Scientia
Interdisciplinary Studies in Computer Science
20(1): 12-24, January/June 2009
 – doi: 10.4013/sct.2009.20.1.02

RaisAware: uma ferramenta de auxílio à Engenharia de Software colaborativa baseada em análises de dependências

Jean M.R. Costa, Rafael M. Feitosa, Cleidson R.B. de Souza

Faculdade de Computação - Universidade Federal do Pará
Rua Auguusto Corrêa, 01, Guamá, Caixa Postal 479, 66075-110, Belém, PA, Brasil

{marcel, rafaelmf, cdesouza}@ufpa.br

Resumo
Este artigo apresenta a RaisAware, uma ferramenta de auxílio ao desenvolvimento colaborativo de software. RaisAware explora o relacionamento sociotécnico entre a arquitetura do software e a coordenação do trabalho de desenvolvimento de software através da análise de dependências entre os artefatos produzidos na etapa de codificação e entre as atividades dos desenvolvedores. As motivações teóricas para a ferramenta são apresentadas, assim como detalhes do projeto e implementação da RaisAware. Uma avaliação da ferramenta também é apresentada, utilizando-se dados reais de projetos de software livre. O artigo conclui com sugestões de trabalhos futuros.

PALAVRAS-CHAVE: desenvolvimento colaborativo de software, *call-graph*, *co-changes*, análise de dependências, arquitetura de software, coordenação.

Abstract
RaisAware: Tool support for collaborative Software Engineering based on dependence analysis. This paper presents RaisAware, a collaborative software development tool aimed at supporting the relationship between software architecture and coordination of software development activities. RaisAware's design is based on dependency analysis between software development artifacts created during the implementation phase and software developers' activities. We describe the theoretical motivations for our tool, detail its design and implementation, and present an evaluation of the tool using open-source project data. Finally, we provide recommendations for future work in this direction.

KEY WORDS: collaborative software development, call-graph, co-changes, dependency analysis, software architecture, coordination.

1 Introdução

Desenvolvimento de software é um típico exemplo de trabalho colaborativo em que diversos engenheiros de software precisam trabalhar juntos para entregar o produto final no prazo, de acordo com as especificações e com a qualidade definida. De fato, a atividade de desenvolvimento de software tem sido parte da pesquisa na área de Sistemas Colaborativos desde os seus primeiros anos (Grudin, 1994). Estudos da atividade de desenvolvimento de software como uma atividade colaborativa têm resultado tanto em contribuições teóricas para a área de Sistemas Colaborativos como em ferramentas para suportar as atividades de desenvolvimento de sistemas. No primeiro caso, Grinter (1995), por exemplo, estudou como os desenvolvedores usam ferramentas de gerência de configuração para coordenar as suas atividades. Um exemplo do segundo caso é o estudo de Halverson *et al*. (2006) sobre visualizações para apoiar a gestão de solicitações de mudança em grandes projetos de desenvolvimento distribuído de software.

Este trabalho mantém a tradição dos estudos anteriores de pesquisar o desenvolvimento de software como atividade colaborativa; assim, descreve a ferramenta

RaisAware de auxílio ao desenvolvimento colaborativo de software. Em particular, a ferramenta visa auxiliar a atividade de implementação de sistemas. A RaisAware foi motivada pelos resultados anteriores de estudos de campo realizados por um dos autores com equipes de desenvolvimento de software (de Souza, 2005). Ela explora o relacionamento entre a arquitetura do software e a coordenação das atividades de desenvolvimento de software, uma relação há muito conhecida (Conway, 1968; Parnas, 1972), mas pouco explorada (Morelli *et al*., 1995; de Souza *et al*., 2004; MacCormack *et al*., 2004; Cataldo *et al*., 2006; de Souza e Redmiles, 2008). Mais especificamente, a ferramenta suporta esse relacionamento a partir da análise (i) dos artefatos produzidos no desenvolvimento de software e (ii) das atividades dos desenvolvedores. A análise dos artefatos é realizada a partir da utilização de técnicas que identificam dependências entre os componentes de software, ao passo que a análise das atividades dos desenvolvedores é baseada na análise dos padrões de atividades dos desenvolvedores relacionados às mudanças feitas no código do projeto.

Em resumo, a contribuição deste artigo é a descrição de uma ferramenta inovadora de apoio às atividades de implementação em contextos colaborativos de desenvolvimento de software. A ferramenta é motivada por resultados de estudos etnográficos com engenheiros de software, que sugerem diversos cenários de uso da ferramenta. Um aspecto importante deste trabalho é a avaliação da ferramenta a partir de dados reais de projetos de desenvolvimento de software.

O restante do artigo é organizado como segue. A próxima seção apresenta uma breve revisão de ferramentas de apoio ao desenvolvimento colaborativo de software. Na seção 3, são descritos alguns dos estudos teóricos e empíricos que motivaram o projeto da RaisAware. Em particular, o *framework* de gerência de impacto proposto por de Souza e Redmiles (2008) é descrito. Após isso, na seção 4, é discutida a relação sociotécnica entre arquitetura de software e coordenação das atividades que norteia o desenvolvimento da RaisAware, além das abordagens utilizadas pela ferramenta para calcular as dependências entre artefatos e atividades. A seção 5 apresenta a ferramenta; é descrita sua arquitetura e são explicitados alguns detalhes de sua implementação, seu funcionamento e, principalmente, as abordagens utilizadas pela ferramenta para calcular as dependências entre artefatos e atividades. A avaliação da RaisAware é realizada na seção 6. Finalmente, a seção 7 apresenta as considerações finais e as propostas para trabalhos futuros.

2 Uma breve revisão de ferramentas de apoio à Engenharia de Software colaborativa

Conforme mencionado, a ferramenta RaisAware não é a primeira a auxiliar o desenvolvimento colaborativo de software: existem diversas ferramentas que visam auxiliar esta atividade. Um das abordagens mais tradicionais é baseada em sistemas de gerência de configuração como o CVS (2009) e Subversion (2009). Estes sistemas gerenciam as diversas versões dos artefatos criados durante o processo de desenvolvimento de software e também permitem que diversos desenvolvedores trabalhem em paralelo. O desenvolvimento paralelo significa que dois ou mais desenvolvedores estão modificando o mesmo arquivo ao mesmo tempo (Perry *et al*., 2001). Essa situação é difícil de se lidar porque ela requer uma coordenação extra entre os desenvolvedores. De fato, modificações em paralelo se correlacionam com código defeituoso (Perry *et al*., 2001). Alguns sistemas de gerência de configuração até mesmo limitam que diversas pessoas editem um mesmo arquivo, para evitar os problemas do desenvolvimento paralelo (Sarma *et al*., 2003).

