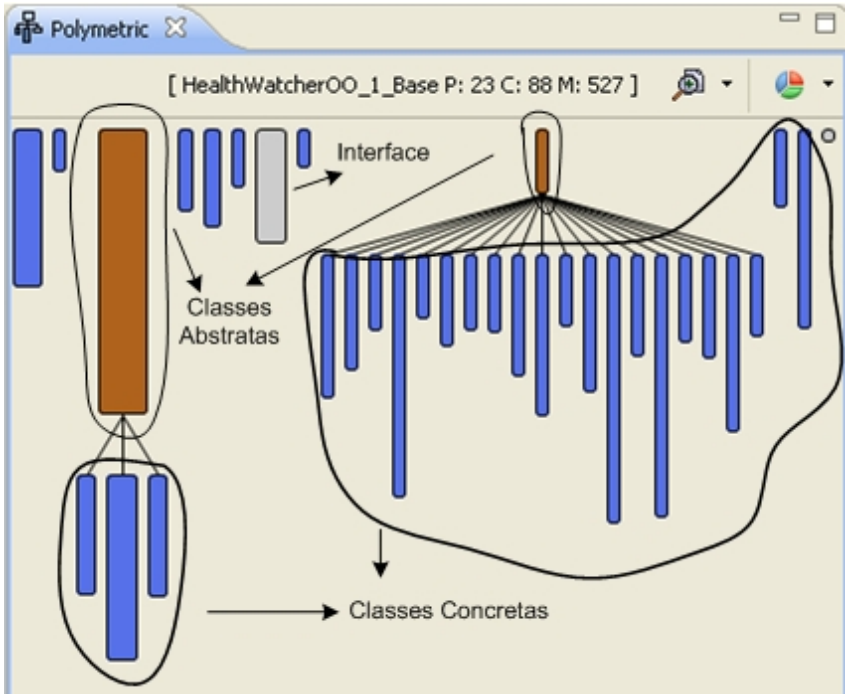

Figura 46 Cores Representando os Elementos de Software

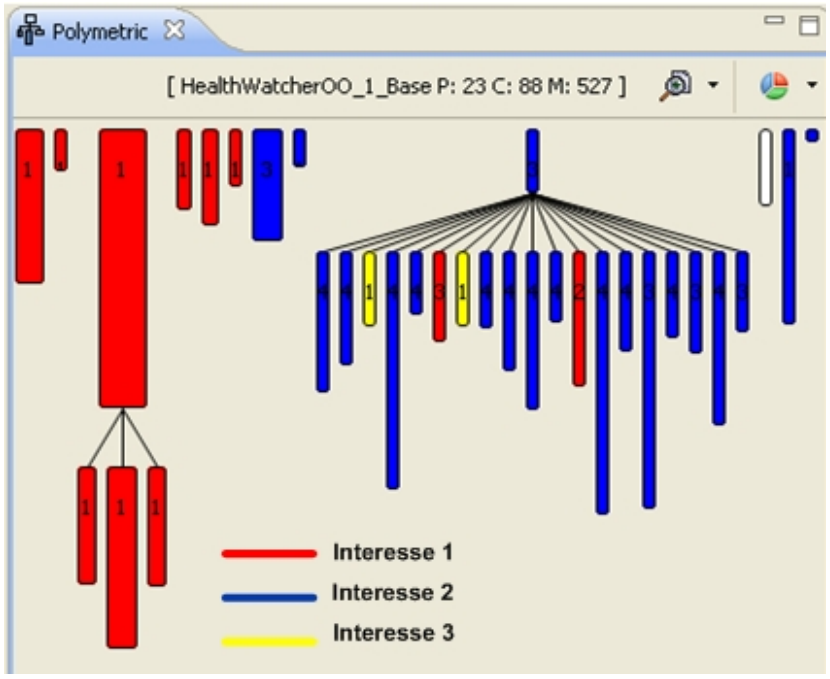

Figura 47 Cores Representando os Interesses

5.3 CONCLUSÃO DO CAPÍTULO

Este capítulo apresentou a arquitetura do SourceMiner composta pelas camadas de Visualização e Renderização (CVR) e pela camada Núcleo do Ambiente (CNA) com seus respectivos módulos e estruturas relevantes. Esta última foi descrita como a camada responsável pela extração dos dados do código fonte através dos recursos disponibilizados pelo Eclipse Java Development Tool (JDT). Também foi enfatizado que além de extrair a estrutura dos dados da aplicação analisada, a CNA também oferece serviços de coordenação entre as visões, filtragem dos dados a serem apresentados. Na seqüência foi apresentada a camada de Renderização e Visualização (CRV) e os módulos que a compõem. Ao longo do capítulo também foram apresentadas possibilidades de extensão do SourceMiner através da identificação de recursos na arquitetura para tal finalidade.

Capítulo

6

Após a conclusão do desenvolvimento conceitual do SourceMiner, os próximos passos foram a sua caracterização e avaliação em situações realísticas de uso. Este capítulo apresenta um estudo observacional conduzido com o objetivo de caracterizar o uso do SourceMiner no apoio à identificação de anomalias de modularidade (code smells).

6 CARACTERIZANDO O USO DO SOURCEMINER

Para a caracterização do uso do SourceMiner foi conduzido um estudo observacional que é descrito neste capítulo. Para sua avaliação foram realizados dois estudos de caso que são descritos no capítulo seguinte.

6.1 ESTUDO OBSERVACIONAL

O estudo observacional é destacado na **parte 2** da Figura 48. O estudo analisou a efetividade do uso da representação visual de interesses (*concerns*) no AIMV para o apoio à identificação de anomalias de modularidade (*code smells*) (Fowler, 1999).

A detecção de anomalias de modularidade (*code smells*) de software é uma atividade de importante em engenharia de software. O estudo teve dois objetivos principais. O primeiro objetivo foi analisar a forma como o SourceMiner apóia os programadores na identificação de anomalias específicas de modularidade. O segundo objetivo foi identificar as

possíveis estratégias adotadas pelos participantes para a identificação destas anomalias. Este estudo foi publicado em (Carneiro, Silva, et al., 2010) e é representado como P13 na Figura 3

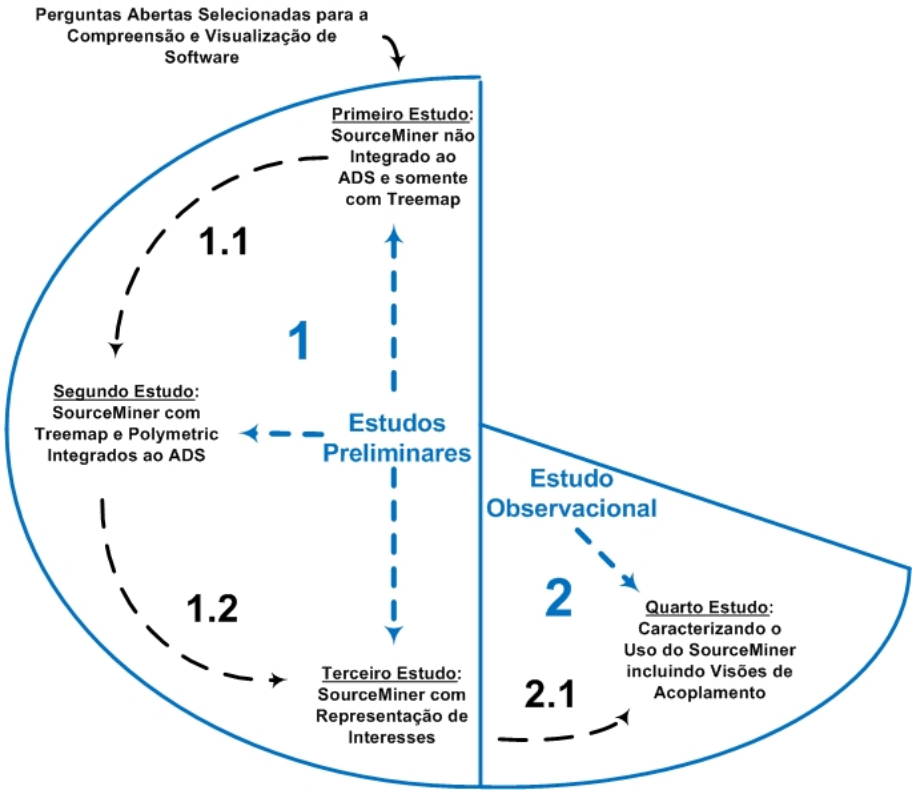

Figura 48 Um Estudo Observacional

6.1.1 Descrição da Versão do SourceMiner Utilizada no Estudo

A versão do SourceMiner utilizada neste estudo (etapa **2J** na Figura 3) era completa e possibilitava a representação visual dos interesses através das perspectivas pacote-classe-método, hierarquia e acoplamento. A diferença da versão do SourceMiner em relação á utilizada em estudos anteriores foi o suporte à visualização de dependências através da criação de visões específicas para esta finalidade: uma visão, baseada em grafos radiais, focando-se nos tipos de acoplamento entre os módulos de software; e outras duas visões, baseadas em estruturas tabulares e grafos egocêntricos, focando-se na força de acoplamento entre os módulos de software.

Esta última propriedade resultou na introdução do conceito de visualização multiforma no Sourceminer, pois três visões são utilizadas para apresentar vários aspectos da propriedade de acoplamento no AIMV. Uma visão, baseada em grafos radiais, é direcionada para os tipos de acoplamento entre os módulos de software. Como visto no Capítulo 4, ela mostra os vários tipos, a direção e o nível (entre pacotes e entre classes) do acoplamento entre módulos de software. As outras duas visões, baseadas em estruturas tabulares e grafos egocêntricos, são direcionadas para a força de acoplamento entre os módulos de software. Como visto no Capítulo 4, a estrutura tabular sumariza a força de acoplamento de todos os módulos do sistema e a visão egocêntrica, obtida a partir de um zoom conceitual da visão tabular, detalha a força de acoplamento de um módulo específico de sistema.

Como discutido no Capítulo 4, todas as visões possuem recursos de decoração consistentes. Propriedades importantes dos interesses, além do entrelaçamento e espalhamento, podem ser analisadas através do uso combinado das três perspectivas e suas respectivas metáforas visuais. Por exemplo, a forma como um interesse influencia os relacionamentos de acoplamento e herança pode ser avaliada visualmente e de forma simultânea. Além disso, nesta versão as propriedades dos interesses podem ser analisadas em conjunto com propriedades dos módulos tais como tamanho e complexidade.

6.1.2 Planejamento do Estudo

O propósito deste estudo foi responder à questão: “O SourceMiner apóia de forma efetiva as atividades de compreensão de software através da representação visual de interesses através das suas três perspectivas?”

O objetivo deste estudo expresso segundo o gabarito GQM é:
Analisar o *SourceMiner, um ambiente de visualização interativo*
Com o propósito de *caracterizá-lo*

Em relação ao *apoio às atividades de compreensão de software através da representação visual de interesses nas suas três perspectivas*

Do ponto de vista *dos pesquisadores do ambiente SourceMiner*

No contexto de *atividades de compreensão de aplicações desenvolvidas em linguagem Java em um ambiente in-vitro.*

As **perguntas de pesquisa** (**PP**) deste estudo foram:

Pergunta de Pesquisa 1 (PP1): Em que medida um ambiente visual baseado em múltiplas visões com representação de interesses apóia a identificação de anomalias de modularidade?

Pergunta de Pesquisa 2 (PP2): Quais são as possíveis estratégias para a identificação de anomalias de modularidade usando a abordagem para múltiplas visões?

Todos os participantes foram solicitados a identificar três tipos de anomalias de modularidade em cinco versões de uma aplicação usando o SourceMiner. Não foi permitido acesso ao código fonte. O objetivo não foi avaliar, mas caracterizar como cada participante executou as atividades solicitadas (Wohlin, Runeson, *et al.*, 2000). No restante desta seção, apresentaremos o planejamento relacionado ao tutorial aplicado, ao objeto de estudo, a lista de referência das anomalias de modularidade, aos participantes e ao questionário de atividades.

O objeto de estudo. Cinco versões consecutivas do MobileMedia (MM) (Figueiredo, Cacho, *et al.*, 2008) foram selecionadas para este estudo. A maior versão tem em torno de quatro mil linhas de código distribuídas em 18 pacotes e 50 classes. O MobileMedia foi selecionado pelo fato do código fonte das versões em Java estar disponível na web e já ter sido utilizado em outros estudos (Carneiro, Sant´Anna, *et al.*, 2009; Figueiredo, Cacho, *et al.*, 2008; Figueiredo, Silva, *et al.*, 2009; Conejero, Figueiredo, *et al.*, 2009), incluindo o terceiro estudo relatado nesta tese.

Os interesses a serem utilizados neste estudo já haviam sido identificados pelos próprios desenvolvedores da aplicação e mapeados para o código fonte (Figueiredo, Cacho, *et al.*, 2008). Por este motivo, a precisão tanto na seleção dos interesses como no seu mapeamento não foi o foco das nossas perguntas de pesquisa.

A lista de referência das anomalias de modularidade. As anomalias de modularidade selecionadas para este estudo foram *Feature Envy* (**FE**), *God Class* (**GC**), e

Divergent Change (**DC**). As suas respectivas definições já foram apresentadas no capítulo 2 (subseção 2.9). Dois especialistas foram convidados para identificar as ocorrências das três anomalias de modularidade nas cinco versões do MobileMedia. Estes especialistas são pesquisadores que já participaram de atividades anteriores relacionadas ao desenvolvimento, manutenção e avaliação do objeto de estudo. Cada especialista foi instruído para utilizar suas próprias estratégias para a identificação das anomalias sem que fosse utilizado o SourceMiner. O primeiro especialista focou na inspeção manual do código e utilizou estratégias para identificação de anomalias de modularidade relatadas em (Lanza, Marinescu e Ducasse, 2005; Marinescu, 2004). Utilizando uma estratégia diferente, o segundo especialista utilizou análise baseada em heurísticas sensíveis a interesses para identificar instâncias das três anomalias de modularidade (Figueiredo, Sant'Anna, *et al.*, 2009). No final, os especialistas estabeleceram consenso para a composição definitiva da lista de referência de anomalias de modularidade. O documento final registrou 42 ocorrências das três anomalias selecionadas.

Os participantes. Cinco desenvolvedores foram selecionados para participação no estudo através de listas de contatos do autor desta tese. O número de participantes permitiu o balanceamento entre o custo do estudo e a análise qualitativa detalhada, assim como a generalização dos resultados (Pfleeger, 1995; Yin, 2002). De forma similar a outros estudos (Robillard, Coelho e Murphy, 2004) que utilizaram o mesmo número de participantes, o objetivo foi fazer observações baseadas na análise detalhada e qualitativa da forma como as atividades foram executadas no lugar de testar a causalidades das hipóteses através de estatística inferencial. Para ser elegível para inclusão no estudo era necessário possuir experiência em programação orientada a objeto. Esta experiência foi verificada através do preenchimento de formulário elaborado para tal finalidade. Verificou-se que os participantes tinham diferentes níveis de experiência. Por exemplo, alguns deles trabalhavam na indústria enquanto outros eram alunos de pós-graduação. Todos participaram como voluntários sem que fosse disponibilizada qualquer compensação para a execução das atividades do estudo. Nenhum membro do nosso grupo de pesquisa foi participante do estudo.

O questionário de atividades. O questionário de atividades disponibilizado aos participantes foi composto de:

- Descrição resumida das funcionalidades do objeto de estudo;

- Descrição e exemplo com trechos em código Java de cada tipo de anomalia a ser identificada;
- Instruções para registrar as versões, classes e ou métodos afetados pelas anomalias de modularidades, assim como solicitação de registro de data e horário da identificação;
- Instruções para solicitação de suporte remoto em caso de dúvidas nos procedimentos;
- Solicitação de descrição das estratégias utilizadas incluindo o uso das metáforas visuais, interesses, e outros recursos disponibilizados pelo SourceMiner;
- Instruções para enviar ao autor desta tese as informações solicitadas e o arquivo de monitoramento de atividades gerado automaticamente pelo SourceMiner.

Tutorial sobre o SourceMiner. Decidiu-se que antes da execução do estudo, os participantes completariam uma sessão de tutorial sobre o uso do SourceMiner. Para se familiarizar com o AIMV e os seus recursos, os participantes receberiam a ferramenta (e sua documentação) e teriam que utilizá-la para responder 28 questões a respeito de uma aplicação selecionada para o treinamento. A aplicação, chamada de HealthWatcher, já tinha sido utilizada em outros estudos sobre modularidade de software (Greenwood, Bartolomei, et al., 2007).

6.1.3 Execução do Estudo

Conforme planejado, todos os participantes realizaram treinamento. Após a disponibilização do SourceMiner e do HealthWatcher, os participantes tiveram um intervalo de 24 horas para responder às 28 perguntas de treinamento. Durante a sessão do tutorial, o autor desta tese esteve disponível por email e bate-papo via internet para fornecer orientações

a respeito do uso do SourceMiner. Depois de finalizadas as atividades do tutorial, os participantes foram solicitados a enviar suas respostas para conferência. No caso de respostas erradas fornecidas pelos participantes, o autor desta tese forneceu instruções complementares via email e bate-papo para que fossem esclarecidas as dúvidas em relação ao uso do AIMV.

Após a disponibilização das versões do MobileMedia, os participantes tiveram um intervalo de 72 horas para responder às 24 perguntas do questionário. Não foi permitido o acesso ao código fonte nem nenhum tipo de pesquisa diretamente nele em nenhuma etapa do estudo. Durante a sessão das atividades, o autor desta tese também esteve disponível por email e bate-papo via internet para fornecer orientações a respeito do uso do SourceMiner. Depois de finalizadas as atividades do questionário, os participantes foram solicitados a enviar suas respostas para o autor da tese. Por último, os participantes preencheram um formulário para registrar sua opinião, pontos positivos e oportunidades de melhorias a respeito do estudo, do uso do SourceMiner e da execução das atividades.

O tempo total de duração das atividades de cada participante (**PA**) foi o seguinte: **PA1** (55 min), **PA2** (01:51 h), **PA3** (49 min), **PA4** (02:48 h) e **PA5** (06:05 h).

6.1.4 Análise e Discussão dos Resultados

Pergunta de Pesquisa 1 (PP1)

Hipótese Nula 1: O SourceMiner não oferece apoio efetivo à identificação de anomalias de modularidade.

Hipótese Alternativa 1: O SourceMiner oferece apoio efetivo à identificação de anomalias de modularidade.

Para a investigação de PP1 foram utilizadas as métricas de precisão e revocação adaptadas de (Moha, Gueheneuc, *et al.*, 2010) e (Rijsbergen, 1979). A precisão quantifica a taxa de anomalias de modularidade corretamente identificadas pelo número de anomalias de

modularidade detectadas. A revocação quantifica a taxa de anomalias de modularidade corretamente identificadas pelo número de anomalias de modularidade da lista de referência. Estas duas métricas são definidas a seguir:

$$\text{Precisão} = \frac{\text{Anomalias de Modularidades Existentes} \cap \text{Anomalias de Modularidades Detectadas}}{\text{Anomalias de Modularidades Detectadas}}$$

$$\text{Revocação} = \frac{\text{Anomalias de Modularidades Existentes} \cap \text{Anomalias de Modularidades Detectadas}}{\text{Anomalias de Modularidades Existentes}}$$

A Tabela 4 apresenta os valores de Precisão (**p**) e Revocação (**r**) para cada participante (de **PA1** a **PA5**) na identificação das anomalias de modularidade em todas as cinco versões do MobileMedia (das versões de 3 até 7).