Citar todas as ferramentas de apoio à engenharia de software colaborativa é uma tarefa complexa e inviável no espaço permitido para este artigo. Dessa forma, o foco desta seção é em ferramentas que oferecem funcionalidades similares à RaisAware. Por exemplo, as ferramentas Palantír (Sarma *et al*., 2003) e Ariadne (Trainer *et al*., 2005) são ferramentas que visam facilitar a coordenação de atividades mediante a percepção[1] (Dourish e Bellotti, 1992) das atividades que ocorrem no projeto no qual um desenvolvedor está inserido. A percepção é obtida pela visualização de informações apresentadas integradas ao ambiente de desenvolvimento do desenvolvedor. Cada uma dessas ferramentas apresenta apenas um tipo de dependência, enquanto que a RaisAware extrai informações a partir de diferentes tipos de dependências (ver seção 4). Por exemplo, o Palantír é voltado para as atividades síncronas de desenvolvimento de software onde dois ou mais desenvolvedores estão modificando os arquivos ao mesmo tempo.

O ambiente Jazz (Cheng *et al*., 2003) é uma ferramenta colaborativa que também visa facilitar a percepção do status dos artefatos no ambiente de desenvolvimento dos outros desenvolvedores, como, por exemplo, se o artefato foi salvo localmente e não foi realizado commit, ou seja, os arquivos não foram enviados para o repositório de configuração. A ferramenta FASTDash (Biehl *et al*., 2007) visa facilitar a percepção das atividades da equipe de desenvolvimento usando uma representação espacial do

[1] Em inglês, *awareness*.

projeto que realça as atividades de outros membros. Esta representação indica, por exemplo, quais arquivos estão sendo visualizados ou que classes e métodos estão sendo modificadas naquele instante.

As ferramentas citadas acima visam auxiliar o desenvolvimento colaborativo de software, oferecendo aos desenvolvedores variados recursos para facilitar a coordenação das atividades de desenvolvimento de software. Uma outra categoria de estudos sugere que até mesmo ferramentas simples podem ser efetivas ao auxiliar a coordenação das atividades de desenvolvimento de software, desde que possuam uma infraestrutura adequada para a troca de informações. Por exemplo, o trabalho de Fitzpatrick *et al*. (2006) ilustra como a utilização de um sistema de notificações pode ser efetivo para facilitar a coordenação das atividades entre os desenvolvedores de um mesmo projeto.

3 Estudos anteriores de Engenharia de Software colaborativa

A ferramenta RaisAware é baseada nos resultados de estudos de campo realizados com equipes de desenvolvimento de software (de Souza, 2005; de Souza e Redmiles, 2008). Estes estudos resultaram em um *framework* analítico para entender o trabalho de desenvolvedores de software, ao lidar com os efeitos das dependências de software nas suas atividades. O *framework* é chamado de *gerência de impacto* e é definido como "o trabalho realizado por um desenvolvedor de software para minimizar o impacto de suas atividades em seus colegas e, ao mesmo tempo, o impacto das atividades de seus colegas em suas atividades" (de Souza e Redmiles, 2008, p. 242). A gerência de impacto possui três aspectos principais:

(i) Primeiro, encontrar o conjunto de pessoas que possivelmente afetam o trabalho de um desenvolvedor e que podem ser afetadas pelo trabalho deste. Este conceito é chamado de "rede de impacto". Identificar esta rede é o aspecto mais importante da gerência de impacto, e de fato esta é a principal funcionalidade da ferramenta RaisAware;

(ii) Segundo, a gerência de impacto "*forward*"[2] é o trabalho realizado por um desenvolvedor para minimizar o efeito do seu trabalho em sua respectiva rede de impacto; e

(iii) Finalmente, a gerência de impacto "*backward*"[3] consiste no trabalho que um desenvolvedor realiza para minimizar o efeito, no seu próprio trabalho, do trabalho dos desenvolvedores presentes em sua rede de impacto.

De Souza e Redmiles (2008) ilustram o *framework* de gerência de impacto com dados etnográficos; consideram não apenas o contexto local dos desenvolvedores estudados, mas o contexto mais amplo de desenvolvimento de software como sugerido por (Dourish, 2006). Alguns desses aspectos são detalhados a seguir, já que foram considerados no projeto da ferramenta.

Um problema comum identificado na literatura da engenharia de software colaborativa é o fato de que os desenvolvedores, muitas vezes, não têm consciência das suas redes de impacto, ou seja, de quais desenvolvedores podem afetar ou ser afetados pelas suas atividades. Desenvolvedores de software adotam diversas abordagens para lidar com essa situação. Algumas dessas abordagens são técnicas (por exemplo, usar um banco de dados (de Souza e Redmiles, 2008) ou consultar arquivos de *log* (McDonald e Ackerman, 1998)), enquanto outras são sociais (usar as redes sociais de seus gerentes, ou atribuir a desenvolvedores a responsabilidade primária de se comunicar com outros desenvolvedores que fornecem componentes de software para o time). A tarefa de identificar a rede de impacto é difícil porque essas redes encolhem e expandem durante o processo de desenvolvimento de software (de Souza e Redmiles, 2008). Estes últimos autores constatam que desenvolvedores se engajam na gerência de impacto porque eles conhecem detalhes da arquitetura do software que estão desenvolvendo. Desenvolvedores experientes, por exemplo, sabem que alguns componentes de software são fontes de dependências para outros componentes e, assim, eles usam esse conhecimento para garantir o adequado andamento das atividades e evitar impactar o trabalho de seus colegas (de Souza e Redmiles, 2008). Em outras palavras, o número de dependências de um componente de software influencia diretamente na forma como os desenvolvedores realizam suas atividades com este componente. Por exemplo, estes componentes são testados de maneira mais cuidadosa (de Souza *et al*., 2003).

Outro importante aspecto do trabalho envolvido no desenvolvimento de software é o ato de evitar o trabalho paralelo (Sarma *et al*., 2003; de Souza *et al*., 2003; Grinter, 2003), isto é, os desenvolvedores adotam estratégias (por exemplo, acelerar os seus trabalhos) para evitar a necessidade de desenvolver em paralelo com os seus colegas.

[2] Do inglês, *forward impact management.*
[3] Do inglês, *backward impact management.*

4 Dependências de software e coordenação das atividades

A relação entre a estrutura de uma organização de software e a própria estrutura do software tem sido debatida desde a proposta de Conway (1968). Este autor argumentou que a estrutura do software produzido é um reflexo da organização que o produziu. Em geral, os pesquisadores têm associado a estrutura do software com as dependências técnicas entre os componentes, que, como sugerido inicialmente por Conway (1968), desempenham um papel importante na coordenação dos trabalhos envolvidos. São essas dependências entre os componentes do software que definem a arquitetura deste.

De um modo geral, pode-se afirmar que todo software é organizado em um conjunto de arquivos de código fonte relacionados, ou seja, que dependem um do outro. Assim, ao modificar um arquivo que pertença a este conjunto, há uma grande possibilidade de vários outros arquivos serem afetados, e consequentemente, de outros desenvolvedores serem afetados. Dessa forma, uma abordagem adotada para fazer inferências sobre a coordenação do trabalho é utilizar técnicas de análise de dependências (de Souza *et al*., 2004; Cataldo *et al*., 2006). Esta seção descreve as duas abordagens mais utilizadas para identificar dependências entre componentes na literatura, ambas implementadas na RaisAware.