Tabela 4 Resultados por Participante do Estudo 4

	PA1		PA2		PA3		PA4		PA5	
	r	*p*	*r*	*p*	*r*	*p*	*r*	*p*	*r*	*p*
GC	**0,7	**0,9	0,5	0,4	0,4	0,4	0,4	0,8	1	0,4
DC	0,4	0,8	**0,8	**0,7	0,4	0,8	0,2	0,4	1	0,4
FE	**0,5	**0,5	*Não identificado		0,0	0,0	0,6	0,2	1	0,2

* Participante 2 não identificou a anomalia Feature Envy
** Melhores pares de precisão – revocação para cada anomalia de modularidade

Como pode ser observado na Tabela 4, ocorreram variações significativas de precisão e revocação tanto comparando os participantes entre si como comparando as anomalias de modularidade entre si. Por exemplo, o valor da revocação para *God Class* (**GC**) variou de 0.4 (**PA3** e **PA4**) para 1.0 (**PA5**). Apesar do alto valor de revocação, **PA5** obteve valores baixos de precisão.

Altos valores de revocação e precisão são desejáveis. Por exemplo, PA2 identificou corretamente 80% (**r** = 0.8) das ocorrências de *Divergent Change* com uma precisão de 70% (**p** = 0.7).

Os resultados da Tabela 4 também apresentam evidências iniciais de que o participante 1 seguiu estratégias diferentes para a identificação de anomalias de modularidade. Por exemplo, pode-se supor que **PA5** adotou uma abordagem otimista. **PA5** identificou todas

as ocorrências de **GC**, **DC**, e **FE** em relação à lista de referência (revocação = 1), mas o fez com baixa precisão (**p** =< 0.4), isto é, não mais do que 40% das recomendações estavam corretas.

Por outro lado, **PA1** demonstrou uma abordagem conservativa e não identificou todas as ocorrências de **GC**, **DC** e **FE** (revocação <= 0.7). Entretanto, entre todos os participantes, **PA1** obteve os melhores valores de precisão na identificação das anomalias de modularidade. **PA2** informou no questionário que não estava confiante na identificação da anomalia *Feature Envy*.

A Tabela 5 apresenta os resultados de precisão e revocação considerando cada anomalia de modularidade deste estudo. Dados desta tabela indicam que, na média, os participantes identificaram corretamente 60% (**r** = 0.6) de todas as instâncias de *God Class* com 60% das recomendações corretas (**p** = 0.6). Os valores correspondentes de precisão e revocação para o *Divergent Change* foram 0.5, enquanto que os resultados para *Feature Envy* foram **r** = 0.4 e **p** = 0.2.

Tabela 5 Resultados por Anomalia de Modularidade

	Revocação	**Precisão**
GC	0,6	0,6
DC	0,5	0,5
FE	0,4	0,2

Os resultados registrados tanto na Tabela 5 como na Tabela 6 apresentam evidências de que a Hipótese Alternativa 1 é verdadeira, pois o SourceMiner apóia de forma efetiva a identificação das anomalias de modularidade *God Class* (**GC**) e *Divergent Change* (**DC**). A análise dos resultados permitiu o registro das observações 1 e 2 descritas a seguir.

Observação 1: Os valores de precisão e revocação indicaram maior efetividade na identificação das anomalias de modularidade *God Class* (**GC**) e *Divergent Change* (**DC**). Um exemplo é a classe *BaseController* identificada como **GC**. A razão principal para que os participantes tenham reconhecido a classe *BaseController* como **GC** é que foram facilmente detectadas em função dos seus valores anômalos em termos de tamanho e do número de interesses por ela implementada. Além disso, métodos como *showImageList* e

handleCommand também foram identificados pelo número maior que os demais em termos de interesses implementados.

Observação 2: Os diversos valores de precisão e revocação obtidos pelos diferentes participantes sugerem, de forma preliminar, que a representação visual de interesses no SourceMiner oferece apoio a diferentes perfis de usuários durante a identificação de anomalias de modularidade, desde os usuários conservadores até os otimistas.

Pergunta de Pesquisa 2 (PP2)

Hipótese Nula 2: O SourceMiner não é flexível o bastante para permitir o uso de estratégias bem sucedidas para a identificação de anomalias de modularidade.

Hipótese Alternativa 2: O SourceMiner é flexível o bastante para permitir o uso de estratégias bem sucedidas para a identificação de anomalias de modularidade.

A partir dos resultados obtidos da análise de **PP1** foi possível verificar que os participantes tiveram diferentes percepções e julgamentos durante a identificação das anomalias de modularidade. Os diferentes valores de precisão e revocação apresentados na Tabela 4, o monitoramento de atividades e os formulários de opinião indicam que os participantes adotaram diferentes estratégias na identificação de anomalias de modularidade.

Através dos registros de monitoração de eventos gravados automaticamente pelo SourceMiner foi possível revelar estratégias bem sucedidas. A Tabela 6 apresenta as visões e os recursos que os participantes mais utilizaram durante a execução das atividades.

Tabela 6 Estratégias na Identificação de Anomalias

	God Class	***Divergent Change***	***Feature Envy***
P1	** Mapa em Árvore + Interesses	Grafos	** Visão Tabular + Grafo Espiral Egocêntrico
P2	Visão Polimétrica + Mapa em Árvore + Interesses	** Visão Polimétrica + Mapa em Árvore + Interesses	Não executou a atividade
P3	Grafos + Interesses	Grafos + Interesses +Visão Tabular	Mapa em Árvore + Interesses
P4	Mapa em Árvore + Interesses	Mapa em Árvore + Interesses	Visão Tabular + Grafo + Visão Polimétrica
P5	Mapa em Árvore + Interesses + Visão Tabular	Visão Tabular	Grafos

** Melhores pares de precisão – revocação para cada anomalia de modularidade

Para identificar *God Class* (**GC**), quase todos os participantes utilizaram plenamente o mapa em árvores com a representação visual de interesses. Observe que PA1, que alcançou melhores valores de precisão e revocação, adotou esta estratégia. Desta maneira, inicialmente os participantes configuraram as visões para representar visualmente os interesses.

Opcionalmente, a visão polimétrica também foi utilizada pelo participante PA2 para identificar situações anômalas. Diferentemente de outros participantes, o participante PA5 também utilizou a visão tabular, além do mapa em árvores, para identificar *God Class*. PA5 identificou todas as ocorrências de *God Class* da nossa lista de referência. Este fato é uma evidência inicial de que as visões de acoplamento podem ser úteis para a identificação de anomalias de modularidade através do número de dependências das classes candidatas a *God Class*. Com base nestas informações, pode ser registrada a observação 3 conforme descrito a seguir.

Observação 3: Uma estratégia possível para identificar candidatos a *God Class* é o uso conjunto das visões mapa em árvores e polimétrica com as respectivas representações de interesses. A visão tabular juntamente com o grafo espiral egocêntrico pode oferecer informação complementar através do número de dependências existentes com as classes candidatas a *God Class*.

Um resultado interessante foi o fato de todos os participantes terem identificado a classe *BaseController* como *God Class* usando as estratégias mencionadas. Como exemplo, a Figura 49 apresenta um cenário com a versão 3 da aplicação MobileMedia no qual as classes *BaseController* e *ImageAcessor* (indicados através de setas) claramente se destacam como candidatas à anomalia de modularidade *God Class*. Além disso, pode-se verificar que estas classes têm métodos que realizam diferentes interesses (representados por cores).

No caso de *Feature Envy*, PA1 também apresentou os melhores valores de precisão e revocação. Este participante informou no questionário que a visão tabular e o grafo espiral egocêntrico foram utilizados para a identificação desta anomalia. Conforme já descrito anteriormente, estas visões representam classes e interfaces em ordem decrescente de dependência.

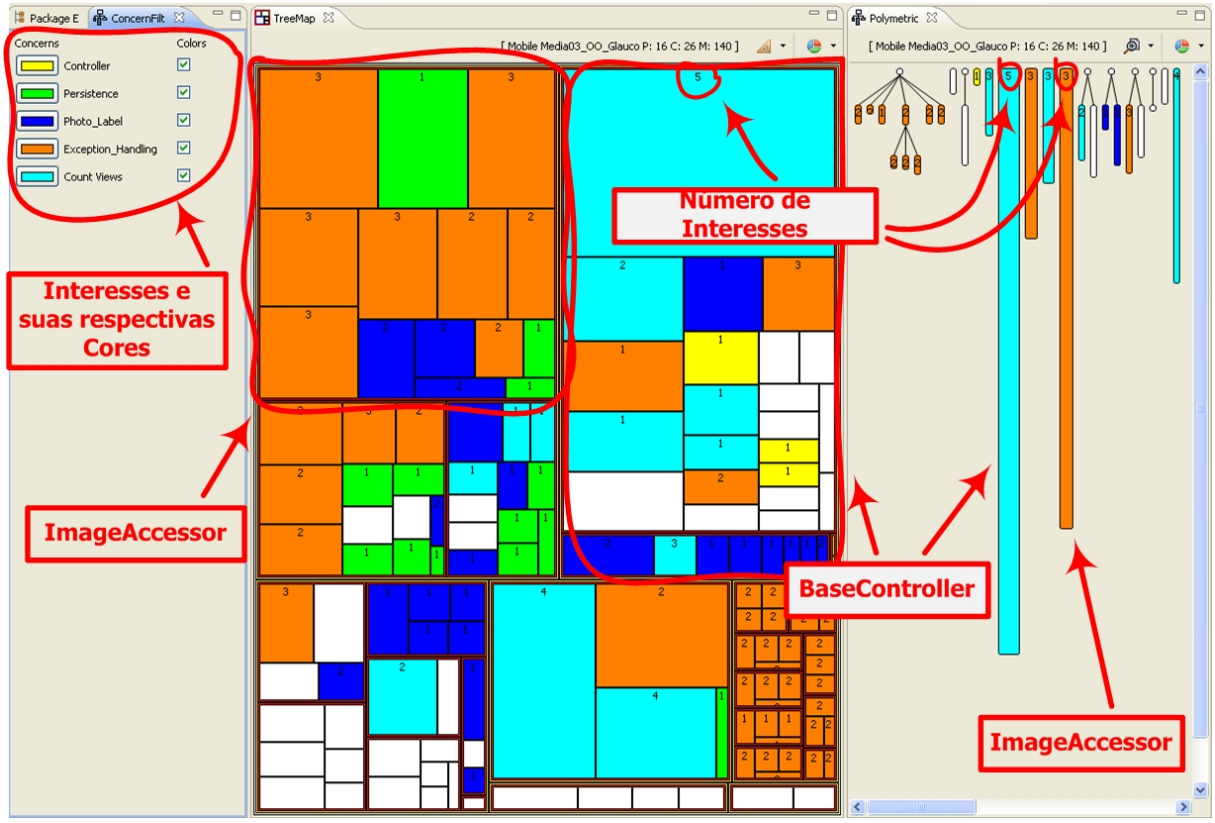

Figura 49 Identificando Anormalidades de Modularidade

A visão tabular apresenta a classe *BaseController*, classe 1 da parte A da Figura 50, no topo esquerdo da tela as classes com maiores valores de dependência. Nesta visão, o participante selecionou e clicou duas vezes no retângulo que a representa. Na seqüência apareceu o grafo espiral egocêntrico apresentando todos os relacionamentos de dependência desta classe.

A visão polimétrica também foi utilizada pelos participantes PA3 e PA4 para confirmar que a classe também realiza outros interesses secundários além do seu interesse principal. Isto foi feito passando-se o mouse sobre os retângulos ou analisando o número de interesses apresentados em cada retângulo. De fato, utilizando as visões mencionadas é possível facilmente confirmar *BaseController* como candidata a *Feature Envy*, pois esta classe se sobressai quando se verifica o seu interesse por outras classes. Com base nestas informações, pode ser registrada a observação a seguir.

Observação 4: Uma possível estratégia para a identificação de classes candidatas a *Feature Envy* candidatas é o uso combinado da visão tabular e do grafo espiral egocêntrico

através dos quais podem ser observados em detalhes os relacionamentos de dependência da classe candidata.

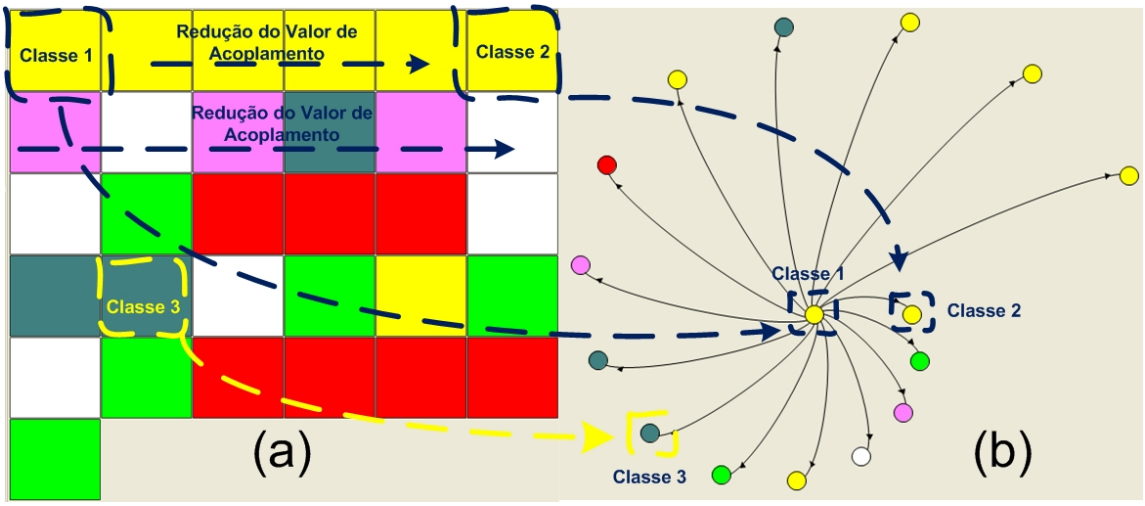

Figura 50 Visões Tabular e Grafo Espiral Egocêntrico

Com relação a *Divergent Change* (**DC**), o participante PA2 fez o uso combinado do mapa em árvores e da visão polimétrica. Estas visões auxiliam na identificação de classes que estão sujeitas a vários tipos de modificações por diferentes razões, mas nenhum dos outros participantes utilizou estas duas visões juntas para a identificação da anomalia citada. Isto pode explicar porque somente PA2 alcançou aproximadamente 70% para os valores de precisão e revocação para esta anomalia (vide Tabela 4). Com base nesta informação, pode ser registrada a observação a seguir.

Observação 5: Uma estratégia possível para a identificação de classes candidatas a *Divergent Change* (**DC**) é o uso o uso combinado do mapa em árvores e da visão polimétrica para identificação de classes que estão sujeitas a vários tipos de modificações por diferentes razões.

Além do registro das observações de 3 a 5 para responder, a análise dos resultados também possibilitou o registro das observações 6 e 7 para responder à pergunta de pesquisa PP2.

Observação 6: Apesar de ter apresentado o segundo menor tempo na execução das atividades, **PA1** demonstrou os melhores resultados em termos de precisão e revocação na identificação de instâncias de *Feature Envy* e *God Class*. Por outro lado, **PA5** gastou o maior

tempo dentre os participantes para a execução das atividades através de um perfil de usuário otimista. Apesar de o estudo ter registrado somente um participante com estilo otimista, um possível padrão de comportamento deste tipo de participante seria a busca por anomalias de modularidade através do uso de um número maior de combinações de visões quando comparado com participantes que utilizam estratégia conservadora.

Observação 7: Outra evidência interessante resultante do estudo foi o fato dos participantes que identificaram corretamente uma classe ou interface em uma versão também as identificaram corretamente nas demais versões de acordo com a lista de referência adotada no estudo. Isto implica que o conhecimento adquirido em uma versão colaborou para a identificação das mesmas anomalias de modularidade nas demais versões.

Estas observações apresentam evidências de que a hipótese alternativa 2 é verdadeira, ou seja, o SourceMiner é flexível o bastante para permitir o uso de estratégias bem sucedidas para a identificação de anomalias de modularidade.

Ameaças à Validade do Estudo. Vários fatores podem afetar potencialmente a validade do estudo. A começar pelo fato de todos os participantes terem sido estudantes ou graduados recentes em Ciência da Computação. Resultados diferentes podem ser obtidos se realizados com diferentes populações de participantes, como, por exemplo, desenvolvedores experientes. Outra importante limitação à generalização dos resultados deste estudo vem do fato das observações da análise do comportamento ter sido feita somente com cinco participantes. Entretanto, como já mencionado, o número de participantes considerou o equilíbrio entre o custo do estudo e a análise qualitativa dos resultados para derivar as observações.

Os participantes executaram tanto o tutorial como a identificação dos três tipos de anomalias de modularidade remotamente. Duas providências foram tomadas para lidar com esta situação. A primeira, a disponibilidade do autor desta tese em tempo real (via correio eletrônico ou bate-papo) para disponibilizar apoio e suporte em como utilizar o SourceMiner, assim como para o esclarecimento de eventuais dúvidas relacionadas ao estudo. A segunda foi o fato dos participantes terem sido avisados que não seria permitido o acesso ao código fonte. Os eventos registrados pelo serviço de monitoramento automático confirmaram que de fato não fora feito nenhum acesso ao código fonte pelos participantes. Conforme já mencionado

anteriormente, não houve interação entre os participantes em qualquer das etapas deste trabalho.

Os exemplos e perguntas utilizados no tutorial e no questionário foram selecionados de forma criteriosa de forma que não exercesse qualquer viés sobre as atividades a serem executadas. Eles não tinham qualquer relacionamento com anomalias de modularidade, o seu objetivo era demonstrar as funcionalidades oferecidas pelo SourceMiner.

Também existem limitações em relações em relação à escolha dos interesses cujos mapeamentos foram disponibilizados na aplicação e também em relação às anomalias de modularidade escolhidas. Por exemplo, os interesses mapeados para as versões do Mobile Media podem não ter sido representativos o bastante para a identificação das anomalias de modularidade. Entretanto, os mesmos interesses foram utilizados em outros estudos tais como os relatados em (Figueiredo, Silva, *et al.*, 2009) e (Figueiredo, Cacho, *et al.*, 2008). Além disso, apesar de pequeno, o conjunto de anomalias de modularidade adotadas no estudo é o mesmo daquele adotado em outros estudos da área (Marinescu, 2004; Lanza e Marinescu, 2005).

Os participantes podem não ter entendido o significado das anomalias *God Class*, *Divergent Change* e *Feature Envy*, assim como seu relacionamento com os interesses. Os exemplos e ilustrações através de trechos de códigos de outras aplicações foram disponibilizados nos questionários para cada anomalia de modularidade. A lista de referências pode não ter o registro de todas as ocorrências reais de anomalias de modularidade. Esta limitação foi tratada através de revisões dos registros contidos na lista e também através do envolvimento de dois especialistas que adotaram diferentes estratégias diferentes para a indicação dos registros na lista. Apesar da aplicação escolhida para o estudo ter cinco versões de uma aplicação considerada de tamanho pequeno-médio, não se pode afirmar que os resultados obtidos possam ser generalizados para todos os casos. Estudos futuros devem replicar o protocolo elaborado em diferentes aplicações para permitir a generalização dos resultados. Não se espera que os novos resultados sejam necessariamente idênticos aos obtidos. Entretanto, como o estudo foi realizado com uma aplicação real e suas respectivas anomalias de modularidade, assume-se que os resultados obtidos alcançaram níveis aceitáveis de qualidade externa.

6.1.5 Lições Aprendidas

Alguns resultados importantes foram obtidos neste estudo. Primeiro, as visões enriquecidas pela representação dos interesses ofereceram suporte à identificação das anomalias de modularidade *God Class* (Riel, 1996) e *Divergent Change* (Fowler, 1999). Segundo, foram identificadas estratégias para a identificação de anomalias de modularidade através do SourceMiner com suas múltiplas visões enriquecidas pela representação de interesses. Os dados coletados foram analisados para responder as duas perguntas de pesquisa (**PP1** e **PP2**). Além disto, ao contrário dos outros estudos, os participantes não reclamaram de ausências de perspectivas no SourceMiner. Conclui-se desta maneira que as perspectivas de estrutura, herança e acoplamento, formam um conjunto mínimo aceitável para visualização estática de software.

6.2 CONCLUSÃO DO CAPÍTULO

Este capítulo apresentou um estudo observacional através do qual é caracterizado o uso do SourceMiner como apoio ferramental para a identificação de anomalias de modularidade. As múltiplas perspectivas combinadas com a visualização interativa de interesses foram úteis para a aplicação de diferentes estratégias para a execução das atividades solicitadas aos participantes. Os resultados do estudo para a identificação de três anomalias de modularidade foram registrados através de sete observações que podem ser utilizadas para a derivação de hipóteses a respeito do uso de múltiplas perspectivas na caracterização de aplicações.

Capítulo

7

Este capítulo apresenta dois estudos de caso com o objetivo de avaliar o uso do SourceMiner na indústria. O primeiro estudo de caso consistiu na avaliação do impacto na mudança de tecnologia de duas funcionalidades de um framework para o desenvolvimento de aplicações web no SERPRO. O segundo estudo de caso consistiu na análise de ocorrências de desvios arquiteturais no desenvolvimento de aplicações em outra empresa pública brasileira de grande porte.

7 AVALIANDO O USO DO SOURCEMINER NA INDÚSTRIA

A avaliação do SourceMiner na indústria teve a finalidade de: (a) caracterizar o seu uso no contexto de um ciclo de vida de desenvolvimento real; e (b) identificar possibilidades de interrupção ou desvios negativos nas metas estabelecidas pelas organizações em decorrência do uso do SourceMiner. Visando atingir estes objetivos, foram conduzidos dois estudos de caso in-vivo com o SourceMiner (**Parte 3** da Figura 51).

Uma importante diferença destes estudos em relação aos anteriores foi que em todas as suas fases, incluindo a formulação do problema e o planejamento da solução, contaram com a participação dos representantes das empresas. Vale a pena ressaltar a importante parceria das duas organizações que se disponibilizaram a utilizar o SourceMiner na execução de algumas de suas atividades de engenharia de software. Além disso, as organizações disponibilizaram participantes que executaram as atividades como parte de suas tarefas de trabalho.

A estratégia para a condução dos estudos de caso foi adaptada de Gorschek, Garre, *et al.* (2006) e é descrita na Figura 52. As etapas destes estudos de caso, incluindo o levantamento das necessidades, a formulação do problema, a execução, a análise e discussão dos resultados e as lições aprendidas foram executadas em conjunto com representantes das empresas.

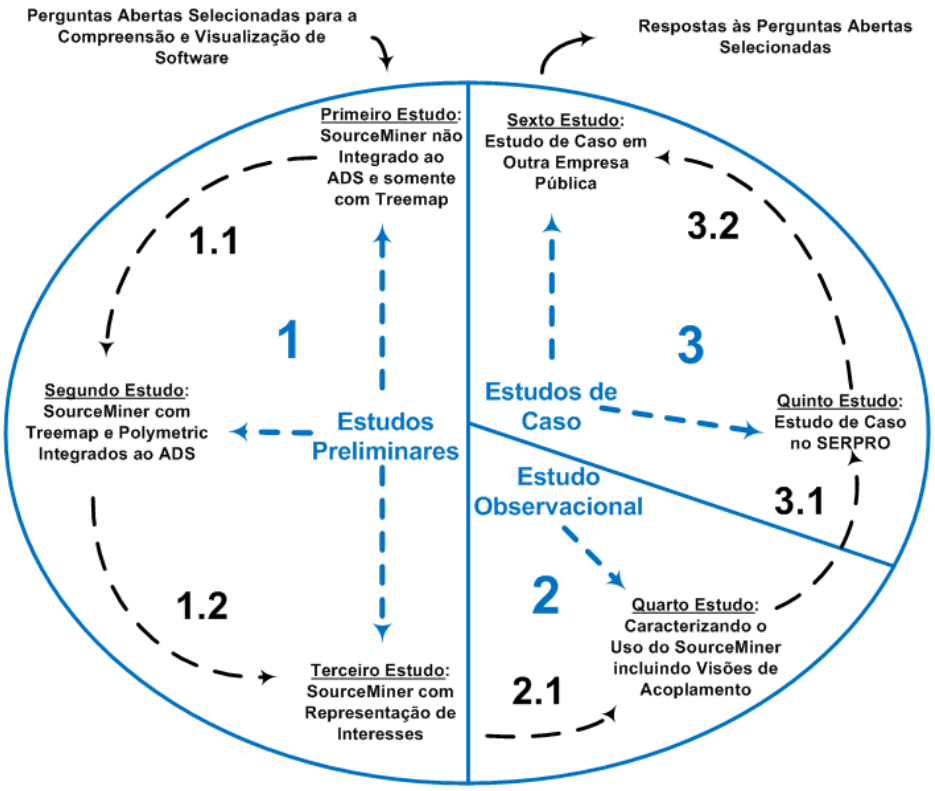

Figura 51 Estudos Experimentais Conduzidos

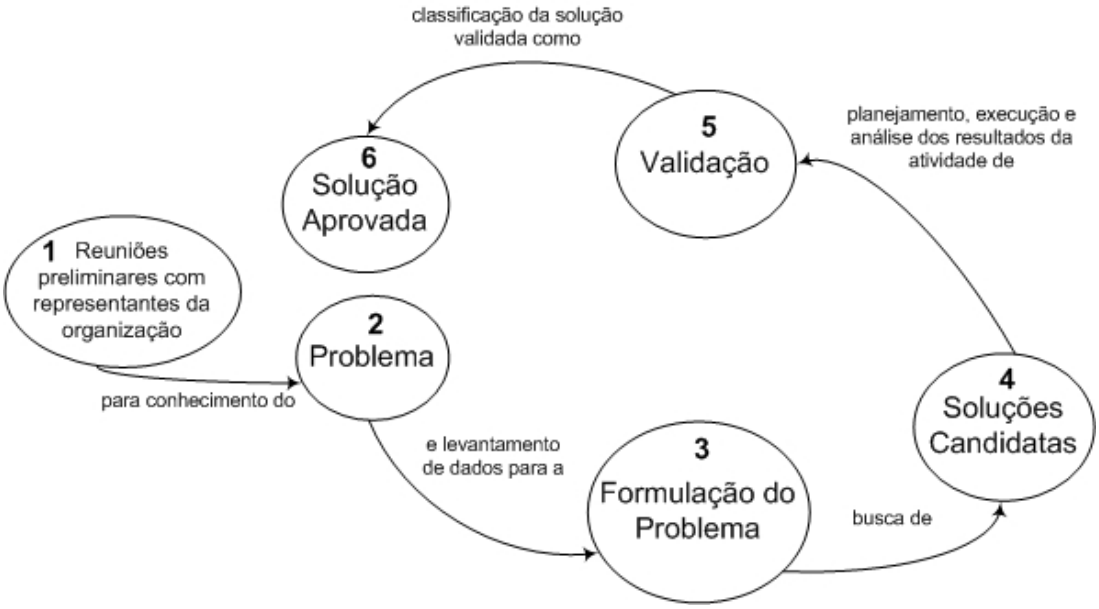

Figura 52 Estratégia para a Condução dos Estudos de Caso

Em ambos os casos, o ponto de partida dos estudos de caso foi o levantamento *in loco* de necessidades da organização para que, a partir daí, ocorresse a formulação do problema específico a ser tratado com o apoio do SourceMiner.

7.1 PRIMEIRO ESTUDO DE CASO: MUDANDO O FRAMEWORK DEMOISELLE

Para avaliar até que ponto o uso do SourceMiner é adequado às atividades do ciclo de vida de uma organização real, decidiu-se primeiro utilizá-lo em uma atividade de manutenção perfectiva de um arcabouço de software livre, o Demoiselle (Demoiselle, 2010). Este arcabouço, ou *framework*, é adotado e suportado pela Companhia de Processamento de Dados do Governo Brasileiro (SERPRO). Esta foi uma oportunidade para verificar como os recursos oferecidos pelo SourceMiner podem contribuir em atividades de engenharia de software, em um ciclo de vida real, de um de software de grau industrial, e sob forte utilização em uma organização de grande porte. O argumento a favor desta iniciativa é que esta interação pode fazer surgir situações que não se manifestariam em uma avaliação *in vitro* da abordagem.

7.1.1 Descrição da Versão do SourceMiner Utilizada no Estudo

A versão do SourceMiner utilizada neste estudo foi a da **etapa 3L** da Figura 3. Nesta versão estavam disponíveis todas as metáforas atualmente disponíveis no SourceMiner: mapa em árvores, visão polimétrica, grafo radial, *matriz de dependências*, visão tabular e grafo espiral egocêntrico. Também estavam disponíveis todos os recursos de interação descritos no Capítulo 4.

7.1.2 Planejamento do Estudo

Este estudo foi realizado em maio de 2010. Seguindo o gabarito GQM (Basili e Rombach, 1988), o objetivo do estudo foi:

Analisar o SourceMiner, um ambiente de visualização interativo

Com o propósito avaliar

Em relação à adequação do uso do SourceMiner em atividades de manutenção perfectiva (mudança de tecnologia) durante o ciclo de vida de uma aplicação

Do ponto de vista dos pesquisadores do ambiente SourceMiner.

No contexto de atividades de compreensão de um framework de desenvolvimento de aplicações web.

A Estratégia de Avaliação. A avaliação da adequação do SourceMiner ao ambiente industrial foi realizada através dos seguintes indicadores: (a) capacidade do SourceMiner em executar a atividade selecionada para o estudo de caso e alcançar os objetivos da atividade; e (b) o parecer dos participantes da organização em relação à viabilidade de uso do SourceMiner em atividades de compreensão de software, em especial na atividade selecionada.

A Estratégia de Planejamento e Execução. A execução deste estudo de caso foi realizada em cinco encontros. Na primeira reunião foram apresentados os recursos do SourceMiner e os principais resultados de avaliação obtidos até então. Na segunda reunião, o foco foi no reconhecimento e formulação do problema seguindo as etapas 1, 2 e 3 da Figura 52. A terceira e quarta reuniões foram dedicadas à execução, enquanto que a última reunião foi dedicada à análise dos resultados e lições aprendidas.

Participantes do Estudo de Caso. Este estudo foi executado com a participação de dois membros sênior da equipe de desenvolvimento do Demoiselle no SERPRO junto com o autor desta tese.

Uma das principais diferenças deste estudo em relação aos anteriores, é que antes havia uma lista de referência disponível para comparar as respostas fornecidas pelos

participantes com um oráculo. As características dos dois últimos estudos de caso deste capítulo não permitem a adoção de uma lista de referência.

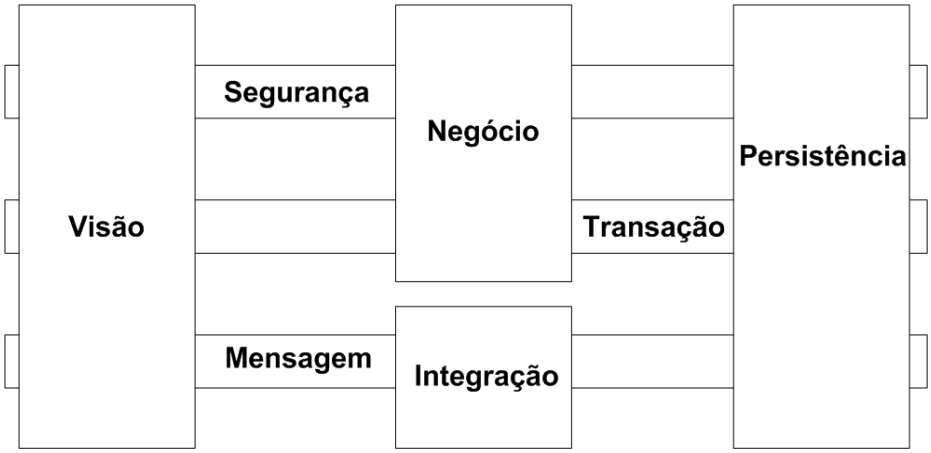

Figura 53 Arquitetura do Demoiselle (Demoiselle, 2010)

O Problema Abordado. O problema a ser analisado era a necessidade da execução de manutenções perfectivas no Demoiselle. A equipe de desenvolvimento tinha um conjunto de atividades de manutenção a ser realizado (Demoiselle SouceForge, 2010) envolvendo a avaliação do impacto na substituição de duas tecnologias relacionadas à injeção de dependências e à persistência.

O Objeto de Estudo. O objeto de estudo foi o Demoiselle Versão 1.1.2, um arcabouço para o desenvolvimento de sistemas de informações para a Web, escrito na linguagem de programação Java, e composto por 38 pacotes, 100 classes e 475 métodos. Apesar de ter código fonte aberto, ele foi desenvolvido, e ainda é largamente mantido, por funcionários do SERPRO. Somente nesta organização, o Demoiselle já foi usado para o desenvolvimento de mais de 30 aplicações. A arquitetura do Demoiselle é apresentada na Figura 53.

A Identificação de Oportunidades de Uso do SourceMiner. A equipe do Serpro relatou a necessidade de adoção de uma abordagem para apoiar a identificação de trechos de código que utilizam injeção de dependências e persistência, assim como da avaliação do impacto da modificação das tecnologias utilizadas para implementá-las no arcabouço. Os representantes da empresa argumentaram que apesar desta atividade poder ser realizada com

os recursos nativos oferecidos pelo ADS Eclipse, experiências de atividades anteriores no mesmo projeto revelaram que estes recursos não ofereciam o nível de abstração esperado, para a obtenção de uma visão abrangente de como estas tecnologias estavam distribuídas (e sendo utilizadas) no arcabouço. Este cenário apresentou uma ótima oportunidade de uso do SourceMiner.

A Formulação e o Tratamento do Problema. O problema identificado era uma necessidade real, pois foi estabelecida como meta pela equipe de gestão para a equipe de desenvolvimento da empresa. A meta era a substituição da tecnologia de persistência de *Java Persistence API* - ***JPA*** para JPA 2.0 (Yang, 2010) e de injeção de dependências de *AspectJ* (Kiczales, Hilsdale, *et al.*, 2001) para *Java Specification Recommendation* JSR 299 (Jendrock, Evans, *et al.*, 2010). Para a execução destas substituições seria necessária a identificação dos trechos de código que utilizavam cada uma destas tecnologias. Na seqüência, seria avaliado o impacto das possíveis mudanças para que finalmente elas pudessem ocorrer. A última etapa deveria ser a validação das modificações através da instanciação de uma aplicação exemplo a partir do Demoiselle.

O Conceito de Injeção de Dependências no Demoiselle. Para melhor contextualizar a avaliação do impacto da mudança de tecnologia de injeção de dependências foi necessário tornar uniforme o conceito de injeção de dependências para todos os participantes do estudo. O objetivo da injeção de dependências é oferecer um objeto específico, na realidade um montador, que popula os atributos em uma classe escutadora com implementação apropriada dos métodos de uma determinada interface (Jendrock, Evans, *et al.*, 2010). Isto significa que no lugar de implementar o código dependente, ele é “injetado” a partir de estrutura externa ao código. Isto é geralmente justificado através do princípio de Hollywood (Fayad e Schmidt, 1997; Fayad, Schmidt e Johnson, 1999) que diz “não nos chame que nós o chamaremos”. A injeção de dependências pode ser indicada para, dentre outras situações,: (i) injetar dados de configuração em um ou mais módulos da aplicação; (ii) injetar a mesma dependência em vários módulos; (iii) injetar diferentes implementações da mesma dependência. A equipe de desenvolvimento do SERPRO decidiu trocar a tecnologia de injeção de dependências que era baseada em *AspectJ* para aquela proporcionada pela *Java Specification Recommendation* JSR 299 (Jendrock, Evans, *et al.*, 2010).

7.1.3 Execução do Estudo

A primeira etapa da execução foi o que chamamos de preparação para o tratamento do problema. Esta etapa foi composta por três passos. O primeiro passo correspondeu ao treinamento dos profissionais da empresa no uso do SourceMiner pelo autor desta tese. O segundo passo consistiu na seleção de 13 interesses relevantes a serem mapeados no framework Demoiselle, incluindo os interesses associados a injeção de dependências e persistência. E, o terceiro passo foi a identificação e o mapeamento dos mesmos. A identificação e o mapeamento dos interesses proporcionaram conhecimento atualizado do framework, além de fornecer subsídios para a avaliação de propriedades importantes dos interesses selecionados tais como dedicação, espalhamento e entrelaçamento. Estas propriedades foram todas analisadas visualmente através do SourceMiner.

Dois participantes da equipe de desenvolvimento do Demoiselle auxiliados pelo autor desta tese identificaram e estabeleceram consenso em relação ao conjunto de 13 interesses considerados relevantes para a compreensão do framework. Em seguida passou-se para a etapa de mapeamento no código fonte usando o plug-in *Concern Mapper* (Robillard e Weigand-Warr, 2005). O mapeamento dos 13 interesses ocorreu em aproximadamente 7 horas. A partir do mapeamento dos interesses no código fonte foi feita a importação e apresentação visual dos mesmos no SourceMiner.

A próxima etapa foi a análise da modularidade dos interesses mapeados. Os interesses reconhecidos como importantes para a análise foram *injection*, *JDBC*, *JPA*, *Hibernate* (Bauer e King, 2006) e *Persistence*. Sendo o primeiro em decorrência da mudança de tecnologia de injeção de dependência, enquanto que os outros quatro interesses eram relevantes para a mudança de tecnologia de persistência. O mapeamento dos outros oito interesses, apesar de não estarem diretamente relacionados com a mudança de tecnologia, também foram úteis para a análise de propriedades de dedicação (quanto de um determinado interesse está sendo realizado em um determinado módulo), entrelaçamento (o número de interesses presentes em um determinado módulo) e espalhamento (o grau em que um determinado interesse está presente em vários módulos).

Durante a análise, primeiro fez-se a avaliação do impacto da mudança da tecnologia de injeção de dependências e depois foi feita a avaliação do impacto da mudança da tecnologia de persistência. Cada avaliação consistiu na análise do framework usando o SourceMiner, complementado pelo acesso ao código fonte dos trechos de código para a realização das modificações necessárias.

A última etapa desta fase consistiu na instanciação de uma aplicação a partir do Demoiselle para verificar se de fato as funcionalidades da nova versão do framework foram corretamente implementadas.

A análise de impacto, a realização e a validação das mudanças das duas tecnologias ocorreram em aproximadamente 5 horas.