4.1 Dependências baseadas na arquitetura do software

Em engenharia de software, diferentes estruturas de dados podem ser utilizadas para permitir a representação e manipulação das dependências de um software. No caso da RaisAware, é utilizado o *call-graph*. O *call-graph* é um grafo direcionado que representa os relacionamentos entre os componentes de software, a partir das chamadas existentes entre os métodos destes. Os nós do *call-graph* são os procedimentos do programa, e as arestas representam as chamadas existentes entre os procedimentos. Por exemplo, uma aresta (p1, p2) existe, se o procedimento pl pode chamar o procedimento p2 de dentro de p1. O *call-graph* tem sido utilizado para revelar a potencial estrutura dinâmica de um sistema de software, ou seja, desvenda dependências entre os desenvolvedores responsáveis pelos componentes de software (Bowman e Holt, 1999; de Souza *et al*., 2004). Por exemplo, assumindo que um componente de software *a*, que está sendo desenvolvido por um programador *A*, depende de um outro componente b, sendo implementado por um desenvolvedor *B*, pode-se concluir que o trabalho do desenvolvedor *A* depende do resultado do trabalho do desenvolvedor *B*. Isto é, estes desenvolvedores, potencialmente, precisam se comunicar e coordenar seus trabalhos a fim de garantir um bom fluxo de trabalho e o adequado resultado final.

4.2 Dependências baseadas nas atividades dos desenvolvedores

O uso da abordagem baseada em *call-graphs* não permite identificar todas as dependências em um projeto de software. Por exemplo, conectores são utilizados por arquitetos de software para modelar as interações entre os componentes (Perry e Wolf, 1992), mas essas interações não são capturadas por meio da análise de dependências com o *call-graph*. O *call-graph* não permite identificar dependências existentes a partir do uso de reflexão ou do uso de padrões de projeto como o de injeção de dependências. Essas dependências podem ser obtidas a partir de uma técnica chamada de *co-changes*, que identifica as dependências entre os artefatos de software com base na identificação de quais artefatos são modificados frequentemente em conjunto (Zimmermann *et al*., 2003). Essa técnica é baseada nos padrões históricos de atividades de modificação do software feitas por desenvolvedores, em vez da análise do software propriamente dito. A ideia básica é a seguinte: se um arquivo foi modificado muitas vezes juntamente com outro arquivo, é porque existe uma dependência lógica, também chamada de dependência evolutiva, entre eles. Diferentemente do *call-graph*, que revela dependências estáticas para os arquivos de código fonte do software, a técnica de *co-changes* revela dependências lógicas para qualquer tipo de artefato. Estas dependências são chamadas, neste artigo, de dependências baseadas nas atividades dos desenvolvedores, pois se baseiam na análise dos arquivos modificados por vários desenvolvedores do projeto, em vez da análise do código fonte.

Em geral, as dependências com base nas atividades dos desenvolvedores são extraídas a partir da mineração de repositórios de código, como os controlados pelo sistema de controle de versões CVS. A identificação de *co-changes* entre os componentes é feita mediante a criação de regras de associação entre os arquivos alterados em conjunto, ou seja, arquivos que são frequentemente modificados juntos são susceptíveis de estarem acoplados.

Zimmermann *et al*. (2003) apresenta duas medidas para identificar o grau de dependência entre os artefatos:

- *Support* (Suporte): calcula quantas vezes um artefato foi modificado juntamente com outro; e
- *Confidence* (Confiança): calcula a proporção das modificações, ou seja, esta porcentagem indica quantas vezes dois arquivos foram modificados juntos.

No caso da RaisAware, um artefato é dito dependente de outro se o valor de suporte for de pelo menos 7, e o valor de confiança for de, pelo menos, 35%. Estes valores estão de acordo com o proposto por Zimmermann *et al*. (2003).

4.3 Dependências na ferramenta RaisAware

A RaisAware utiliza tanto a arquitetura do software como as atividades dos desenvolvedores para identificar as dependências existentes entre os componentes; desta forma, ela é mais completa que outras ferramentas de auxílio ao desenvolvimento colaborativo de software, como a Palantír (Sarma *et al*., 2003) e Ariadne (Trainer *et al*., 2005). A Palantir é uma ferramenta com uma proposta similar à RaisAware: ela é um *plugin* para a IDE Eclipse, que captura informações sobre os artefatos que estão sendo modificados, identifica as dependências existentes entre eles, e envia as informações obtidas para os outros desenvolvedores afetados. Ou seja, ela identifica dependências apenas da arquitetura do software. A Ariadne também é um *plugin* para a IDE Eclipse, cujo objetivo é apresentar visualizações sociais (desenvolvedores) e técnicas (artefatos) das dependências existentes em projetos de desenvolvimento de software. Assim como a Palantir, a Ariadne identifica as dependências entre os artefatos a partir das chamadas existentes entre as classes, mas não obtém as dependências baseadas nas atividades dos desenvolvedores.

A RaisAware, ao correlacionar os arquivos modificados com potenciais conflitos, como os indiretos e *co-changes*, permite ao desenvolvedor conhecer a rede de impacto, uma funcionalidade que não existe nas ferramentas Palantir, Ariadne, Jazz e FASTDash, anteriormente descritas. A ferramenta RaisAware é descrita em mais detalhes na próxima seção.

5 A ferramenta RaisAware

Esta seção descreve a ferramenta RaisAware, que é o objetivo deste trabalho. Inicialmente, são apresentadas as suas principais funcionalidades de acordo com o *framework* teórico de gerência de impacto anteriormente descrito. Posteriormente, é apresentada a arquitetura da ferramenta e a descrição do seu funcionamento.

5.1 Identificação dos artefatos mais dependentes

Conforme já discutido, desenvolvedores experientes possuem conhecimento sobre a arquitetura do software e utilizam este conhecimento para ajustar suas atividades. Desenvolvedores inexperientes, por outro lado, não têm esse conhecimento sobre a arquitetura, sobre os arquivos que apresentam um alto grau de dependências, o que torna suas atividades mais propensas a erros (de Souza *et al*., 2003). Visando facilitar o trabalho destes desenvolvedores, a RaisAware oferece um mecanismo de identificação dos arquivos mais dependentes. Para identificar estes arquivos, é feita uma análise sobre os dados contidos na matrizes que contém informações sobre as dependências, de modo a encontrar os 15% de arquivos com mais dependências.

A visualização dos arquivos mais dependentes é feita mediante o uso de *Decorators*, ou seja, enfeites aplicados sobre os ícones correspondentes aos arquivos mais dependentes, conforme pode ser visto na Figura 1. Nessa figura, os ícones com *Decorators* estão contornados por círculos. A ideia é que, dessa forma, quando um desenvolvedor nota um *Decorator* sobre o ícone do arquivo que ele está pensando em modificar, passa a ter ciência de que aquele arquivo precisa ser manipulado com mais cuidado (de Souza *et al*., 2003).

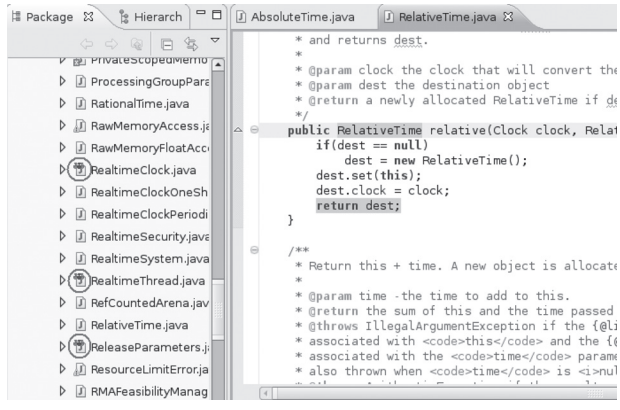

Figura 1. *Decorators* usados para identificar os arquivos mais dependentes.
Figure 1. Decorators used to identify the most dependent files.

5.2 Identificação dos componentes de software impactados

Esta funcionalidade permite que sejam identificados os arquivos que serão impactados, a partir de um conjunto de arquivos a serem modificados. A visualização desses arquivos pode ser feita pelo usuário da RaisAware de duas formas: (i) clicando com o botão direito do mouse sobre o ícone de um arquivo (visualizando os arquivos impactados ou que impactam este arquivo) ou (ii) através do ícone do projeto (visualizando todos os arquivos e seus correspondentes impactos), ao escolher a opção "View Technical Network".

Para gerar essa visualização, são identificados, inicialmente, os arquivos que estão sendo alterados por um desenvolvedor de software. Estes arquivos são comparados com os arquivos alterados por outros desenvolvedores, uma vez que as informações sobre as atividades dos outros desenvolvedores também são armazenadas localmente, quando enviadas pelos outros *Workspaces*. Sempre que a RaisAware identifica que dois desenvolvedores estão mudando o mesmo arquivo, ou seja, envolvidos em desenvolvimento paralelo, ela cria dois nós representando este mesmo arquivo e uma aresta ligando estes dois arquivos, para indicar esse potencial conflito entre os desenvolvedores. Depois disso, a RaisAware pesquisa, nas duas matrizes de informações sobre dependências, quais são os arquivos impactados e, uma vez identificados estes arquivos, verifica-se se estão sendo remotamente alterados. Em caso afirmativo, assim como na representação de desenvolvimento paralelo, são criados nós e uma aresta para representar que um arquivo modificado por um programador pode afetar ou ser afetado pelo outro. O RaisAware usa, então, um *framework* de visualização, chamado Prefuse (2009), para exibir essas informações.

A Figura 2, apresenta um exemplo de visualização dessas informações. O desenvolvimento paralelo (conflito direto) é representado pelas arestas mais espessas; as dependências entre artefatos (conflito indireto) são indicadas pelas arestas direcionadas e, finalmente, as dependências de atividades dos desenvolvedores (*co-changes*) são representadas como arestas não direcionadas na rede. Na Figura 2, o arquivo Clock está envolvido em conflito direto, ou seja, enquanto um desenvolvedor está editando o arquivo Clock, outro desenvolvedor também está paralelamente editando este mesmo arquivo. Devido ao desenvolvimento paralelo deste arquivo, dois arquivos Clock são apresentados na Figura. AbsoluteTime e Clock são exemplos de conflitos indiretos, e, finalmente, RealTimeClock e Clock indicam um exemplo de *co-changes*.

As Figuras 3 e 4 apresentam outras telas da ferramenta. Na Figura 3, há um *menu* no canto direito com um *checklist*, a partir do qual o usuário pode filtrar a visualização do Raisaware, apresentando apenas os nós e as arestas correspondentes aos tipos de dependências selecionados. Além disso, há um *tooltip* com informações mais detalhadas sobre o conflito direto envolvido no arquivo Clock: é possível saber, por exemplo, que outro usuário está editando paralelamente este arquivo. Na Figura 4, é mostrado um *tooltip* que indica os motivos da existência de um conflito indireto entre o arquivo HighResolutionTime e Clock.

5.3 Identificação da rede de impacto

Com base nas informações dos arquivos que afetam e são afetados por outros arquivos, a RaisAware gera uma rede de impacto (de Souza, 2005; de Souza e Redmiles, 2008). Essa rede indica as pessoas que precisam potencialmente coordenar seus trabalhos para evitar conflitos.

A Figura 5 apresenta um exemplo de rede de impacto. Assim como com o conjunto de arquivos modificados, o desenvolvimento paralelo é representado pelas arestas mais espessas, as dependências baseadas no *call-graph* são indicadas pelas arestas direcionadas, e as dependên-

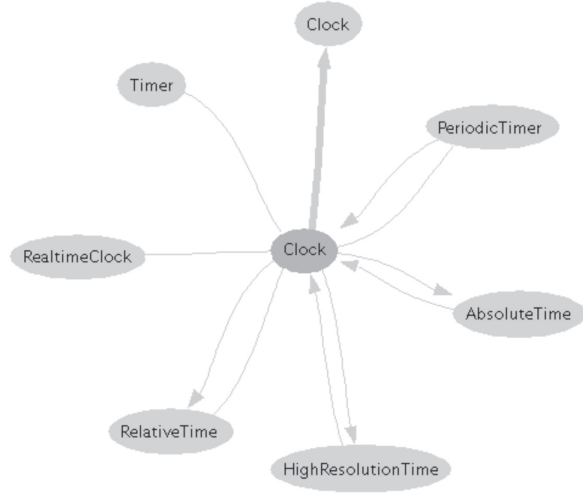

Figura 2. Conjunto de Arquivos Impactados.
Figure 2. Set of impacted files.

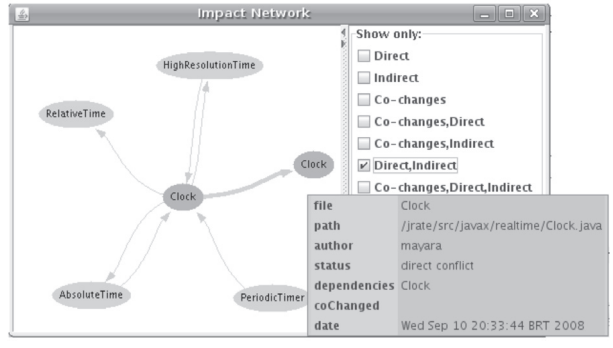

Figura 3. Filtro "Direct, Indirect" selecionado e detalhes do arquivo Clock sendo mostrados.
Figure 3. Direct and Indirect filters selected. Details of file Clock are showed.

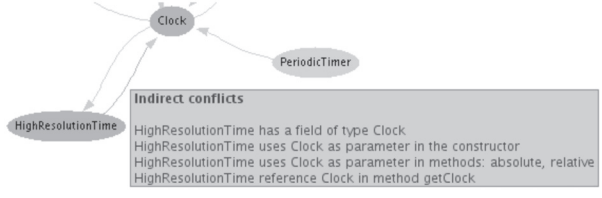

Figura 4. Explicações sobre a existência do conflito indireto.
Figure 4. Explanations about the existence of indirect conflict.