7.1.4 Análise e Discussão dos Resultados

Os participantes da equipe do Demoiselle relataram que o planejamento, discussão, execução e avaliação das atividades do estudo de caso possibilitaram a eles conhecer uma abordagem que poderia ser utilizada em outras etapas do ciclo de vida do framework e não somente na avaliação de impacto da substituição de tecnologias. Um dos participantes comentou que “o SourceMiner mostrou de forma eficiente como visualmente identificar os pontos da aplicação relacionadas a um ou mais interesses, assim como analisá-los sob várias perspectivas. Planejamos estender o uso da ferramenta para comunicar como ocorreram as atividades de avaliação de impacto de mudança de tecnologia no Demoiselle, assim como as oportunidades de melhorias encontradas ao longo deste processo”.

Também foi relatado o rápido aprendizado no uso das interfaces e dos recursos de interatividade. A efetividade no uso do AIMV para a seleção foi evidente para os participantes:“o ambiente visual oferece uma forma objetiva para a identificação de discrepâncias na aplicação analisada”.

Os participantes relataram que já trabalharam em atividades similares em outros projetos. Segundo eles, ficou evidente a capacidade que a apresentação visual da aplicação

segundo os interesses oferece. Isto apoiou de forma considerável a construção dos modelos mentais acerca do framework Demoiselle.

Ainda segundo os participantes, a atividade de mapeamento de interesses serviu de apoio para o entendimento de como estes estavam sendo utilizados na estrutura do framework. Esta atividade também serviu para auxiliar na construção de modelos mentais dos representantes da empresa a respeito do Demoiselle, principalmente pelo fato de ter sido reservado horário específico para a atividade de mapeamento. A cada interesse mapeado iniciava-se uma discussão em relação aos trechos de código analisado e de detalhes da realização de um ou mais interesses nestes trechos. A partir da atividade de mapeamento em nível de linha de código foram relatadas abstrações a respeito da forma como os interesses estavam realizados no framework, contribuindo, a cada momento, para a evolução do modelo mental a respeito do framework analisado. Isto ocorreu porque foram percorridos todos os trechos de código relacionados a cada um dos interesses, oportunidade em que eles discutiram entre si a forma de realização dos mesmos.

O resultado foi o conhecimento de como cada um dos interesses estavam relacionados. Em função do número de interesses (13) e do tamanho do framework, os participantes revelaram que mesmo tendo evoluído seus modelos mentais sobre o objeto de estudo ainda não teriam subsídios para a identificação dos pontos onde as modificações deveriam ser realizadas e nem qual seria o impacto destas modificações. Isto abriu caminho para o uso da representação visual dos interesses, pois as propriedades de dedicação, espalhamento e entrelaçamento não puderam ser analisadas de forma adequada somente com os recursos do ADS.

Figura 54 Injeção de Dependências no Mapa em Árvores

Com relação ao uso do SourceMiner para a análise dos interesses, a ordem de análise dos interesses foi decidida pelos participantes. Optou-se por primeiro analisar a injeção de dependência e depois a persistência.

Inicialmente foi utilizado mapa em árvores para identificar os métodos que realizam o interesse estudado. A Figura 54 apresenta um cenário visual inicial utilizado durante a análise da injeção de dependências.

Na figura acima é possível identificar os métodos cujos trechos de código deveriam ser analisados com relação à injeção de dependências. Além disso, é possível visualizar também o espalhamento do referido interesse. Através da configuração do mapa em árvores para a representação dos outros interesses mapeados para o Demoiselle foi possível analisar a propriedade de entrelaçamento do interesse selecionado com os demais.

As outras visões do SourceMiner foram então utilizadas para a análise de impacto e priorização das modificações. A Figura 55 apresenta o grafo radial que foi utilizado na identificação das classes que têm relação de dependência com as classes associadas à injeção de dependências (coloridas em preto). Adicionalmente, a Figura 56 mostra a visão tabular e o grafo espiral egocêntrico. Estas visões foram utilizadas para auxiliar na decisão da ordem de prioridade das modificações.

Os participantes decidiram que seria dada prioridade de preferência de modificação às classes com maior força de acoplamento. Através da visão tabular foi possível identificar estas classes para que fossem modificadas uma a uma.

Os cenários representados pela Figura 54, Figura 55 e Figura 56 mostraram-se suficientes para a identificação dos trechos de código candidatos à alteração e depois confirmados através da análise direta no próprio código fonte. Como dois dos participantes do estudo conheciam o framework a fundo, a identificação dos trechos de código a partir das representações visuais foi praticamente imediata. A diferença desta etapa em relação à anterior onde foi feita o mapeamento é que a disponibilidade das representações visuais dos interesses tornou bem mais efetiva a construção dos modelos mentais a respeito da aplicação analisada. Antes, sem o SourceMiner, isto não era possível em função do tamanho do framework, do número de relacionamentos a ser analisado através do código fonte e número de interesses a ser analisado.

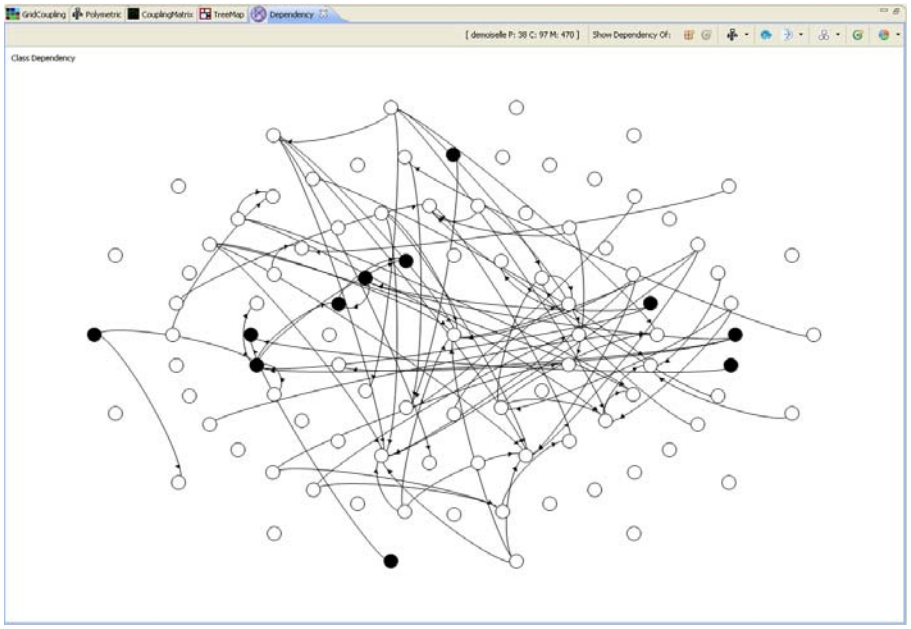

Figura 55 Injeção de Dependências na Visão de Grafos Radiais

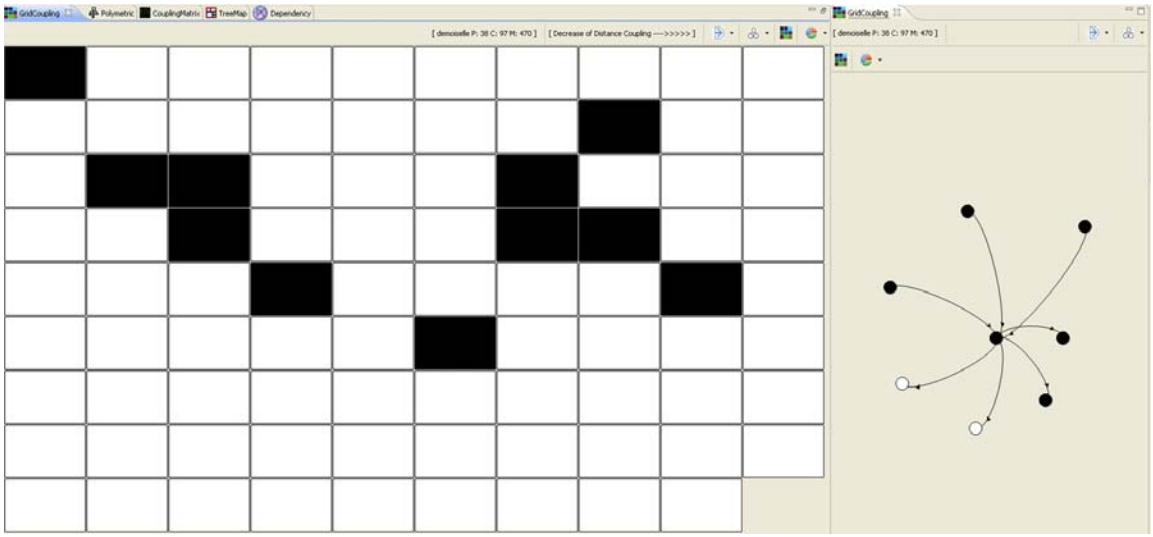

Figura 56 Injeção de Dependências nas Visões Tabular e Grafo

A situação da identificação dos pontos relacionados à persistência foi análoga à descrita para injeção de dependências. A diferença é que agora a análise considerou vários interesses – *JDBC, JPA, Persistence Controller* e *Hibernate* – para a avaliação do impacto na substituição da tecnologia de persistência.

Após a identificação dos trechos de código a serem substituídos e da compreensão destes trechos no contexto dos interesses injeção de dependências e persistência, partiu-se para a substituição propriamente dita. Neste momento, à medida que os trechos de código eram modificados, também eram feitas atualizações no mapeamento dos interesses com o *ConcernMapper* de forma que a nova versão do framework pudesse também ter os seus interesses visualizados.

A terceira e última etapa consistiu na validação das substituições através da instanciação de uma aplicação exemplo a partir dela. Para cada uma das substituições de tecnologia foi feita uma instanciação de forma bem sucedida e depois uma nova instanciação com as duas substituições simultaneamente.

Em resumo, os seguintes resultados foram obtidos neste estudo:

a) Os especialistas identificaram e analisaram visualmente como os interesses estavam relacionados entre si. Como exemplo, a Figura 54 mostra classes que foram afetadas pelo interesse injeção de dependências usando o grafo radial de dependência entre classes.

Baseado nesta visão, os especialistas identificaram as classes afetadas por este último interesse e como elas estavam relacionadas entre si.

b) Os especialistas, após os resultados descritos no item anterior, conseguiram planejar e executar a mudança da implementação da injeção de dependências e de persistência de forma bem sucedida.

Em relação à estratégia de avaliação adotada, constatou-se que o SourceMiner foi adequado ao uso no ambiente industrial, tendo em vista que foi executada de forma bem sucedida a atividade selecionada para o estudo de caso para que fossem alcançados os objetivos da atividade. Além disso, os participantes da organização relataram a viabilidade do uso do SourceMiner em atividades de compreensão de software, em especial a atividade selecionada para este estudo.

Ameaças à Validade do Estudo.

Para a realização da análise qualitativa, o autor desta tese observou a execução das atividades pelos participantes da empresa. Este fato pode ter conduzido a situações em que o observador tenha influenciado no comportamento dos participantes. Entretanto, o fato de ter sido estabelecida uma meta de forma objetiva, incluindo a avaliação de impacto da substituição das tecnologias e a sua validação através da instanciação de uma aplicação a partir da nova versão do framework foi decisivo para demonstrar que as atividades realizadas pelo SourceMiner foram efetivas.

7.1.5 Lições Aprendidas

Com a realização deste estudo pode-se verificar a adequação do SourceMiner a atividades do ciclo de vida do software em um ambiente industrial. Neste estudo verificou-se que as atividades de seleção e mapeamento dos interesses contribuíram para a construção dos modelos mentais a respeito do framework. Mesmo sendo os participantes especialistas na

aplicação analisada, eles relataram refinamento do conhecimento da aplicação através das atividades de seleção e mapeamento dos interesses. Eles comentaram que estas atividades forneceram um novo caminho para o aprofundamento do conhecimento da aplicação. Este caminho passa pela seleção e mapeamento visual dos interesses mais importantes do framework, e chega à análise seqüencial do código fonte na perspectiva de cada um destes interesses. Esta abordagem foi considerada altamente didática e efetiva pelos participantes. Esta efetividade foi confirmada durante o uso e manipulação das representações visuais dos interesses.

A formalização desta abordagem pode fornecer uma metodologia interessante para a análise, compreensão e evolução de sistemas de software. O conhecimento tácito obtido em atividades anteriores de evolução do software é refinado e explicitado através do mapeamento de interesses aprofundado pela análise visual detalhada de como estes interesses estão de fato sendo realizados no sistema.

A possibilidade de análise dos registros coletados pela monitoração automática com a própria equipe do Demoiselle foi também levantada neste estudo. Os participantes do estudo sugeriram não somente o uso dos registros para a identificação de estratégias de uso das metáforas visuais e dos recursos do SourceMiner, mas também para apoiar a transferência de conhecimentos entre os membros de uma equipe. Por exemplo, algum tipo de representação visual que apresentasse o que foi feito durante a substituição de tecnologia de um determinado interesse poderia ser reutilizada para treinar novos participantes da equipe.

Uma forma de fazer isto em versões futuras é através da inclusão de comentários nas representações visuais para registrar as etapas de execução de uma determinada atividade. Por exemplo, ao utilizar a visão tabular para a identificação das classes com maior força de acoplamento, o executor da atividade poderia inserir comentários informando quais classes já foram modificadas. Estes comentários também seriam registrados pelo serviço de monitoração do ambiente. Com a construção de uma base ampla de comentários, membros mais experientes da equipe poderiam selecionar os comentários e as ações consideradas mais relevantes para certas atividades. Membros menos experientes utilizariam o SourceMiner com estes comentários e teriam acesso às seqüências de ações para a realização da atividade. A inovação do uso desta abordagem estaria na apresentação dos comentários no ambiente

baseado em múltiplas perspectivas visuais e não no código fonte como tem sido proposto em outros trabalhos (Oezbek e Prechelt, 2007; Alwis e Murphy, 2006).

7.2 SEGUNDO ESTUDO DE CASO: DESVIOS ARQUITETURAIS DE SOFTWARE

O próximo passo na avaliação do SourceMiner visou aprofundar o conhecimento a respeito da adequação do seu uso no contexto industrial. Quanto mais refinados os ajustes no ambiente do SourceMiner e também na metodologia adotada para seu uso, maior a probabilidade de sua aceitação no contexto industrial (Shull, Carver e Travassos, 2001). O estudo de caso foi realizado em outra empresa brasileira de grande porte. Em função do termo de confidencialidade adotado para a execução do estudo, não será possível a divulgação do seu nome. Apesar de TI não ser a atividade fim desta empresa, ela possui várias unidades de desenvolvimento de software, espalhadas por todo o país.

7.2.1 Descrição da Versão do SourceMiner Utilizada no Estudo

Da mesma forma que no estudo anterior, a versão do SourceMiner utilizada neste estudo foi a da etapa 3L da Figura 3. Nesta versão estavam disponíveis todas as metáforas atualmente disponíveis no SourceMiner: mapa em árvores, visão polimétrica, grafo radial, *matriz de dependências*, visão tabular e grafo espiral egocêntrico. Também estavam disponíveis todos os recursos de interação descritos no Capítulo 4.

7.2.2 Planejamento do Estudo

Este estudo foi realizado em setembro de 2010. Seguindo o gabarito GQM (Basili e Rombach, 1988), o objetivo do estudo foi:

Analisar o SourceMiner, um ambiente de visualização interativo

Com o propósito de avaliar

Em relação à adequação do uso do SourceMiner em atividades de verificação de conformidade de sistemas em relação a uma arquitetura de referência

Do ponto de vista dos pesquisadores do ambiente SourceMiner

No contexto de atividades de avaliação de desvios arquiteturais de aplicações web.

A Estratégia de Avaliação. Da mesma forma que no estudo de caso anterior, a avaliação da adequação do SourceMiner ao ambiente industrial foi realizada através dos seguintes indicadores: (a) capacidade do SourceMiner em executar a atividade selecionada para o estudo de caso e alcançar os objetivos da atividade; e (b) parecer dos participantes da organização em relação à viabilidade de uso do SourceMiner em atividades de compreensão de software, em especial a atividade selecionada para este estudo.

A Estratégia de Planejamento e Execução. A execução deste estudo de caso foi realizada em quatro encontros. Na primeira reunião foram apresentados os recursos do SourceMiner e os principais resultados já obtidos com a ferramenta. Na segunda reunião o foco foi no conhecimento e formulação do problema seguindo as etapas 1, 2 e 3 da Figura 47. A terceira reunião foi dedicada à execução, enquanto que a última reunião foi dedicada à análise dos resultados e lições aprendidas.

Os Participantes do Estudo. O estudo contou com a participação de três profissionais da empresa e também contou com a participação do autor desta tese. Na fase de mapeamento de interesses contou-se ainda com a colaboração de dois alunos de doutorado, um da UFBA e outro da PUC-Rio.

O Problema Abordado. O problema a ser analisado era a necessidade de avaliar se as aplicações web desenvolvidas na organização estavam de fato seguindo a arquitetura de

referência original definida pela organização. Um conjunto de três aplicações foi selecionado para análise e comparação com a estrutura da aplicação web definida como padrão pela organização.

O Objeto de Estudo. O objeto de estudo foram aplicações web desenvolvidas em linguagem Java. Três aplicações, denominadas neste estudo de aplicações X, Y e Z, foram selecionadas para a análise. A estrutura de referência da organização para o desenvolvimento de aplicações web, denominada neste estudo de aplicação de referência, também foi analisada. Esta estrutura tinha 25 pacotes, 125 classes e 1346 métodos e era, na realidade, uma aplicação Java a partir da qual as outras atividades deveriam ser desenvolvidas.

A Identificação de Oportunidades de Uso do SourceMiner. A equipe da empresa tinha dúvidas se a utilização de uma aplicação de referência era suficiente para o desenvolvimento de aplicações consistentes em termos de arquitetura e reuso de componentes. Ela procurava uma abordagem que permitisse a análise de desvios arquiteturais das aplicações sendo desenvolvidas a partir da aplicação de referência. Segundo os participantes da empresa, a avaliação tão somente baseada no código fonte e nos recursos disponibilizados pelo ADS Eclipse mostrou-se não apropriada para esta finalidade. Antes da atividade deste estudo de caso, os participantes relataram que estavam avaliando a possibilidade do uso de métricas e de diagramas UML para esta finalidade. Entretanto, verificaram que informações importantes como os interesses realizados por cada entidade não seriam representadas nos diagramas. Este cenário apresentou uma ótima oportunidade de uso do SourceMiner.

Este estudo de caso, da mesma forma que o anterior não teve disponível uma lista de referência para comparação, tendo em vista que não havia conhecimento prévio dos artefatos a serem analisados.

A Formulação e o Tratamento do Problema. O problema a ser analisado era a necessidade de avaliar se as aplicações web desenvolvidas estavam de fato seguindo a arquitetura original definida pela organização. Outra questão a ser analisada seria até que ponto a arquitetura original disponibilizada permitia a ocorrência de possíveis desvios.

A Arquitetura Disponibilizada pela Organização para o Desenvolvimento de Aplicações Web. A arquitetura disponibilizada na realidade era uma aplicação com funcionalidades comuns aos sistemas de negócio da empresa (vide Figura 57). Esta aplicação

utiliza as seguintes tecnologias: *JavaServer Faces* (JSF) (Geary e Horstmann, 2010) para a camada de apresentação; *Spring* (Walls, 2007) para promover a integração da camada de apresentação e a camada de negócios e *Java Persistence API* (JPA) (Yang, 2010) para a camada de persistência juntamente com o Hibernate (Bauer e King, 2006).