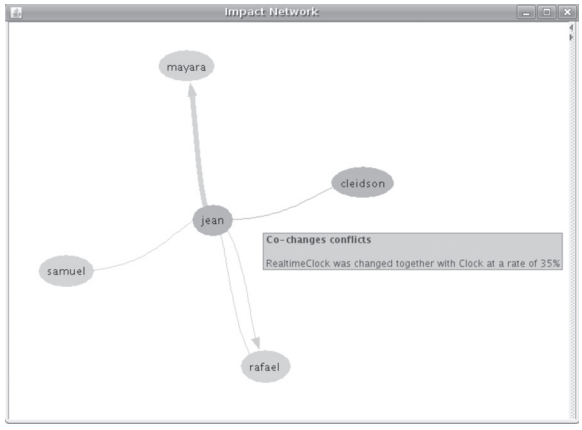

Figura 5. Exemplo da rede de impacto social
Figure 5. Impact network example.

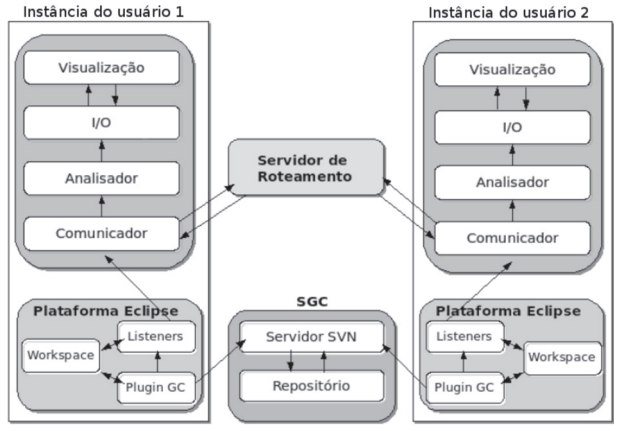

Figura 6. Arquitetura da RaisAware.
Figure 6. RaisAware's Architecture.

cias baseadas nas atividades dos desenvolvedores estão representadas pelas arestas não direcionadas.

A visualização da rede de impacto social é dinâmica, assim como a rede de impacto técnica (entre artefatos). Quando o usuário mantém o mouse em cima de um dos nós por alguns segundos, informações como: nome do desenvolvedor, data da modificação, arquivos impactados e razões dos impactos são mostradas em um *tooltip*. Da mesma forma, quando o usuário mantém o mouse em cima de uma aresta, é mostrado o tipo de dependência existente entre os artefatos e a razão pelo qual a aresta existe. Na Figura 5, por exemplo, a aresta que liga o desenvolvedor Jean com o desenvolvedor Cleidson representa uma dependência baseada nas atividades (*co-changes*), uma vez que os arquivos RealTimeClock e Clock (que estão sendo modificados por estes desenvolvedores) foram modificados juntos a uma taxa superior a medida de confiança.

5.4 Arquitetura

A ferramenta RaisAware foi desenvolvida com a utilização da linguagem de programação Java e implementada como um *plugin* para a popular IDE (*Integrated Development Environment* – Ambiente de Desenvolvimento Integrado) *Eclipse*. A ideia de integrar a ferramenta ao Eclipse visa permitir a utilização de inúmeros recursos disponibilizados pela IDE. Por exemplo, é possível monitorar eventos em cada instância do Eclipse sendo utilizada. Dessa forma, sempre que um desenvolvedor começar uma ação considerada relevante pela ferramenta, como, por exemplo, iniciar a modificação de arquivos de código fonte, é disparado um evento que, uma vez processado pela RaisAware, transmite as informações obtidas para todos os desenvolvedores.

A arquitetura do sistema é mostrada na Figura 6. As setas representam o fluxo de informações, ao passo que os retângulos indicam os componentes da RaisAware. Os blocos superiores de cada instância, contendo Visualização, I/O (entrada e saída de dados), Analisador e Comunicador são os componentes do *plugin* RaisAware implementados, e os demais identificam os componentes externos. A descrição dos componentes e de como eles interagem é feita na seção seguinte.

5.5 O funcionamento da RaisAware

O primeiro passo para a utilização da RaisAware é registrar um projeto na ferramenta, ou seja, selecionar o projeto para o qual a RaisAware irá coletar e transmitir dados a partir dos eventos ocorridos. Assim que o projeto é registrado, a ferramenta se encarrega de coletar as informações de dependências entre os seus artefatos. Basicamente, duas *threads* são iniciadas, uma que recupera dependências, baseando-se na arquitetura do software – ou seja, localmente – e outra que as obtém a partir das atividades dos desenvolvedores – ou seja, remotamente. A seguir, é apresentada uma descrição de como esses dados são obtidos.

5.5.1 Extração de dependências baseadas na arquitetura do software

Para coletar as dependências baseadas na arquitetura do software, é utilizado um recurso disponibilizado pelo Eclipse, chamado de *ASTParser*. Este recurso permite, dentre outras ações, identificar as referências existentes entre os arquivos de código fonte de um projeto Java. Assim que o usuário registra um determinado projeto na RaisAware, são calculadas as dependências para todas as classes Java do projeto, e, a partir dessas informações, é gerado um *call-graph*, representando essas dependências.

Este *call-graph* é armazenado como uma matriz {classe X classe}, chamada de *call-graph matrix*, que é armazenada no diretório raiz do projeto na forma de um arquivo CSV (*Comma Separated Values*), um formato que apresenta dados tabelados.

5.5.2 Extração de dependências baseadas nas atividades dos desenvolvedores

Para coletar as dependências baseadas nas atividades dos desenvolvedores, é utilizado o método de *co-changes* (Zimmermann *et al*., 2003). Este método permite que sejam descobertas dependências evolutivas entre artefatos a partir da frequência com que eles foram modificados juntos. Para identificar se um artefato foi modificado junto com outro, são obtidos dados do repositório de gerência de configuração do projeto, ao qual os artefatos pertencem. Todos os arquivos que foram submetidos juntos em cada operação de *commit* são analisados, a partir do componente Plugin GC (Plugin de Gerência de Configuração), que permite acessar e recuperar dados de sistemas de gerência de configuração (SGC). Na implementação atual, como Plugin GC, é possível utilizar o *plugin* Subclipse para acessar repositórios gerenciados pelo sistema Subversion, e um *plugin* integrado no Eclipse para acesso a repositórios CVS. Assim que os dados são obtidos, é criada uma matriz {arquivo X arquivo}, chamada de *co-changes matrix*, que indica quantas vezes cada artefato foi modificado juntamente com outro. Essa matriz é armazenada no diretório raiz do projeto na forma de um arquivo CSV, e é a partir da leitura e processamento dos dados contidos nela que as medidas de suporte e confiança são calculadas e utilizadas.

Uma vez que os dois arquivos {*call-graph matrix e co-changes matrix*} contendo as informações de dependências são gerados, a ferramenta está pronta para coletar e transmitir informações para os outros desenvolvedores. O processo todo começa no *Workspace*[4] do desenvolvedor: enquanto o desenvolvedor realiza ações no *Workspace*, são disparados eventos que são capturados pelos *Listeners* do *Workspace*. Estes *Listeners* são interfaces integradas ao Eclipse que monitoram o *Workspace* para identificar a ocorrência de eventos. Quando são disparados eventos indicando que o usuário realizou uma ação de salvar modificações em um arquivo, fechar um arquivo no editor da IDE ou uma operação de *commit*, o componente Comunicador da RaisAware, que implementa os *Listeners*, captura esses eventos e os encaminha para o servidor de notificação. Este componente se encarrega de enviar os dados obtidos (nome do arquivo, desenvolvedor que o modificou, data de modificação etc.) para todos os desenvolvedores que estiverem usando a RaisAware, para o projeto no qual o evento foi capturado.

O servidor de notificação foi implementado utilizando o serviço de roteamento de eventos *Avis* (2008). Na instância do RaisAware de outro desenvolvedor, o próprio Comunicador é responsável pela obtenção dos dados transmitidos pelo servidor de notificação. Assim que os dados são obtidos, são repassados para o componente Analisador, que busca o nome do arquivo obtido nas matrizes de dependências e, ao encontrá-lo, processa os dados identificando as dependências existentes e repassa o resultado para o componente I/O. Este componente gera um arquivo XML contendo as informações de todos os arquivos que estão sendo modificados. Por fim, a camada de Visualização se encarrega de apresentar ao usuário as interfaces gráficas referentes à rede de impacto e a outros aspectos. Os dados necessários para a criação das interfaces são repassados pela camada I/O, que efetua um parsing sobre os dados contidos no arquivo XML gerado.

6 Avaliação da ferramenta

A avaliação da ferramenta RaisAware foi feita de maneira indireta. Em outras palavras, não foi possível avaliar a efetividade da ferramenta em um ambiente real e nem em um experimento. Dessa forma, optou-se por simular situações de uso da ferramenta a partir de dados reais, a fim de ser possível avaliar o projeto da ferramenta, especialmente suas visualizações. Todos estes aspectos são discutidos nesta seção.

6.1 Metodologia

Para avaliar a RaisAware, foi utilizada a abordagem de avaliação de uma ferramenta de apoio à análise de impacto de modificações em programas Java, descrita em Ren *et al.* (2004). A abordagem consiste em (i) obter dados sobre os *commits* efetuados em um projeto de desenvolvimento de software durante o período de um ano, (ii) particionar estes dados em semanas, e (iii) analisar os *commits* realizados durante os intervalos de cada semana, de modo que esses *commits* sejam considerados

[4] O Workspace corresponde ao componente da IDE Eclipse que possui tudo o que o desenvolvedor precisa para escrever, compilar, rodar e testar os seus projetos.

como atividades paralelas. Ou seja, esta abordagem assume que, se foram realizados *commits* por mais de um desenvolvedor dentro de um intervalo de uma semana, possivelmente ocorreram atividades paralelas durante esse período, e, consequentemente, houve conflitos diretos durante o desenvolvimento. A análise do código-fonte dos arquivos modificados durante os *commits* permite a identificação de conflitos indiretos, e a análise histórica dos *commits* permite identificar conflitos *co-changes*. Em resumo, utilizando-se esta abordagem, a ferramenta RaisAware pôde ser avaliada a partir da identificação e observação dos possíveis conflitos (diretos, indiretos e *co-changes*) que potencialmente ocorreram em cada semana do projeto analisado.

Em vista da dificuldade em se obter dados de projetos corporativos de desenvolvimento de software para avaliar a ferramenta, optou-se por analisar um projeto de software livre, visto que todos os dados desses projetos estão amplamente acessíveis. O projeto escolhido foi o MegaMek, um jogo em rede desenvolvido em Java e que está registrado no site SourceForge.net desde 2002, contando com a participação de 33 desenvolvedores na sua implementação. Este projeto, por ser um dos mais ativos da comunidade, e conseqüentemente devido ao alto potencial de conflitos que poderiam ocorrer no mesmo, foi a escolha ideal para a avaliação da ferramenta. Entretanto, é necessário garantir que, no ano de desenvolvimento analisado, tenham sido efetuados *commits* por mais de um desenvolvedor em um curto intervalo de tempo. Para se identificar o intervalo de tempo a ser utilizado foi utilizada a ferramenta TransFlow (Sousa Júnior *et al*., 2009; Costa, 2009), que permite visualizar informações sobre os *commits* efetuados em um projeto de software.

A Figura 7 apresenta uma visualização gerada pela TransFlow para o primeiro ano de desenvolvimento do projeto MegaMek. Na figura, cada quadrado corresponde a um *commit* realizado por um desenvolvedor, e a cor do quadrado é utilizada para diferenciar os autores dos *commits*. Os quadrados são desenhados com uso de coordenadas cartesianas para plotar valores para duas variáveis. No caso da Figura 7, X é um eixo temporal que corresponde ao número de *commits* efetuado, enquanto o eixo Y indica o número de classes novas adicionadas ao projeto[5]. A partir deste gráfico, é possível identificar os períodos em que vários desenvolvedores trabalharam ativamente no projeto. Por exemplo, o retângulo representado pelo número 1 indica um período em que um único desenvolvedor efetuou *commits* no projeto, enquanto o retângulo representado pelo número 2 indica um período em que diversos outros desenvolvedores realizaram *commits*. Assim, a partir da visualização gerada pela TransFlow, foi constatado que o primeiro ano (períodos 1 e 2) de desenvolvimento do projeto MegaMek poderia ser utilizado para avaliar a ferramenta, pois diversos desenvolvedores atuaram nele.

Visando obter os dados do repositório de gerência de configuração do projeto Megamek e para calcular os conflitos que ocorreram durante o desenvolvimento, foi implementado um módulo na RaisAware que permite identificar a ocorrência de conflitos diretos, indiretos e *co-changes* para cada semana de desenvolvimento de um projeto de software. Uma vez que os conflitos são identificados, os dados referentes a esses conflitos (número de conflitos, semana, nome dos arquivos etc.) são exportados em formato CSV, para que possam ser visualizados na forma de gráficos em aplicativos de planinha eletrônica.

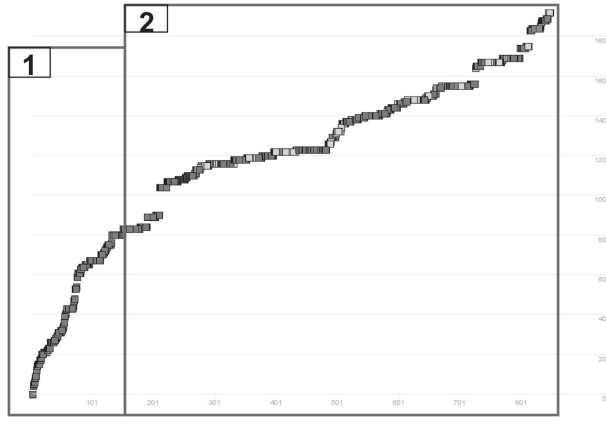

Figura 7. Visualização gerada pela TransFlow para o primeiro ano de desenvolvimento do projeto MegaMek.
Figure 7. TransFlow's visualization for the first year of the project development MegaMek.