Segundo a equipe que definiu esta solução, o principal argumento do uso desta estrutura é o ganho de produtividade obtido através do direcionamento do esforço de desenvolvimento para as regras de negócio. A equipe também argumentou que a solução proposta implicaria na padronização das tecnologias adotadas e na forma como as mesmas seriam utilizadas. Apresentou-se a expectativa de que as aplicações desenvolvidas teriam estrutura compatível e aderente com a da referência disponibilizada, fato que facilitaria as atividades de manutenção.

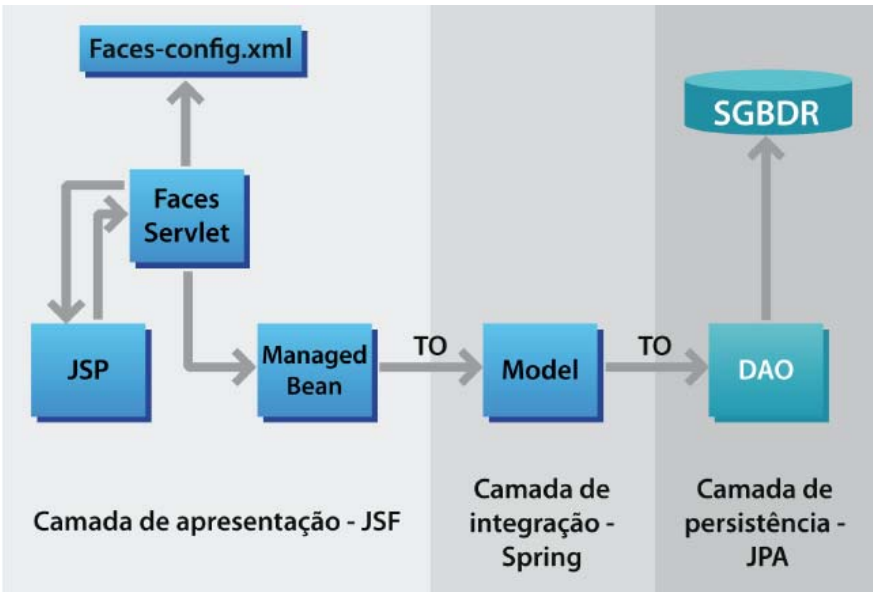

Figura 57 Arquitetura da Aplicação de Referência

7.2.3 Execução do Estudo

Nesta etapa do estudo foram avaliadas as soluções possíveis para a análise de ocorrências de desvios arquiteturais nas aplicações desenvolvidas na empresa. A abordagem para tratamento do problema foi composta por três passos iniciais. O primeiro passo correspondeu ao treinamento dos profissionais da empresa no uso do SourceMiner pelo autor desta tese. O segundo passo consistiu na seleção dos interesses relevantes. para a arquitetura de referência. O terceiro passo consistiu nos mapeamentos dos interesses na aplicação de referência e nas aplicações a serem analisadas. Estes três passos se beneficiaram fortemente das lições aprendidas no estudo de caso anterior.

O quarto passou consistiu na comparação das representações visuais de cada aplicação com as representações visuais da aplicação de referência. Durante este passo, foram realizadas entrevistas com os desenvolvedores para que fossem obtidas informações sobre o que motivou os desvios e modificações detectadas durante a análise.

O quinto e último passo do estudo consistiu no levantamento de oportunidades de melhorias do SourceMiner. Identificando-se como ele poderia incluir funcionalidades para melhor apoiar a atividade de comparação das aplicações desenvolvidas na organização.

A seleção dos interesses relevantes para a análise foi feita pela equipe da empresa. O autor desta tese apresentou como sugestão inicial a relação de interesses mapeados no framework do estudo anterior e os interesses selecionados ao final foram: interface gráfica do usuário, tratamento de exceções, persistência, negócio e segurança.

Após o mapeamento dos interesses, representações visuais dos mesmos foram utilizadas para analisar as propriedades de entrelaçamento, espalhamento e dedicação nas aplicações desenvolvidas.

Um fator diferente dos outros estudos é que agora havia a necessidade de comparar aplicações diferentes entre si.

Verificou-se que a versão do SourceMiner utilizada no estudo não possibilitava a apresentação simultânea de visões de aplicações diferentes através de uma mesma metáfora visual. Para contornar esta questão, os participantes utilizaram estações diferentes para a análise de cada aplicação. Posicionando as estações lado a lado, foi possível analisar

comparativamente e em detalhes as aplicações e discutir entre os participantes do estudo as características de cada uma delas.

A comparação entre as versões das aplicações para a identificação de situações de desvios arquiteturais durou aproximadamente 2 horas.

7.2.1 Análise e Discussão dos Resultados

A visão polimétrica se destacou no apoio à análise comparativa das aplicações. Ela foi efetiva na apresentação de situações de disparidades de tamanho nas classes quando comparadas com a aplicação de referência.

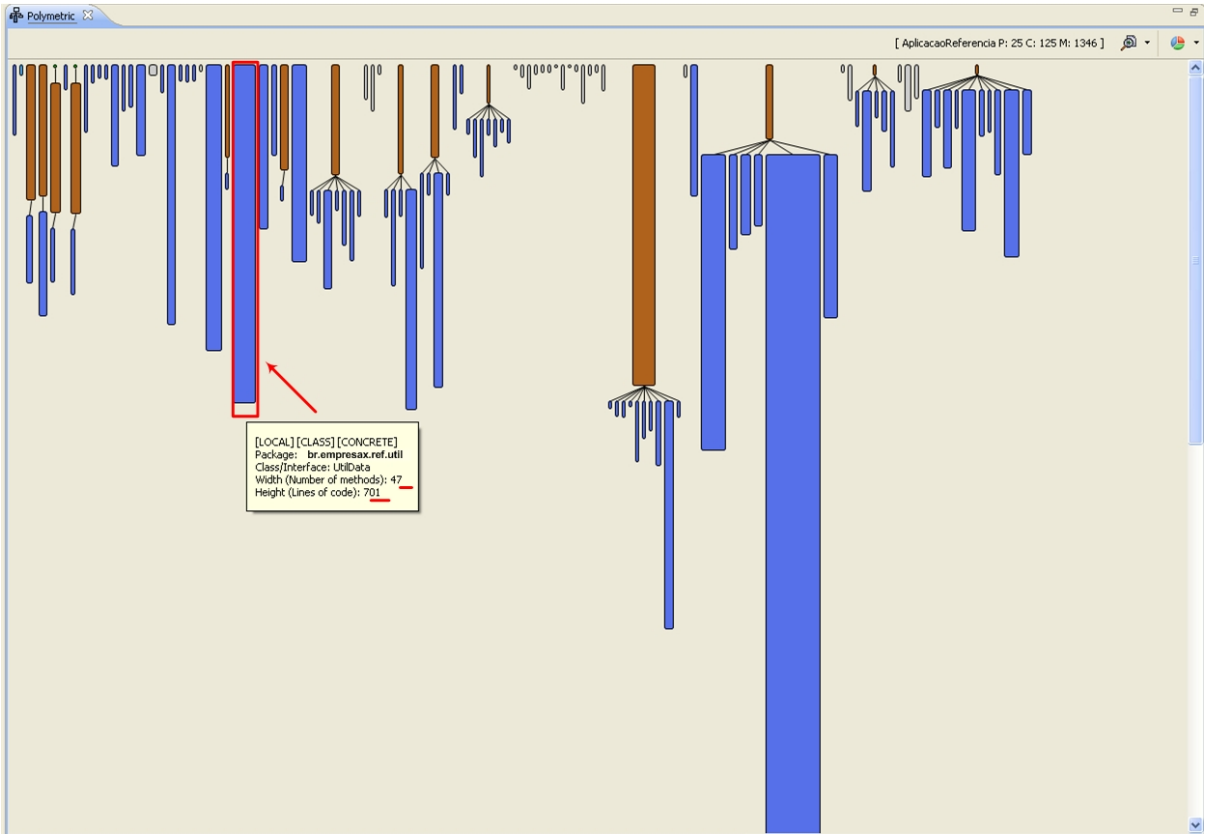

Figura 58 Aplicação de Referência na Visão Polimétrica

Figura 59 Aplicação X na Visão Polimétrica

Como exemplo, a Figura 58 e a Figura 59 mostram a comparação das visões polimétricas entre a aplicação de referência e a aplicação X, respectivamente. No exemplo, as figuras apontam que a classe *UtilData* aumentou consideravelmente de tamanho, passando de 701 linhas e 47 métodos na aplicação de referência para 2089 linhas e 160 métodos na aplicação X.

A questão evidenciada não foi somente referente ao aumento de tamanho da classe citada, mas da natureza deste aumento.

Esta é uma classe utilitária e, por conseguinte, não deveria sofrer alterações na aplicação X e sim na aplicação de referência. A forma como foi feita a alteração viola os princípios do uso da aplicação de referência. Em entrevista com os desenvolvedores que realizaram as modificações da classe UtilData na aplicação X, foi relatado que diversas modificações foram realizadas na classe citada para a inclusão de “novas funcionalidades utilitárias”, a exemplo de um recurso de calendário incluído.

Ainda na entrevista, não foi justificado o motivo de tal necessidade não ter sido reportada à equipe de desenvolvimento da aplicação de referência para que tal funcionalidade fosse incluída na classe UtilData daquela aplicação. Caso esta necessidade tivesse sido reportada, outras aplicações da organização poderiam também utilizar os recursos incluídos, além de ser promovida a padronização no uso dos recursos.

As outras aplicações apresentaram situações similares de aumento de tamanho da classe UtilData, indicando uma clara limitação na abordagem de reuso adotada na organização.

Os participantes também utilizaram outras metáforas visuais disponibilizadas nas perspectivas para analisar as aplicações. Por exemplo, através da visão dos mapas em árvores foi possível verificar que as aplicações mantinham a modularidade dos interesses constatada na aplicação de referência.

A análise final indicou que a abordagem adotada pela organização gera oportunidades de não reuso de código, além de contribuir para a degeneração da arquitetura original proposta pela aplicação de referência.

Em relação à estratégia de avaliação adotada, constatou-se que o SourceMiner foi adequado ao uso no ambiente industrial, tendo em vista que foi executada de forma bem sucedida a atividade selecionada para o estudo. Além disso, os participantes da organização endossaram a viabilidade do uso do SourceMiner em atividades de compreensão de software, em especial para a atividade selecionada para este estudo. Após o estudo o SourceMiner foi adotado como ambiente para a caracterização das aplicações desenvolvidas na organização.

Ameaças à Validade do Estudo

As aplicações analisadas, incluindo a estrutura de referência adotada pela empresa podem ser caracterizadas como de tamanho pequeno-médio e a análise realizada seria mais complexa em aplicações de maior porte. O argumento para tratar esta ameaça é que as situações de desvios arquiteturais poderiam ser identificadas desde que ocorresse um ajuste na configuração do ambiente através dos recursos de interação, filtragem e zoom geométrico e semântico. Apesar da natureza típica das aplicações, a adoção de aplicações de referências pode ser dependente do seu domínio. Outros estudos precisam ser realizados em outros contextos para que os resultados sejam mais bem generalizados.

7.2.2 Lições Aprendidas

Neste estudo verificou-se que o SourceMiner foi efetivo para a análise de desvios de implementação das aplicações web a partir da estrutura original disponibilizada pela empresa. Partindo das metáforas visuais e dos recursos de interatividade disponibilizados foi possível identificar ocorrências de desvios nas aplicações desenvolvidas que contrariavam o objetivo do uso da aplicação de referência.

Foi verificado que as aplicações desenvolvidas através da solução vigente na época do estudo de caso sempre apresentam pontos de desvios, na maioria das vezes, adição de novas funcionalidades que não estavam diretamente relacionadas às regras de negócio e que, por este motivo, deveriam ser incluídos na aplicação de referência. As evidências de desvios identificadas neste estudo foram relatadas para a equipe de gestão do desenvolvimento de aplicações para que fosse avaliada a possibilidade de nova estratégia para a disponibilização de uma estrutura de referência a exemplo de um framework a partir do qual novas aplicações poderiam ser derivadas e instanciadas na organização.

Neste estudo os interesses assumiram papel secundário. Entretanto, após o estudo, a empresa tem utilizado o SourceMiner para a caracterização das aplicações desenvolvidas para análise de realização de interesses e das suas propriedades (dedicação, entrelaçamento e espalhamento) a partir das informações disponibilizadas pelos casos de uso.

Uma importante lição aprendida neste estudo foi a necessidade de fornecer funcionalidade para visualização de mais de um projeto de forma simultânea no SourceMiner. Isto facilitaria a análise comparativa dos projetos em uma única estação. Outra possibilidade identificada foi a utilização das visões do SourceMiner para análise diferencial de aplicações. Nesta abordagem, as decorações das visões, tais como cores dos elementos visuais, poderiam ser utilizadas para realçar as diferenças entre a aplicação analisada (mostrada nas visões) e uma aplicação de referência (utilizada no cálculo diferencial de cores). Esta abordagem está agora sendo desenvolvida no escopo de outra tese de doutorado para análise de evolução de software (Novais, Carneiro e Mendonça, 2011).

7.3 CONCLUSÃO DO CAPÍTULO

Neste capítulo foram apresentados e discutidos os estudos de caso realizados para avaliação final desta tese de doutorado. Seguindo a abordagem proposta por Shull e colegas (Shull, Carver e Travassos, 2001), a primeira etapa, descrita no Capítulo 3, consistiu na condução de estudos de viabilidade com o objetivo de adquirir conhecimento necessário a respeito do apoio de um ambiente interativo a atividades de compreensão de software. O foco foi no refinamento da tecnologia e também a geração de novas hipóteses a serem investigadas nos estudos seguintes (Mafra, Barcelos e Travassos, 2006). Os resultados desta primeira etapa indicaram a viabilidade da proposta e novas funcionalidades foram incluídas ao SourceMiner com o objetivo de tornar o seu uso mais efetivo. Ao final desta etapa, o SourceMiner, conforme descrito na Figura 3, apresentava as seguintes funcionalidades:

- Integração plena com o ADS Eclipse e utilização da infra-estrutura por ele disponibilizada;
- Recursos interativos como filtragem, zoom geométrico e semântico e acesso ao código fonte a partir das representações visuais;
- Uso do conceito de perspectivas através das perspectivas de pacote-classe-método e de hierarquia de herança;
- Metáforas visuais do mapa em árvores e da visão polimétrica;
- Serviço de monitoração automática de ações executadas pelo usuário.

A segunda etapa, descrita no Capítulo 6, foi composta por um estudo observacional com a finalidade de melhorar o entendimento em relação aos recursos oferecidos pelo SourceMiner com o objetivo de refiná-los ainda mais. Ao final desta etapa, existiam evidências do apoio do SourceMiner à análise da modularidade de interesses e à identificação de anomalias de modularidade em aplicações Java. O SourceMiner, conforme descrito na Figura 3, apresentava as seguintes novas funcionalidades:

- Representação visual de interesses;
- Inclusão da perspectiva de acoplamento;

- Inclusão das metáforas visuais grafos radiais de acoplamento, visão tabular, grafo espiral egocêntrico e matrizes de acoplamento;
- Recursos interativos como filtragem, zoom geométrico e semântico e acesso ao código fonte a partir das representações visuais;

A última etapa da avaliação, descrita neste capítulo, consistiu na execução de dois estudos de caso na indústria com o objetivo de caracterizar o uso do SourceMiner para identificar até que ponto a tecnologia proposta é útil e pode ser integrada em um ambiente industrial real. Nos dois estudos de caso, foram alcançados os objetivos estabelecidos pelos participantes da indústria através do uso do SourceMiner. Também foram identificadas oportunidades de melhorias que estão sendo conduzidas no escopo de trabalhos em andamento ou ainda planejadas como trabalhos futuros.

Capítulo

8

Este capítulo apresenta a conclusão desta tese de doutorado, juntamente com as suas contribuições. Também são apresentados os trabalhos em andamento e as próximas atividades previstas com o SourceMiner.

8 CONCLUSÃO E PERSPECTIVAS FUTURAS

Esta tese apresentou um ambiente interativo para a visualização de software baseado em múltiplas perspectivas chamado SourceMiner. As visões disponibilizadas estão integradas entre si e também com o ADS. Além disso, elas podem ser ajustadas dinamicamente para a realização de atividades de Engenharia de Software.

O foco do ambiente é na visualização estática do software. Ele não analisa a sua execução, mas sim a sua estrutura e os relacionamentos entre as entidades que o compõem. Os elementos básicos desta estrutura são pacotes, classes, métodos, e os seus respectivos atributos. O modelo adotado na concepção do ambiente pode ser aplicado a outros tipos de visualização de software tais como na sua representação dinâmica e de sua evolução.

Foi adotada uma abordagem iterativa e incremental nas etapas de concepção, desenvolvimento e avaliação do ambiente. O modelo concebido inicialmente foi ampliado e aprofundado a cada novo estudo e etapa de evolução.

Ao longo dos capítulos desta tese foram apresentados conceitos de compreensão de software, visualização de informação e visualização de software considerados relevantes para a contextualização deste trabalho e para o entendimento do SourceMiner como um ambiente interativo baseado em múltiplas visões (AIMV). A arquitetura e o modelo conceitual adotados para o ambiente também foram apresentados e discutidos. Foram

introduzidos exemplos de uso das perspectivas e das visões do SourceMiner para contextualizá-las no uso para atividades de compreensão de software.

Nos capítulos 3, 6 e 7 foram apresentados estudos conduzidos com o SourceMiner. O conjunto de estudos foi composto por estudos preliminares (Capítulo 3), um estudo observacional (Capítulo 6) e dois estudos de caso na indústria (Capítulo 7).

8.1 CONTRIBUIÇÕES

Esta tese apresenta as seguintes contribuições:

- Um modelo de referência para visualização de software baseado em múltiplas perspectivas;
- Uma arquitetura para um ambiente de visualização de software suportando o modelo proposto;
- A concepção de duas novas metáforas para visualização de software, nominalmente: a visão tabular de força de acoplamento e o grafo espiral egocêntrico para representação de propriedades de acoplamento;
- O desenvolvimento de um ambiente interativo baseado em múltiplas visões (**AIMV**) integrado ao ADS Eclipse para apoio às atividades de compreensão de software;
- Uma solução para a representação visual de interesses;
- A concepção e desenvolvimento de um serviço de monitoração automática de ações executadas para apoiar a realização de estudos de perfil de uso do ADS Eclipse e das múltiplas visões desenvolvidas.

8.2 LIMITAÇÕES

Diversas limitações foram identificadas ao longo deste trabalho, especialmente durante a realização dos estudos relatados no capítulo 5.

A primeira limitação é que o SourceMiner não faz a atualização contínua dos cenários visuais a partir de cada modificação executada no código fonte. Cenários visuais precisam ser totalmente remontados a partir da seleção de um item de menu sobre a pasta do projeto em análise no ADS. Também não foi implementada a coordenação plena entre as visões no ambiente. Por este motivo, nem todas as ações de interação são propagadas para todas as visões em um dado instante. Por exemplo, a aplicação de um zoom (geométrico ou semântico) em uma visão não é propagada para as demais. Nos grafos radiais de acoplamento, ainda não foi implementada navegação entre os níveis de acoplamento de pacotes, classes e métodos como foi implementado nas matrizes de dependências.

Apesar destas limitações, o ambiente proposto apoiou de forma efetiva a execução de diferentes atividades de compreensão de software in-vitro e in-vivo.

8.3 ESTRATÉGIA DE USO E EVOLUÇÃO DO PROJETO SOURCEMINER

O trabalho desenvolvido nesta tese prevê a utilização do SourceMiner como arcabouço experimental, a sua disponibilização como projeto de código aberto e o seu uso para fortalecer parcerias com outros grupos de pesquisa.

A área de visualização de software ainda está em um estágio inicial. O grande esforço atual é no desenvolvimento e avaliação de metáforas visuais e seus recursos (Koschke, 2003) (Storey, 2006) (Penta, Kurt e Kraemer, 2007). O uso combinado de metáforas e sua integração com os recursos do ADS ainda precisam ser mais bem explorados. Por este motivo, o SourceMiner também foi também desenvolvido como um arcabouço

experimental para a investigação de perfis de uso tanto do AIMV como do ADS, no caso, o Eclipse.

A disponibilização do SourceMiner como um projeto de código aberto já está em fase de planejamento com o objetivo de se identificar a melhor estratégia de sua disponibilização e interação com a comunidade. Já foram reservados um domínio para o projeto (SourceMiner, 2010) e também uma área específica no *SourceForge* para tal finalidade. A sua disponibilização como projeto de código aberto possibilitará a inclusão de novas visões pela comunidade de visualização de software de acordo com as demandas tanto acadêmicas como da indústria.

Conforme visto no Capítulo 5, o SourceMiner tem dois pontos importantes de extensão na sua arquitetura. Suas estruturas visuais podem ser estendidas para suportar novas perspectivas. Seu modelo de eventos permite a inclusão de novas visões para cada uma das perspectivas já implementadas no AIMV.

Além disto, conforme discutido na Seção 5.1.4, o ambiente possui um módulo de monitoramento de atividades justamente para coleta de dados em estudos experimentais. Os registros de uso do SourceMiner-Eclipse gerados por este módulo podem ser analisados para identificar configurações de uso típicas, adotadas por usuários durante o uso do ambiente em certas atividades de engenharia de software.

Ainda como parte da estratégia de gestão do projeto SourceMiner, foram desenvolvidas várias atividades em parceria com grupos de pesquisa conforme descrito a seguir:

- Grupo de Engenharia de Software e Aplicações (GESA) da Universidade Salvador, Prof. José Maria David (agora na Universidade Federal Juiz de Fora) – Foi desenvolvida uma parceria para a adaptação do SourceMiner para dar suporte a atividades colaborativas. Este trabalho está em andamento através de uma co-orientação de dissertação de mestrado;
- Grupo ASiDE da Universidade Federal da Bahia, Profs. Christina Chavez e Cláudio Sant'Anna: na análise de modularidade de interesses, e participação na concepção e execução de dois dos estudos descritos no Capítulo 5 desta tese;

- Universidade Federal de Minas Gerais, Prof. Eduardo Figueiredo: concepção das representações visuais de interesses no SourceMiner e na participação e concepção do estudo observacional para identificar anomalias de modularidade descrito no Capítulo 5;
- Grupo OPUS da Pontifícia Universidade Católica do Rio de Janeiro, Prof. Alessandro Garcia e equipe: no planejamento e análise dos dados dos estudos sobre a análise da efetividade da representação visual de interesses também descrito no Capítulo 5.

O SourceMiner foi ainda utilizado em atividades de compreensão de software em uma disciplina do curso de graduação em Ciência da Computação da Universidade Federal de Pernambuco (UFPE).

8.4 TRABALHOS EM ANDAMENTO E FUTUROS

Diversos trabalhos estão em andamento para ampliar o uso e as funcionalidades do SourceMiner. A Figura 60 ilustra estes trabalhos:

- O SourceMiner está sendo adaptado para atividades colaborativas de compreensão de software (a idéia é que duas ou mais estações remotas possam apresentar e interargir sobre o mesmo conjunto de visões);
- O SourceMiner está sendo adaptado para representação de informações de evolução de software (um abordagem diferencial já foi criada para comparar visualmente diferentes versões do software extraídas de sistemas de controle de versão);
- Uma metodologia para analisar o perfil de uso do ADS e o do SourceMiner em atividades de desenvolvimento e manutenção de software está sendo desenvolvida (abordagens baseadas em visualização e mineração de dados

estão sendo desenvolvidas para analisar os dados obtidos pelo monitor de atividades associado ao SourceMiner);;

- Reavaliação da arquitetura do SourceMiner usando Metodologias Ágeis (a arquitetura do ambiente está sendo revista para abertura do seu código à comunidade de engenharia de software);

No aspecto de inserção científica e serviços para o avanço da área, resultante da experiência ganha com este trabalho, pode-se mencionar ainda a coordenação do I Workshop Brasileiro de Visualização de Software (WBVS) cuja proposta foi aceita no escopo do Segundo Congresso Brasileiro de Software (CBSOFT 2011). Apesar de alguns grupos nacionais de pesquisa já estarem trabalhando com visualização de software, ainda não havia no Brasil um evento em que essa comunidade pudesse discutir os desafios da área, assim como compartilhar experiências da indústria e da academia sobre as iniciativas neste contexto.

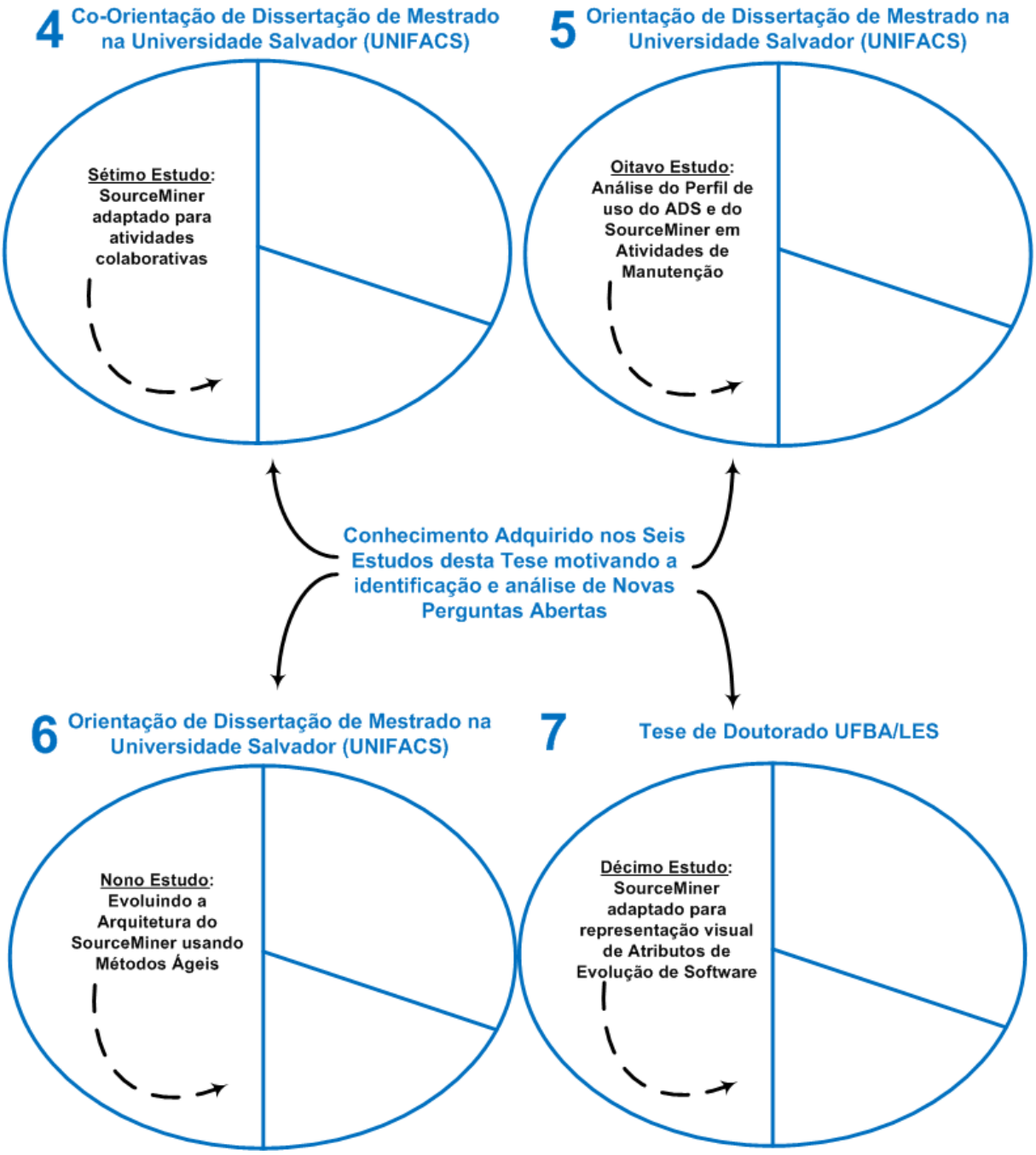

Figura 60 Trabalhos em Andamento

REFERÊNCIAS

AINSWORTH, S. The functions of multiple representations. **Comput. Educ.**, v. 33, p. 131-152, 1999.

ALAM, S.; BOCCUZZO, S.; WETTEL, R.; DUGERDIL, P.; GALL, H. LANZA, M.. EvoSpaces - Multi-dimensional Navigation Spaces for Software Evolution. In: **Human Machine Interaction**. Springer-Verlag Berlin, Heidelberg. 2009. p. 167-192.

ALWIS, B. D.; MURPHY, G. C. **Using Visual Momentum to Explain Disorientation in the Eclipse IDE**. Proceedings of the Visual Languages and Human-Centric Computing. IEEE Computer Society. 2006. p. 51-54.

ANDREWS, K.; HEIDEGGER, H. **Information Slices:** Visualizing and exploring large hierarchies using cascading, semi-circular discs. Proceedigns of the IEEE Symposium on Information Visualization. IEEE Computer Society Press. 1998. p. 9-12.

ANSLOW, C.; MARSHALL, S.; NOBLE, J.; BIDDLE, R. **Evaluating X3D for use in software visualization**. Proceedings of the ACM Symposium on Software Visualization. 2006. p. 161-162.

BAECKER, R. M.; MARCUS, A. **Human factors and typography for more readable programs**. ACM, 1989.

BAECKER, R.; MARCUS, A. Design principles for the enhanced presentation of computer program source text. **SIGCHI Bull.**, v. 17, p. 51-58, 1986.

BALL, T.; EICK, S. G. **Visualizing program slices**. Proceedings of the IEEE Symposium on Visual Languages. IEEE Computer Society Press. 1994. p. 288-295.

BALL, T.; EICK, S. G. Software visualization in the large. **Computer Journal.** 1996. v. 29, n. 4, p. 33-43.

BALZER, M. et al. **Software Landscapes:** Visualizing the Structure of Large Software Systems. Proceedings of the Joint Eurographics IEEE TCVG Symposium on Visualization (VisSym 2004), pages 261-266, Konstanz, Germany, May 19–21, 2004. p. 261-266.

BALZER, M.; DEUSSEN, O. **Hierarchy Based 3D Visualization of Large Software Structures**. Proceedings of the Conference on Visualization. 2004. p. 4-14.

BALZER, M.; DEUSSEN, O. **Level-of-detail visualization of clustered graph layouts**. Proceedings of the 6th International Asia-Pacific Symposium on Visualization. 2007. p. 133-140.

BALZER, M.; DEUSSEN, O.; LEWERENTZ, C. **Voronoi treemaps for the visualization of software metrics**. Proceedings of the 2005 ACM Symposium on Software Visualization. 2005. p. 165-172.

BASILI, V. R. **The role of experimentation in software engineering:** past, current, and future. Proceedings of the 18th International Conference on Software Engineering. IEEE Computer Society. 1996. p. 442-449.

BASILI, V. R.; ROMBACH, H. D. The TAME Project: Towards Improvement-Oriented Software Environments. **IEEE Trans. Software Eng.**, v. 14, n. 6, p. 758-773, 1988.

BATTISTA, G.; EADES, P.; TAMASSIA, R.; IOANNIS, G. Algorithms for Drawing Graphs: an Annotated Bibliography. **Comput. Geom.**, v. 4, p. 235-282, 1994.

BAUER, C.; KING, G. **Java Persistence with Hibernate**. Manning Publications Co., 2006.

BECKS, A.; SEELING, C. **SWAPit:** a multiple views paradigm for exploring associations of texts and structured data. Proceedings of the Working Conference on Advanced Visual Interfaces. ACM. 2004. p. 193-196.

BERARD, E. **Abstraction, Encapsulation, and Information Hiding**. Prentice Hall, v. I, 1993.

BOUKHELIFA, N.; RODGERS, P. J. A model and software system for coordinated and multiple views in exploratory visualization. **Information Visualization**, v. 2, p. 258-269, 2003.

BOUKHELIFA, N.; RODGERS, P. J. A model and software system for coordinated and multiple views in exploratory visualization. **Information Visualization Jounal**, v. 2, n. 4, p. 258-269, 2003.

BOULANGER, J.-S.; ROBILLARD, M. P. **Managing Concern Interfaces**. Proceedings of the 22nd IEEE International Conference on Software Maintenance. 2006. p. 14-23.

BRATTHALL, L.; WOHLIN, C. Is it Possible to Decorate Graphical Software Design and Architecture Models with Qualitative Information?-An Experiment. **IEEE Trans. Softw. Eng.**, v. 28, p. 1181-1193, 2002.

BRIAND, L. C. **The Experimental Paradigm in Reverse Engineering:** Role, Challenges, and Limitations. Proceedings of the 13th Working Conference on Reverse Engineering. IEEE Computer Society. 2006. p. 3-8.

BRIAND, L. C.; DALY, J. W.; WUST, J. K. A Unified Framework for Coupling Measurement in Object-Oriented Systems. **IEEE Trans. Softw. Eng.**, v. 25, p. 91-121, 1999.

BROOKS, R. Towards a theory of the comprehension of computer programs. **International Journal of Man-Machine Studies**, v. 18, n. 6, p. 543-554, 1983.

BROWN, M. H. Exploring Algorithms Using Balsa-II. **Computer**, v. 21, n. 5, p. 14-36, 1988.

BROWN, M. H.; SEDGEWICK, R. A system for algorithm animation. **SIGGRAPH Comput. Graph.**, v. 18, n. 3, p. 177-186, 1984.

BROWNING, T. R. Applying the design structure matrix to system decomposition and integration problems: a review and new directions. **IEEE Transactions on Engineering Management**, v. 48, n. 3, p. 292-306, 2001.

BUSCHMANN, F. et al. **Pattern-oriented software architecture:** a system of patterns. John Wiley & Sons, Inc., 1996.

BYELAS, H.; TELEA, A. **Visualization of areas of interest in software architecture diagrams**. Proceeding of the ACM Symposium on Software Visualization. 2006. p. 105-114.

BYELAS, H.; TELEA, A. **Visualizing metrics on areas of interest in software architecture diagrams**. Proceeding Proceedings of the IEEE Pacific Visualization Symposium. 2009. p. 33-40.

CACHO, N.; SANT´ANNA, C.; FIGUEIREDO, E.; GARCIA, A.; BATISTA, T.; LUCENA, C. **Composing design patterns: a scalability study of aspect-oriented programming**. Proceedings of the 5th International Conference on Aspect-oriented Software Development. ACM. 2006. p. 109-121.

CARD, S. K.; MACKINLAY, J. D.; SHNEIDERMAN, B. (Eds.). **Readings in information visualization:** using vision to think.Morgan Kaufmann Publishers Inc., 1999.

CARNEIRO, G. **Using Visual Metaphors to Enhance Software Comprehension Activities**. Talk at the International Summer School on Software Engineering at University of Salerno, Italy. 26th September, 2007.

CARNEIRO, G; MAGNAVITA, R.; SPINOLA, E.; SPINOLA, F.; MENDONÇA, M. **Evaluating the usefulness of software visualization in supporting software comprehension activities**. Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement ACM. 2008. p. 276-278.

CARNEIRO, G.; SANT´ANNA, C.; GARCIA, A.; CHAVEZ, C.; MENDONÇA, M. **On the Use of Software Visualization to Support Concern Modularization Analysis**. Proceedings of the 3rd Workshop on Assessment of Contemporary Modularization Techniques (ACoM 09), Co-located with OOPSLA. 2009. p. 2-10.

CARNEIRO, G.; SILVA, M.; MARA, L.; FIGUEIREDO, E.; SANT´ANNA, C.; GARCIA, A.; MENDONÇA, M. **Identifying Code Smells with Multiple Concern Views**. Proceedings of the CBSOFT-SBES. 2010. p. 128-137.

CARNEIRO, G.; ORRICO, A.; MENDONÇA, M. **Empirically Evaluating the Usefulness of Software Visualization Techniques in Program Comprehension Activities**. In: the Proceedings of VI Ibero-American Symposium on Software Engineering and Knowledge Engineering. 2007. p. 341-348.

CARNEIRO, G.; MAGNAVITA, R.; MENDONÇA, M. **Combining Software Visualization Paradigms to Support Software Comprehension Activities**. In Proceedings of the ACM Symposium on Software Visualization, ACM. 2008. p. 201-202.

CARNEIRO, G.; MAGNAVITA, R.; MENDONÇA, M. **An Experimental Platform to Characterize Software Comprehension Supported by Visualization**. International Conference on Software Engineering (ICSE) Research Demo Poster Session, 2009, Vancouver, BC - Canada. Proceeedings of the 31st International Conference on Software Engineering, 2009. p. 1-3.

CARNEIRO, G.; MAGNAVITA, R.; MENDONÇA, M. **Proposing a Visual Approach to Support the Characterization of Software Comprehension Activities**. In: IEEE International Conference on Program Comprehension Tool Demonstration Session, 2009, Vancouver, BC, Canada. Proceedings of the 17th IEEE International Conference on Program Comprehension, 2009. v. 1. p. 291- 292.

CARNEIRO, G.; MAGNAVITA, R.; MENDONÇA, M. G. **An Eclipse Based Visualization Tool for Software Comprehension**. In the XVII Brazilian Symposium on Software Engineering, Unicamp, São Paulo - Brazil. 2008.

CARNEIRO, G.; MENDONÇA, M. **The Importance of Cognitive and Usability Elements in Designing Software Visualization Tools**. Proceedings of the 20th Annual Psychology of Programming Interest Group Conference. Lancaster University, UK. 2008. p. 1-12.

CARNEIRO, G.; SANT´ANNA, C.; MENDONÇA, M. **On the Design of a Multi-Perspective Visualization Environment to Enhance Software Comprehension Activities**. Proceedings of the VII Workshop on Modern Software Maintenance (WMSWM 2010), Co-located with SBQS, 2010. Belém, Pará - Brazil. 2010. p. 1-8.

CASERTA, P.; ZENDRA, O. Visualization of the Static Aspects of Software: A Survey. **IEEE Transactions on Visualization and Computer Graphics**, v. 99, n. RapidPosts, 2010.

CHEN, Y.-F.; NISHIMOTO, M. Y.; RAMAMOORTHY, C. V. The C Information Abstraction System. **IEEE Trans. Softw. Eng.**, v. 16, p. 325-334, 1990.

CHI, E. H. **A Taxonomy of Visualization Techniques Using the Data State Reference Model**. Proceedings of the IEEE Symposium on Information Vizualization. IEEE Computer Society. 2000. p. 69-78.