6.2 Resultados

Os resultados obtidos por meio da simulação para avaliar a ferramenta podem ser observados nos gráficos a seguir. Os dados gerados podem ser visualizados para todo o projeto ou considerando apenas a rede de impacto de determinados arquivos, seja do ponto de vista técnico ou social.

Inicialmente, a Figura 8 apresenta a ocorrência de conflitos diretos, indiretos e *co-changes*, sob o ponto de vista técnico, para cada semana do ano considerado. Deve-se observar que, inicialmente, os valores para os conflitos são iguais a zero, o que corresponde ao período 1 da Figura 7, quando apenas um desenvolvedor atuava no projeto. Somente após 10 semanas, isto é, na semana 11, é

[5] Esta métrica é frequentemente utilizada para indicar a taxa de crescimento de um software (Ren *et al.*, 2004).

que os valores de conflitos começam a surgir, exatamente no período 2 da Figura 7, quando outros desenvolvedores começam a participar do projeto.

Além disso, é possível perceber que houve semanas em que a quantidade de arquivos envolvidos em conflitos aumentou consideravelmente. Por exemplo, na penúltima semana, foram identificados mais de 45 possíveis conflitos. Isso pode ser justificado pela quantidade de *commits* efetuada nesta semana e também pelo número de desenvolvedores que submeteram código durante o período. A Figura 9 apresenta uma visualização gerada pela TransFlow; mostram-se os *commits* que foram efetuados durante essa semana. Na figura, o eixo X corresponde ao número de *commits* efetuado, enquanto que o eixo Y indica o número de classes novas adicionadas ao projeto. É possível observar, na visualização, que foram efetuados 24 *commits*, um número bem maior do que a média de *commits* de cada semana do ano (13). Além disso, o número de desenvolvedores que atuaram durante a semana é outro fator a ser considerado, já que 4 desenvolvedores efetuaram *commits* ao longo da semana: o dobro da média de desenvolvedores que realizaram *commits* em cada semana do ano analisado.

O gráfico da Figura 8 apresenta o número de arquivos potencialmente envolvidos em conflito em cada semana do ano. Este é o número total de arquivos que podem estar em conflito em uma dada semana, não o número de arquivos modificados por um determinado desenvolvedor. Dessa forma, o gráfico da Figura 8 apresenta um super-conjunto dos possíveis arquivos que seriam afetados, ou afetariam, um único desenvolvedor. Para avaliar o número especifico de arquivos de interesse

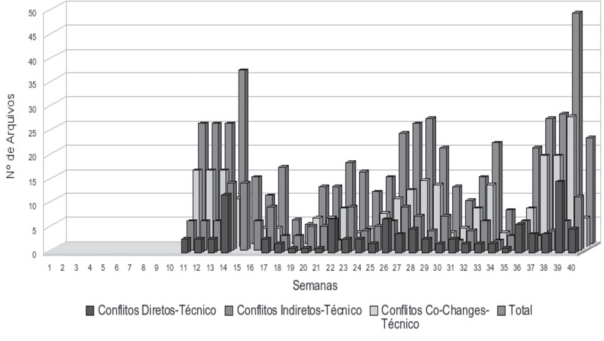

Figura 8. Total de conflitos do ponto de vista técnico.
Figure 8. Total of conflicts from a technical point of view.

Figura 9. Visualização dos commits efetuados durante a semana em que foram identificados mais potenciais conflitos diretos.
Figure 9. Visualization of commits made during the week that were identified most direct potential conflicts.

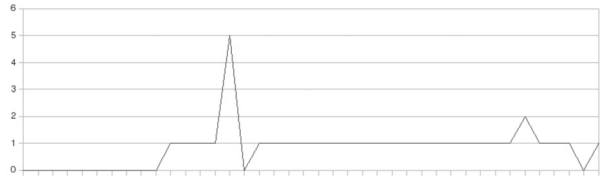

Figura 10. Média do número de arquivos impactados por conflitos indiretos para cada arquivo.
Figure 10. Average number of files impacted by indirect conflicts for each file.

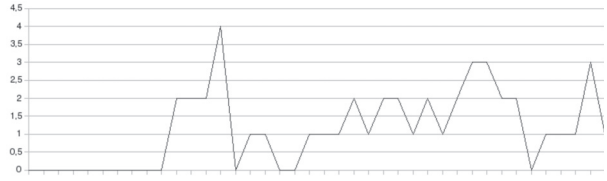

Figura 11. Média do número de arquivos impactados por conflitos do tipo *co-changes* para cada arquivo
Figure 11. Average number of files impacted by co-changes conflicts for each file.

de um desenvolvedor, a seguinte análise foi realizada: para cada arquivo identificado em uma semana da análise, foi identificado o número de arquivos ao qual este arquivo está conectado na rede de impacto, isto é, o grau do arquivo. A média do grau de cada arquivo envolvido em possíveis conflitos foi calculado para cada semana. Os resultados são mostrados na Figura 10 e na Figura 11, que apresentam a média resultante de cada semana do ano para os conflitos indiretos e para os *co-changes*, respectivamente.

A Figura 12, a seguir, apresenta os dados sobre os possíveis conflitos identificados no ano, só que agora considerando a rede de impacto social. Novamente, deve-se observar que até a semana 10, nenhum conflito foi identificado, visto que só havia um desenvolvedor ativo no projeto, conforme ilustrado na Figura 7, disponibilizada anteriormente.

Finalmente, a média e o desvio padrão do número de arquivos (ou pessoas) possivelmente envolvidos em um conflito foi calculada para cada tipo de conflito e os

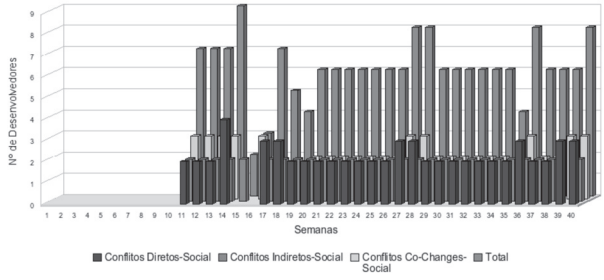

Figura 12. Total de conflitos do ponto de vista social.
Figure 12. Total of conflicts from a socialperspective.

resultados são apresentados na Tabela 1. É possível observar que a média de conflitos por *co-changes*, do ponto de vista técnico, é maior que os dois outros tipos de conflitos, ou seja, a quantidade de arquivos na rede de impacto baseada no método *co-changes* tende a ser maior que os valores observados nos conflitos diretos e indiretos. Além disso, pode-se também perceber que o número de arquivos afetados (rede técnica) é sempre maior que o número de desenvolvedores correspondente (rede social).