CHUAH, M. C. **Dynamic Aggregation with Circular Visual Designs**. Proceedings of the IEEE Symposium on Information Visualization. IEEE Computer Society. 1998. p. 35-43.

CLAYBERG, E.; RUBEL, D. **Eclipse Plug-ins**. 3rd. ed. Addison Wesley Professional, 2009.

CLEMENTS, P. et al. **Documenting Software Architectures:** Views and Beyond. Second. ed.Pearson Education, 2010.

CONATI, C.; MACLAREN, H. **Exploring the role of individual differences in information visualization**. AVI '08: Proceedings of the working conference on Advanced visual interfaces. ACM. 2008. p. 199-206.

CONEJERO, J. M. et al. **Early Crosscutting Metrics as Predictors of Software Instability**. Objects, Components, Models and Patterns. Springer Berlin Heidelberg. 2009. p. 136-156.

CONROW, K.; SMITH, R. G. NEATER2: a PL/I source statement reformatter. **Commun. ACM**, v. 13, p. 669-675, 1970.

CONSENS, M. P. et al. Architecture and applications of the Hy+ visualization system. **IBM Syst. J.**, v. 33, p. 458-476, 1994.

CORBI, T. A. Program Understanding: Challenge for the 1990s. **IBM Systems Journal**, v. 28, n. 2, p. 294-306, 1989.

D'AMBROS, M.; LANZA, M.; LUNGU, M. Visualizing Co-Change Information with the Evolution Radar. **IEEE Trans. Softw. Eng.**, v. 35, p. 720-735, 2009.

DEMEYER, S.; DUCASSE, S.; LANZA, M. **A Hybrid Reverse Engineering Approach Combining Metrics and Program Visualization**. Proceedings of the Sixth Working Conference on Reverse Engineering. 1999. p. 175-186.

VAN DEURSEN, A.; HOFMEISTER, C.; HOSCHKE, R.; MOONEN, L.; RIVA, C.. **Viewpoints in Software Architecture Reconstruction**. Proceedings 6th Workshop on Software Reengineering (WSR). Bad Honnef. 2004.

DHAMBRI, K.; SAHRAOUI, H. A.; POULIN, P. **Visual Detection of Design Anomalies**. Proceedings of the 12th European Conference on Software Maintenance and Reengineering. 2008. p. 279-283.

DIEBERGER, A. **Browsing the WWW by interacting with a textual virtual environment— A Framework for Experimenting with Navigational Metaphors**. Proceedings of the the Seventh ACM Conference on Hypertext. ACM. 1996. p. 170-179.

DIEBERGER, A.; FRANK, A. U. A City Metaphor to Support Navigation in Complex Information Spaces. **J. Vis. Lang. Comput.**, v. 9, n. 6, p. 597-622, 1998.

DIEHL, S. (Ed.). **Software Visualization, International Seminar Dagstuhl Castle, Germany, May 20-25, 2001, Revised Lectures**.Springer, v. 2269, 2002.

DIEHL, S. **Software Visualization:** Visualizing the Structure, Behaviour, and Evolution of Software.Springer-Verlag New York, Inc., 2007.

DIJKSTRA, E. W. **A Discipline of Programming**. 1st. ed.Prentice Hall PTR, 1997.

DUCASSE, S.; GIRBA, T.; KUHN, A. **Distribution Map**. Proceedings of the 22nd IEEE International Conference on Software Maintenance. IEEE Computer Society. 2006. p. 203-212.

DUCASSE, S.; LANZA, M. The Class Blueprint: Visually Supporting the Understanding of Classes. **IEEE Trans. Software Eng.**, v. 31, n. 1, p. 75-90, 2005.

EADDY, M.; ZIMMERMANN, T.; SHERWOOD, K.; GARG, V.; MURPHY, G.; NAGAPPAN, N.; AHO, A. Do Crosscutting Concerns Cause Defects? **IEEE Trans. Softw. Eng.**, v. 34, p. 497-515, 2008.

EADDY, M.; AHO, A.; MURPHY, G. C. **Identifying, Assigning, and Quantifying Crosscutting Concerns**. Proceedings of the First International Workshop on Assessment of Contemporary Modularization Techniques. IEEE Computer Society. 2007. p. 2-12.

EICK, S. G. **Data visualization sliders**.ACM. 1994. p. 119-120.

EICK, S. G.; STEFFEN, J. L.; SUMNER, E. E. Seesoft-A Tool for Visualizing Line Oriented Software Statistics. **IEEE Trans. Softw. Eng.**, v. 18, p. 957-968, 1992.

EICK, S. G.; WILLS, G. J. **Navigating large networks with hierarchies**. Proceedings of the 4th Conference on Visualization. IEEE Computer Society. 1993. p. 204-209.

FAVRE, J. **GSEE:** A Generic Software Exploration Environment. Proceedings of the 9th International Workshop on Program Comprehension. IEEE Computer Society. 2001.

FAYAD, M. E.; SCHMIDT, D. C.; JOHNSON, R. E. **Building application frameworks:** object-oriented foundations of framework design.John Wiley & Sons, Inc., 1999.

FAYAD, M.; SCHMIDT, D. C. Object-oriented application frameworks. **Commun. ACM**, v. 40, p. 32-38, 1997.

FERREIRA, M. C.; LEVKOWITZ, H. From Visual Data Exploration to Visual Data Mining: A Survey. **IEEE Transactions on Visualization and Computer Graphics**, v. 9, n. 3, p. 378-394, 2003.

FIGUEIREDO, E.; CACHO, N.; SANT´ANNA, C.; MONTEIRO, M.; KULESZA, U.; GARCIA, A.; SOARES, S. **Evolving software product lines with aspects:** an empirical study on design stability. ICSE '08 Proceedings of the 30th International Conference on Software Engineering. 2008. p. 261-270.

FIGUEIREDO, E.; SANT´ANNA, C.; GARCIA, A.; LUCENA, C. **Applying and Evaluating Concern-Sensitive Design Heuristics**. Proceedings of the 2009 XXIII Brazilian Symposium on Software Engineering. IEEE Computer Society. 2009. p. 83-93.

FIGUEIREDO, E.; CARREIRO, B.; SANT´ANNA, C.; GARCIA, A. **Crosscutting patterns and design stability:** An exploratory analysis. In 17th International Conference on Program Comprehension. 2009. p. 138-147.

ECLIPSE FOUNDATION. **Eclipse User Interface Guideline Number 13.3 Integration with Other Views and Editors.** Disponível em http://www.eclipse.org/articles/Article-UI-Guidelines. 2011.

FOWLER, M. **Refactoring:** Improving the Design of Existing Code. 1. ed. Addison-Wesley Professional, 1999.

GALL, H.; JAZAYERI, M.; RIVA, C. **Visualizing Software Release Histories: The Use of Color and Third Dimension**. Proceedings of the IEEE International Conference on Software Maintenance. IEEE Computer Society. 1999. p. 99-108.

GAMMA, E. **Design patterns:** elements of reusable object-oriented software.Addison-Wesley, 1995.

GARCIA, A.; SANT´ANNA, C.; FIGUEIREDO, E.; KULESZA, U.; LUCENA, C.; VON STAA, A. **Modularizing Design Patterns with Aspects:** a Quantitative Study. Proceedings of the 4th International Conference on Aspect-oriented Software Development. ACM. 2005. p. 3-14.

GEARY, D.; HORSTMANN, C. S. **Core JavaServer Faces**. 3rd. ed.Prentice Hall Press, 2010.

GHONIEM, M.; FEKETE, J.-D.; CASTAGLIOLA, P. **A Comparison of the Readability of Graphs Using Node-Link and Matrix-Based Representations**. Proceedings of the IEEE Symposium on Information Visualization 2004. p. 17-24.

GIL, J.; KENT, S. **Three Dimensional Software Modelling**. Proceedings of the 20th International Conference on Software Engineering. Proceedings of the IEEE Symposium on Information Visualization (INFOVIS '04). IEEE Computer Society. 1998. p. 105-114.

GLASER, R.; RESNICK, L. B. **Knowing, learning, and instruction:** Essays in Honor of Robert Glaser.L. Erlbaum Associates. 1989.

GOGOLLA, M.; RADFELDER, O.; RICHTERS, M. **Towards three-dimensional Representation and Animation of UML Diagrams**. Proceedings of the 2nd International Conference on The Unified Modeling Language. Springer-Verlag. 1999. p. 489-502.

GOLDSTEIN, H.; NEUMANN, J. **Planning and Coding Problems of an Electronic Computing Instrument.** Macmillan, 1947. 80-151 p.

GORSCHEK, T., WOHLIN, C., GARRE, P., LARSSON, S. A Model for Technology Transfer in Practice. **IEEE Software**, v. 23, n. 6, p. 88-95, 2006.

GRACANIN, D.; MATKOVIC, K.; ELTOWEISSY, M. Software visualization. **Innovations in Systems and Software Engineering: A NASA Journal**, v. 1, n. 2, p. 221

GRAHAM, H.; YANG, H. Y.; BERRIGAN, R. **A solar system metaphor for 3D visualisation of object oriented software metrics**. Proceedings of the 2004 Australasian symposium on Information Visualisation - Volume 35. Australian Computer Society, Inc. 2004. p. 53-59.

GRAHAM, M.; KENNEDY, J. **Multiform Views of Multiple Trees**. Proceedings of the 12th International Conference Information Visualisation. IEEE Computer Society. 2008. p. 252-257.

GREENWOOD, P.; BARTOLOMEI, T.; FIGUEIREDO, E.; DOSEA, M.; GARCIA, A.; CACHO, N.; SANT´ANNA, C.; SOARES, S.; BORBA, P.; KULESZA, U.; AWAIS, R. **On the Impact of Aspectual Decompositions on Design Stability: An Empirical Study**. ECOOP 2007 – Object-Oriented Programming. 2007. p. 176-200.

GRINSTEIN, G.; TRUTSCHL, M.; CVEK, U. **High-Dimensional Visualizations**. Proceedings of the Visual Data Mining workshop (KDD'2001), 2001

GRISWOLD, W. G.; YUAN, J. J.; KATO, Y. **Exploiting the map metaphor in a tool for software evolution**. Proceedings of the 23rd International Conference on Software Engineering. IEEE Computer Society. 2001. p. 265-274.

HAARSLEV, V.; MÖLLER, R. Visualization and graphical layout in object-oriented systems. **Journal of Visual Languages & Computing**, v. 3, n. 1, p. 1-23, 1992.

HAIBT, L. M. **A program to draw multilevel flow charts**. Western Joint Computer Conference. 1959. p. 131-137.

HOLT, R.; PAK, J. Y. **GASE:** visualizing software evolution-in-the-large. Proceedings if the Working Conference on Reverse Engineering. 1996. p. 163-167.

HOLTEN, D. Hierarchical Edge Bundles: Visualization of Adjacency Relations in Hierarchical Data. **IEEE Trans. Vis. Comput. Graph.**, v. 12, n. 5, p. 741-748, 2006.

HOLTEN, D.; VLIEGEN, R.; VAN, J. J. **Visual Realism for the Visualization of Software Metrics**. VISOFT 2005. p. 27-32.

HUERAS, J.; LEDGARD, H. An automatic formatting program for PASCAL. **SIGPLAN Not.**, v. 12, p. 82-84, 1977.

HUNDHAUSEM, C.; DOUGLAS, S.; STASKO, J. A Meta-Study of Algorithm Visualization Effectiveness. **Journal of Visual Languages \& Computing**, v. 13, n. 3, p. 259-290, 2002.

HUNDHAUSEN, C. D. **Toward effective algorithm visualization artifacts:** designing for participation and negotiation in an undergraduate algorithms course. Conference on Human Factors in Computing Systems (CHI 98). ACM. 1998. p. 54-55.

HUNDHAUSEN, C. D. **Toward effective algorithm visualization artifacts: designing for participation and communication in an undergraduate algorithms course**. Doctoral Dissertation. University of Oregon Eugene, OR, USA. 1999.

HUNDHAUSEN, C. D. Using end-user visualization environments to mediate conversations: a 'Communicative Dimensions' framework. **J. Vis. Lang. Comput.**, v. 16, p. 153-185, 2005.

IN, H.; ROY, S. **Visualization Issues for Software Requirements Negotiation**. Proceedings of the 25th International Computer Software and Applications Conference on Invigorating Software Development. IEEE Computer Society. 2001. p. 10-15.

INSELBERG, A.; REIF, M.; CHOMUT, T. Convexity algorithms in parallel coordinates. **J. ACM**, v. 34, p. 765-801, 1987.
JACOBSON, I.; BOOCH, G.; RUMBAUGH, J. **The Unified Software Development Process**. Addison-Wesley Professional, 1999.

JENDROCK, E. et al. **The Java EE 6 Tutorial:** Basic Concepts. 4th. ed.Prentice Hall Press, 2010.

KAUSHIK, A. **Web Analytics 2.0:** The Art of Online Accountability and Science of Customer Centricity. SYBEX Inc., 2009.

KEIM, D. A. Pixel-oriented Visualization Techniques for Exploring Very Large Databases. **Jounal of Computational and Graphical Statistics**. v. 5, n. 1, p. 58–77, 1996.

KEIM, D. A. Designing pixel-oriented visualization techniques: Theory and applications.
IEEE Transactions on Visualization and Computer Graphics, v. 6, n. 5, p. 59–78, 2000.

KEIM, D. A. Information visualization and visual data mining. **IEEE Transactions on Visualization and Computer Graphics**, v. 8, n. 1, p. 1-8, 2002.

KEIM, D. A.; KRIEGEL, H.-P. Visualization Techniques for Mining Large Databases: A Comparison. **IEEE Trans. on Knowl. and Data Eng.**, v. 8, p. 923-938, 1996.

KELLER, R.; ECKERT, C. M. Matrices or node-link diagrams: which visual representation is better for visualising connectivity models? **Information Visualization Journal**, v. 5, p. 62-76, 2006.

KERSTEN, M.; MURPHY, G. C. **Using task context to improve programmer productivity**. SIGSOFT '06/FSE-14: Proceedings of the 14th ACM SIGSOFT international symposium on Foundations of software engineering. 2006. p. 1-11.

KICZALES, G.; LAMPING,J.; MENDHEKAR, A.; MAEDA, C.; VIDEIRA, C.; LOINGTIER, J.; IRWIN, J. **Aspect-Oriented Programming**. Proceedings of the Workshops on Object-Oriented Technology ECOOP 1997. p. 220-242.

KICZALES, G. et al. **An Overview of AspectJ**.Springer-Verlag. 2001. p. 327-353.

KITCHENHAM, B.; PFLEEGER, S.; PICKARD, L.; JONES, P.; HOAGLIN, D.; KHALED, E.; ROSENGERB, J. A. et al. Preliminary Guidelines for Empirical Research in Software Engineering. **IEEE Transactions on Software Engineering**, v. 28, n. 8, p. 721-734, 2002.

KLEYN, M. F.; GINGRICH, P. C. GraphTrace— Understanding Object-oriented Systems Using Concurrently Animated Views. **SIGPLAN Not.**, v. 23, p. 191-205, 1988.

KNIGHT, C.; MUNRO, M. **Virtual but Visible Software**. Proceedings of the International Conference on Information Visualisation. IEEE Computer Society. 2000. p. 198-206.

KNODEL, J.; MUTHIG, D.; NAAB, M. **Understanding Software Architectures by Visualization--An Experiment with Graphical Elements**.IEEE Computer Society. 2006. p. 39-50.

KNUTH, D. E. Computer-drawn flowcharts. **Commun. ACM**, v. 6, n. 9, p. 555-563, 1963.

KNUTH, D. E. Literate Programming. **The Computer Journal**, v. 27, n. 2, p. 97-111, 1984.

KO, A. J. et al. An Exploratory Study of How Developers Seek, Relate, and Collect Relevant Information during Software Maintenance Tasks. **IEEE Trans. Softw. Eng.**, v. 32, p. 971-987, 2006.

KO, A. J.; DELINE, R.; VENOLIA, G. **Information Needs in Collocated Software Development Teams**. ICSE '07 Proceedings of the 29th international conference on Software Engineering. IEEE Computer Society. 2007. p. 344-353.

KOSCHKE, R. Software visualization in software maintenance, reverse engineering, and re-engineering: a research survey. **Journal of Software Maintenance**, v. 15, p. 87-109, 2003.

KRASNER, G. E.; POPE, S. T. A cookbook for using the model-view controller user interface paradigm in Smalltalk-80. **J. Object Oriented Program.**, v. 1, p. 26-49, 1988.

LAKOFF, G.; JOHNSON, M. **Metaphors We Live By**. First Edition. ed.University Of Chicago Press, 1980.

LANGELIER, G.; SAHRAOUI, H.; POULIN, P. **Visualization-based analysis of quality for large-scale software systems**. Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering ACM. 2005. p. 214-223.

LANZA, M. **CodeCrawler - Lessons Learned in Building a Software Visualization Tool**. Proceedings of the Seventh European Conference on Software Maintenance and Reengineering. IEEE Computer Society. 2003. p. 409-418.

LANZA, M.; DUCASSE, S. **A Categorization of Classes based on the Visualization of their Internal Structure:** The Class Blueprint. OOPSLA '01 Proceedings of the 16th ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications 2001. p. 300-311.

LANZA, M.; DUCASSE, S. Polymetric Views-A Lightweight Visual Approach to Reverse Engineering. **IEEE Trans. Softw. Eng.**, v. 29, p. 782-795, 2003.

LANZA, M.; MARINESCU, R. **Object-Oriented Metrics in Practice**. Springer-Verlag New York, Inc., 2005.

LATOZA, T.; GARLAN, D.; HERBSLEB, J.; MYERS, B. **Program comprehension as fact finding**. Proceedings of the the 6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering. 2007. p. 361-370.

LEHMAN, M. M.; BELADY, L. A. (Eds.). **Program evolution:** processes of software change. Academic Press Professional, Inc., 1985.

LETHBRIDGE, T. C.; SIM, S. E.; SINGER, J. Studying Software Engineers: Data Collection Techniques for Software Field Studies. **Empirical Softw. Eng.**, v. 10, p. 311-341, 2005.

LEUNG, Y. K.; APPERLEY, M. D. A review and taxonomy of distortion-oriented presentation techniques. **ACM Trans. Comput.-Hum. Interact.**, v. 1, p. 126-160, 1994.

LINOS, P. K. et al. Visualizing program dependencies: an experimental study. **Softw. Pract. Exper.**, v. 24, p. 387-403, 1994.

LINTERN, R.; MICHAUD, J.; STOREY, M.; WU, X. **Plugging-in visualization: experiences integrating a visualization tool with Eclipse**. SoftVis '03: Proceedings of the 2003 ACM Symposium on Software visualization. 2003. p. 47--ff.

LIU, Z.; STASKO, J. Mental Models, Visual Reasoning and Interaction in Information Visualization: A Top-down Perspective. **IEEE Transactions on Visualization and Computer Graphics**, v. 16, p. 999-1008, 2010.

LUCCA, G.; PENTA, M. D. **Experimental Settings in Program Comprehension:** Challenges and Open Issues. 2006. p. 229-234.

LUNGU, M.; LANZA, M.; TUDOR, G.; REINOUT, H. **Reverse Engineering Super-Repositories**. Proceedings of the 14th Working Conference on Reverse Engineering. 2007. p. 120-129.

LUNGU, M. et al. The Small Project Observatory: Visualizing software ecosystems. **Science of Computer Programming**, v. 75, n. 4, p. 264-275, 2010. Experimental Software and Toolkits (EST 3): A special issue of the Workshop on Academic Software Development Tools and Techniques (WASDeTT 2008).

MACKINLAY, J. Automating the design of graphical presentations of relational information. **ACM Trans. Graph.**, v. 5, p. 110-141, 1986.

MAFRA, S. N.; BARCELOS, R. F.; TRAVASSOS, G. H. **Aplicando uma Metodologia Baseada em Evidência na Definição de Novas Tecnologias de Software**. XX Simpósio Brasileiro de Engenharia de Software. 2006. p. 239-254.

MALETIC, J. I.; MARCUS, A.; COLLARD, M. L. **A Task Oriented View of Software Visualization**. Proceedings of the 1st International Workshop on Visualizing Software for Understanding and Analysis (VISSOFT '02). 2002. p. 32-40.

MALETIC, J. I.; MARCUS, A.; FENG, L. **Source Viewer 3D (sv3D) - A Framework for Software Visualization**. Proceedings of the 25th International Conference on Software Engineering. 2003. p. 812-813.

MARCUS, A.; FENG, L.; MALETIC, J. I. **Comprehension of Software Analysis Data Using 3D Visualization**. Proceedings of the 11th IEEE International Workshop on Program Comprehension (IWPC '03). 2003. p. 105-114.

MARIN, M.; MOONEN, L.; DEURSEN, A. V. **SoQueT:** Query-Based Documentation of Crosscutting Concerns. Proceedings of the 29th International Conference on Software Engineering. IEEE Computer Society. 2007. p. 758-761.

MARINESCU, R. **Detection Strategies:** Metrics-Based Rules for Detecting Design Flaws. Proceedings of the 20th IEEE International Conference on Software Maintenance. IEEE Computer Society. 2004. p. 350-359.

MAYER, R. E.; ANDERSON, R. B. Animations Need Narrations: An Experimental Test of a Dual-Coding Hypothesis. **Journal of Educational Psychology**, v. 83, n. 4, p. 484-490, 1991.

MAYRHAUSER, A. V.; VANS, A. M. **From Code Understanding Needs to Reverse Engineering Tool Capabilities**. In Proceedings of the 6th International Workshop on Computer-Aided Software Engineering. 1993. p. 230-239.

MAYRHAUSER, A. V.; VANS, A. M. Program Comprehension During Software Maintenance and Evolution. **Computer**, v. 28, p. 44-55, 1995.

MAZZA, R. **Introduction to Information Visualization**. Springer Publishing Company. 2009.

MESNAGE, C.; LANZA, M. **White Coats:** Web-Visualization of Evolving Software in 3D. VISSOFT '05 Proceedings of the 3rd IEEE International Workshop on Visualizing Software for Understanding and Analysis. 2005. p. 1-6.

MOBILEMEDIA. **MobileMedia**. Disponível em http://mobilemedia.sourceforge.net. 2006.

MOHA, N. et al. DECOR: A Method for the Specification and Detection of Code and Design Smells. **IEEE Trans. Software Eng.**, v. 36, n. 1, p. 20-36, 2010.

MOHA, N.; GUEHENEUC, Y.-G.; LEDUC, P. **Automatic Generation of Detection Algorithms for Design Defects**. Proceedings of the 21st IEEE/ACM International Conference on Automated Software Engineering. 2006. p. 297-300.

MONTEIRO, M. P.; FERNANDES, J. M. **Towards a catalog of aspect-oriented refactorings**. AOSD '05 Proceedings of the 4th international conference on Aspect-oriented software development. 2005. p. 111-122.

MUKHERJEA, S.; FOLEY, J. D. **Requirements and Architecture of an Information Visualization Tool**. Proceedings of the IEEE Visualization '95 Workshop on Database Issues for Data Visualization. 1996. p. 57-75.

MULLER, H. A. et al. A Reverse-engineering Approach to Subsystem Structure Identification. **Journal of Software Maintenance: Research and Practice**, v. 5, n. 4, p. 181-204, 1993.

MULLER, H. A.; KLASHINSKY, K. **Rigi: a System for Programming-in-the-large**. Proceedings of the 10th International Conference on Software Engineering. 1988. p. 80-86.

MULLER, H. A.; TILLEY, S. R.; WONG, K. **Understanding software systems using reverse engineering technology perspectives from the Rigi project**. Proceedings of the Conference of the Centre for Advanced Studies on Collaborative Research: Software Engineering - Volume 1. IBM Press. 1993. p. 217-226.

MURPHY, G. C.; KERSTEN, M.; FINDLATER, L. How Are Java Software Developers Using the Eclipse IDE? **IEEE Softw.**, v. 23, p. 76-83, 2006.

MYERS, B. Taxonomies of visual programming and program visualization. **Journal of Visual Languages & Computing**, v. 1, n. 1, p. 97-123, 1990.

MYERS, B. A. **Visual programming, programming by example, and program visualization:** a taxonomy. Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. ACM. 1986. p. 59-66.

NASSI, I.; SHNEIDERMAN, B. Flowchart techniques for structured programming. **SIGPLAN Not.**, v. 8, n. 8, p. 12-26, 1973.

NORTH, C.; SHNEIDERMAN, B. **Snap-together visualization:** a user interface for coordinating visualizations via relational schemata. Proceedings of the Working Conference on Advanced Visual Interfaces ACM. 2000. p. 128-135.

OEZBEK, C.; PRECHELT, L. **JTourBus: Simplifying Program Understanding by Documentation that Provides Tours Through the Source Code**. In the Proceeding s of the International Conference on Software Maintenance. 2007. p. 64-73.

PAIVIO, A. **The Empirical Case for Dual Coding**. **Imagery, memory and cognition**, Hillsdale, NJ, Lawrence ErlbaumErlbaum, 1983. 307-332 p.

PANAS, T. et al. **Communicating Software Architecture using a Unified Single-View Visualization**. In the Proceedings of the 12th IEEE International Conference on Engineering Complex Computer Systems (ICECCS 2007). 2007. p. 217-228.

PANAS, T.; BERRIGAN, R.; GRUNDY, J. **A 3D Metaphor for Software Production Visualization**. Proceedings of the Seventh International Conference on Information Visualization. IEEE Computer Society. 2003. p. 314--.

PARNAS, D. L. On the criteria to be used in decomposing systems into modules. **Commun. ACM**, v. 15, p. 1053-1058, 1972.

PARNIN, C.; GORG, C.; NNADI, O. **A catalogue of lightweight visualizations to support code smell inspection**. Proceedings of the 4th ACM Symposium on Software Visualization. 2008. p. 77-86.

PAUW, W. D. et al. Visualizing the Behavior of Object-oriented Systems. **SIGPLAN Not.**, v. 28, n. 10, p. 326-337, 1993.

PENTA, M. D.; KURT, R. E.; KRAEMER, E. **Designing your Next Empirical Study on Program Comprehension**. Proceedings of the Internacional Conference on Program Comprehension. 2007. p. 281-285.

PERRY, D. E.; PORTER, A. A.; VOTTA, L. G. **Empirical studies of software engineering:** a roadmap. Proceedings of the Conference on The Future of Software Engineering. ACM. 2000. p. 345-355.

PETRE, M. Mental imagery and software visualization in high-performance software development teams. **J. Vis. Lang. Comput.**, v. 21, n. 3, p. 171-183, 2010.

PFLEEGER, S. L. Experimental Design and Analysis in Software Engineering. ACM SIGSOFT Software Engineering Notes **Ann. Software Eng.**, v. 1, p. 219-253, 1995.

PIGOSKI, T. M. **Practical Software Maintenance:** Best Practices for Managing Your Software Investment.John Wiley & Sons, Inc., 1996.

PRICE, B. A Principled Taxonomy of Software Visualization. **Journal of Visual Languages & Computing**, v. 4, n. 3, p. 211-266, 1993.

NOVAIS, R; CARNEIRO, G.; MENDONÇA, M. **On the Use of Software Visualization to Analyze Software Evolution:** An Interactive Differencial Approach. Proceedings of the 13th International Conference on Enterprise Information Systems (ICEIS 2011). 2011. p. 288-298.

RADFELDER, O.; GOGOLLA, M. **On better understanding UML diagrams through interactive three-dimensional visualization and animation**. Proceedings of the Working Conference on Advanced Visual Interfaces. 2000. p. 292-295.

RAJLICH, V.; DAMASKINOS, N.; LINOS, P.; KHORSHID, W. VIFOR: a tool for software maintenance. **Softw. Pract. Exper.**, v. 20, p. 67-77, 1990.

REISS, S. P. **Pecan:** Program development systems that support multiple views. Proceedings of the 7th International Conference on Software Engineering. IEEE Press. 1984. p. 324-333.

RICCA, F.; DI PENTA, M.; TORCHIANO, M.; TONELLA, P.; CECCATO, M. How Developers' Experience and Ability Influence Web Application Comprehension Tasks Supported by UML Stereotypes: A Series of Four Experiments. **IEEE Trans. Softw. Eng.**, v. 36, p. 96-118, 2010.

RIEL, A. J. **Object-Oriented Design Heuristics**. Addison-Wesley Professional, 1996.

RIJSBERGEN, C. J. **Information Retrieval**.Butterworth, 1979.

ROBERTS, J. C. **Multiple-View and Multiform Visualization**. IS&T and SPIE Visual data exploration and analysis. 2000. Vol. 3960 p. 176-185.

ROBERTS, J. C. **State of the Art:** Coordinated & Multiple Views in Exploratory Visualization. Proceedings of the Fifth International Conference on Coordinated and Multiple Views in Exploratory Visualization. IEEE Computer Society. 2007. p. 61-71.

ROBILLARD, M. P.; COELHO, W.; MURPHY, G. C. How Effective Developers Investigate Source Code: An Exploratory Study. **IEEE Trans. Softw. Eng.**, v. 30, p. 889-903, 2004.

ROBILLARD, M. P.; MURPHY, G. C. Representing concerns in source code. **ACM Trans. Softw. Eng. Methodol.**, v. 16, 2007.

ROBILLARD, M. P.; WEIGAND-WARR, F. **ConcernMapper:** simple view-based separation of scattered concerns. Proceedings of the 2005 OOPSLA Workshop on Eclipse Technology eXchange. ACM. 2005. p. 65-69.

ROMAN, G. C.; COX, K. C. A taxonomy of program visualization systems. **Computer**, v. 26, n. 12, p. 11-24, 1993.

ROMAN, G.-C.; COX, K. C. **Program visualization:** the art of mapping programs to pictures. Proceedings of the 14th International Conference on Software Engineering. ACM. 1992. p. 412-420.

SANGAL, N. et al. **Using Dependency Models to Manage Complex Software Architecture**. Proceedings of the 20th Annual ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications. 2005. p. 167-176.

SHNEIDERMAN, B. Tree visualization with tree-maps: 2-d space-filling approach. **ACM Trans. Graph.**, v. 11, p. 92-99, 1992.

SHNEIDERMAN, B. Dynamic Queries for Visual Information Seeking. **IEEE Softw.**, v. 11, p. 70-77, 1994.

SHNEIDERMAN, B.; MAYER, R. Syntactic/semantic interactions in programmer behavior: A model and experimental results. **International Journal of Parallel Programming**, v. 8, p. 219-238, 1979. 10.1007/BF00977789.

SHNEIDERMAN, B.; PLAISANT, C. **Designing the User Interface - Strategies for Effective Human-Computer Interaction (5. ed.)**.Addison-Wesley, 2010. I-XVIII, 1-606 p.

SHULL, F.; MENDONÇA, M.; BASILI, V.; CARVER, J.; MALDONADO, J.; FABBRI, S.; TRAVASSOS, G,; FERREIRA. M. Knowledge-Sharing Issues in Experimental Software Engineering. **Empirical Softw. Eng.**, v. 9, p. 111-137, 2004.

SHULL, F.; CARVER, J.; TRAVASSOS, G. H. **An Empirical Methodology for Introducing Software Processes**. Proceedings of the 8th European Software Engineering Conference held jointly with 9th ACM SIGSOFT International Symposium on Foundations of Software Engineering. 2001. p. 288-296.

SHULL, F.; SINGER, J.; SJOBERG, D. **Guide to Advanced Empirical Software Engineering**.Springer-Verlag New York, Inc., 2007.

SILLITO, J.; MURPHY, G; VOLDER, K. **Asking and answering questions during a programming change task**. IEEE Transactions on Software Engineering. Volume 34 Issue 4, July 2008. p. 434-451.

SILLITO, J.; VOLDER, K.; FISHER, B.; MURPHY, G. **Managing Software Change Tasks: An Exploratory Study**. Proceedings of the International Symposium on Empirical Software Engineering. IEEE Computer Society. 2005. p. 23-32.

SILLITO, J.; MURPHY, G. C.; VOLDER, K. D. **Questions programmers ask during software evolution tasks**. Proceedings of the 14th ACM SIGSOFT International Symposium on Foundations of Software Engineering. 2006. p. 23-34.

SJOBERG, D.; DYBA, T.; JORGENSEN, M. **The Future of Empirical Methods in Software Engineering Research**. FOSE '07 2007 Future of Software Engineering. IEEE Computer Society. 2007. p. 358-378.

SMARTMONEY. SmartMoney. Disponível em http://www.smartmoney.com/map-of-the-market. 2011.

SOLOWAY, E.; EHRLICH, K. Empirical studies of programming knowledge. Software reusability. ACM, 1989. p. 235-267.

SOUZA, C.; QUIRK, S.; TRAINER, E.; REDMILES, D. **Supporting collaborative software development through the visualization of socio-technical dependencies**. Proceedings of the 2007 international ACM conference on Supporting group work ACM. 2007. p. 147-156.

SOURCEMINER. **Sourceminer: Um Ambiente Integrado para Visualização Multi-Perspectiva de Software**. Disponível em www.sourceminer.org. 2010.

SPENCE, R. **Information Visualization:** Design for Interaction (2nd Edition). 2. ed.Prentice Hall, 2007.

STASKO, J. et al. (Eds.). **Software Visualization:** Programming as a Multimedia Experience.MIT Press, 1998.

STASKO, J.; ZHANG, E. **Focus+Context Display and Navigation Techniques for Enhancing Radial, Space-Filling Hierarchy Visualizations**. Proceedings of the IEEE Symposium on Information Vizualization. IEEE Computer Society. 2000. p. 57--.

STOREY, M.; MULLER, H. A. **Manipulating and Documenting Software Structures using SHriMP Views**. Proceedings of the International Conference on Software Maintenance. 1995. p. 275-284.

STOREY, M.-A. D.; FRACCHIA, F. D.; MULLER, H. A. Cognitive design elements to support the construction of a mental model during software exploration. **J. Syst. Softw.**, v. 44, p. 171-185, 1999.

STOREY, M.-A. D.; WONG, K.; MULLER, H. A. **How Do Program Understanding Tools Affect How Programmers Understand Programs**.IEEE Computer Society. 1997. p. 12-21.

SZYPERSKI, C. **Component Software:** Beyond Object-Oriented Programming. 2nd. ed.Addison-Wesley Longman Publishing Co., Inc., 2002.

CARPENDALE, M.; COWPERTHWAITE, D. J.; FRACCHIA, F. D. **Readings in information visualization**. Morgan Kaufmann Publishers Inc., 1999. Cap. Extending distortion viewing from 2D to 3D, p. 368-379.

TARR, P.; OSSHER, H.; HARRISON, W.; SUTTON, S. **N Degrees of Separation: Multi-Dimensional Separation of Concerns**. Proceedings of the 22th International Conference on Software Engineering. 1999. p. 107-119.

TEITELMAN, W. **A Tour through Cedar**. Proceedings of the 7th International Conference on Software Engineering. IEEE Press. 1984. p. 181-195.

TERMEER, M. et al. **Visual Exploration of Combined Architectural and Metric Information**. 2005. p. 21-26.

TIBCO. Spotfire. Disponível em http://spotfire.tibco.com. 2011.

THEUS, M. Interactive Data Visualization using Mondrian. **Journal of Statistical Software**, v. 7, n. 11, p. 1-9, 2002.

TURO, D.; JOHNSON, B. **Improving the Visualization of Hierarchies with Treemaps:** Design Issues and Experimentation. Proceedings of the 3rd Conference on Visualization. IEEE Computer Society Press. 1992. p. 124-131.

VISWANADHA, S.; SANKAR, S. **JavaCC**. Disponível em http://javacc.java.net. 2011.

WALLS, C. **Spring in Action**. 2. ed.Manning Publications, 2007.

WALRATH, K. et al. **The JFC Swing Tutorial:** A Guide to Constructing GUIs. 2. ed. Addison-Wesley, 2004.

WANG, M. Q.; WOODRUFF, A.; KUCHINSKY, A. **Guidelines for using multiple views in information visualization**. Proceedings of the working conference on Advanced visual interfaces. ACM. 2000. p. 110-119.

WANG, T. D.; PLAISANT, C.; SHNEIDERMAN, B.; SPRING, N.; ROSEMAN, D.; MARCH, G., MUKHERJEE, V.; SMITH, M. Temporal Summaries: Supporting Temporal Categorical Searching, Aggregation and Comparison. **IEEE Trans. Vis. Comput. Graph.**, v. 15, n. 6, p. 1049-1056, 2009.

WANG, W.; WANG, H.; DAI, G.; WANG. H. **Visualization of Large Hierarchical Data by Circle Packing**. Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. ACM. 2006. p. 517-520.

WARD, M. O. **XmdvTool: I**ntegrating Multiple Methods for Visualizing Multivariate Data. Proceedings of the Conference on Visualization. IEEE Computer Society Press. 1994. p. 326-333.

WARE, C. **Information Visualization, Second Edition:** Perception for Design (Interactive Technologies). 2. ed.Morgan Kaufmann, 2004.

WARE, C. Visual Queries: The Foundation of Visual Thinking. In: TERGAN, S.-O. ed Springer Berlin / Heidelberg. 2005.

KELLER, T.; SIGMAR-OLAF, T. **Visualizing Knowledge and Information: An Introduction**. Knowledge and Information Visualization. Springer Berlin / Heidelberg, v. 3426, 2005. p. 121-144.

WEGER, G. Semiologie graphique. In: **Cours de cartographie**. Cours de cartographie. Ecole Nationale des Sciences. 1997.

WENGER, E. **Artificial intelligence and tutoring systems: computational and cognitive approaches to the communication of knowledge**. Morgan Kaufmann Publishers Inc., 1987.

WETTEL, R.; LANZA, M. **Program Comprehension through Software Habitability**. Proceedings of the 15th IEEE International Conference on Program Comprehension. 2007. p. 231-240.

WETTEL, R.; LANZA, M. **Visually localizing design problems with disharmony maps**. Proceedings of the 4th ACM Symposium on Software Visualization. 2008. p. 155-164.

WOHLIN, C. et al. **Experimentation in software engineering:** an introduction.Kluwer Academic Publishers, 2000.

WU, J.; STOREY, M.-A. D. **A multi-perspective software visualization environment**. Proceedings of the Conference of the Centre for Advanced Studies on Collaborative Research. IBM Press. 2000. p. 41-50.

YANG, D. **Java Persistence with JPA**.Outskirts Press, 2010.

YIN, R. K. **Case Study Research:** Design and Methods, Third Edition, Applied Social Research Methods Series, Vol 5. 3rd. ed.Sage Publications, Inc, 2002.

ZHANG, K. (Ed.). **Software Visualization:** From Theory to Practice. Kluwer Academic Publishers, 2003.