6.3 Discussão

Os gráficos apresentados na seção anterior demonstram que a ferramenta RaisAware foi capaz de identificar potenciais conflitos no projeto Megamek durante o período de análise. É importante ressaltar que nenhum gráfico indica qualquer tipo de conflito (valores iguais a zero) para as primeiras 10 semanas do projeto. Este resultado é coerente com a análise da Figura 7, indicativa de que, nessas semanas, havia apenas um único desenvolvedor efetuando *commits* no projeto: não existiam outros desenvolvedores no projeto; logo, potenciais conflitos não podem ser identificados.

Desconsiderando as 10 semanas iniciais, é possível observar que as semanas restantes apresentam valores bem diferentes, quando os gráficos são comparados entre si. Por exemplo, considerando os valores encontrados para a penúltima semana do gráfico de conflitos técnicos, o valor é igual a 27 para os conflitos do tipo *co-changes*; 15, para os conflitos diretos; e 6, para os conflitos indiretos. Este resultado é bastante significativo, visto que não existe um consenso geral da comunidade científica sobre qual método de identificação de conflitos é o melhor (Kagdi e Maletic, 2007). Entretanto, como a proposta da ferramenta é auxiliar o desenvolvedor de software na tarefa de identificação dos conflitos, é possível que o próprio desenvolvedor, com a sua experiência de uso da ferramenta, possa filtrar quais informações são úteis e quais não o são.

Um aspecto importante a ser avaliado a partir dos gráficos é se a visualização que a RaisAware oferece é de fato efetiva. Analisando os gráficos que apresentam os valores do ponto de vista técnico, considerando todos os arquivos modificados em cada semana, é possível perceber que uma grande quantidade de potenciais conflitos é identificada. Assim, a apresentação das informações na forma de grafo não seria a mais adequada, já que grafos com muitos elementos comprometem a visualização, devido à sobreposição de nós e arestas (Herman *et al.*, 2000). Isso dificulta a análise do usuário para identificar cada conflito, levando, até mesmo, a requerer-se a intervenção do usuário (movendo nós e arestas na visualização) para observar mais atentamente um conflito em particular.

Por outro lado, ao se observarem os gráficos da Figura 10 e 11, pode-se perceber que a média do número de arquivos impactados por cada arquivo não chega a ultrapassar o valor de 5, na Figura 10; e 4, na Figura 11. Dessa forma, a rede de impacto gerada para cada um dos arquivos que um usuário está modificando não contém tantos elementos, o que sugere que a visualização por meio de grafos da rede de arquivos impactados pode até mesmo ser adequada. Para se obter uma resposta definitiva para esta pergunta, pretende-se realizar um experimento controlado (Wohlin *et al.*, 2000) com estudantes de graduação.

De maneira similar, pode-se afirmar que a visualização é adequada para apresentar as redes de impacto sociais, já que o gráfico da Figura 12 não apresenta valores altos: o valor máximo total da rede é 9, que só ocorre uma vez. Isso implica em uma pequena quantidade de elementos no grafo de visualização da rede de impacto social e, consequentemente, pode-se concluir que a visualização utilizada na RaisAware é efetiva.

Visto que as visualizações da RaisAware são efetivas, pode-se argumentar que os desenvolvedores usuários da mesma serão capazes de identificar as suas respectivas redes de impacto, seja ela a rede técnica ou a social. Do ponto de vista técnico, o desenvolvedor, ao visualizar as dependências entre os arquivos que estão sendo modificados, tem percepção de que precisa coordenar suas atividades com outros desenvolvedores, para evitar que as modificações no código do projeto levem a problemas no software. A rede social de impacto permite a identificação das pessoas que podem ser impactadas

Tabela 1. Média e desvio padrão dos conflitos analisados.*
Table 1. Average and standard deviation of analyzed conflicts.

	Técnico		Social	
	Média	**Desvio Padrão**	**Média**	**Desvio Padrão**
Conflitos Diretos	3,63	3,24	2,17	0,79
Conflitos Indiretos	5,9	3,13	1,87	0,51
Conflitos por *Co-Changes*	8,63	6,7	2,07	0,94

* Calculado a partir do momento em que houve desenvolvimento paralelo.

pelo código do desenvolvedor ou possíveis de impactar o código do mesmo. Dessa forma, ela indica com quem um dado desenvolvedor precisa coordenar suas atividades, algo considerado problemático em estudos anteriores sobre o trabalho de engenheiros de software (Sarma *et al*., 2003; de Souza *et al*., 2003). A RaisAware, portanto, facilita o trabalho dos engenheiros de software envolvidos em atividades de implementação, substituindo várias abordagens que são necessárias para a gerência de impacto (de Souza e Redmiles, 2008).

7 Considerações finais

Neste artigo, foi apresentada a RaisAware, uma ferramenta de auxílio ao desenvolvimento colaborativo de software integrada no popular IDE Eclipse, especialmente na fase de codificação de software. Os aspectos teóricos encontrados na literatura que motivaram a implementação da ferramenta foram apresentados, além da descrição da arquitetura e das técnicas de identificação de dependências utilizadas pela RaisAware.

O objetivo da ferramenta é suportar a gerência de impacto conforme o framework proposto por (de Souza, 2005; de Souza e Redmiles, 2008), ao invés de focar no suporte de estratégias particulares que são específicas para certos contextos. A ferramenta fornece dados em tempo real sobre os impactos das atividades realizadas pelos desenvolvedores e, desta forma, acredita-se que ela permita aos seus usuários terem ciência (Dourish e Bellotti, 1992) de suas redes de impacto, para que possam adotar estratégias que não prejudiquem o trabalho de seus colegas, bem como para evitar que outras pessoas afetem os seus próprios trabalhos. A ferramenta RaisAware foi avaliada por meio de uma abordagem baseada em uma simulação de uso da ferramenta a partir de dados reais de um projeto de software livre. O resultado dessa avaliação sugere que as visualizações geradas pela RaisAware podem ser efetivas em um contexto similar ao do projeto analisado.

Um aspecto que precisa ser investigado em trabalhos futuros é a influência de diferentes modelos de desenvolvimento na ocorrência de conflitos entre engenheiros de software. Modelos tradicionais de desenvolvimento de software baseados em processos disciplinados, com pouca rotatividade de desenvolvedores, clara definição e separação de papéis podem estar menos propensos ao risco de conflitos do que, por exemplo, o modelo de desenvolvimento adotado em projetos de software livre. Outro possível fator a ser investigado é a distribuição geográfica dos desenvolvedores. Neste caso, o resultado desta análise pode levar à utilização de outras formas de visualização da rede de impacto. Por exemplo, diferentes formas geométricas podem ser usadas para indicar a localização dos desenvolvedores presentes na rede de impacto. Ainda como propostas para trabalhos futuros, planeja-se utilizar a ferramenta entre estudantes de graduação, para que eles possam avaliar a sua usabilidade e efetividade. Neste caso, pretende-se realizar experimentos controlados (Wohlin *et al*., 2000) com esses estudantes.

Agradecimentos

Esta pesquisa tem financiamento do CNPq por intermédio do Edital Universal 2006, processo 479206/2006-6, do Edital Universal 2008, processo 473220/2008-3 e da Fundação de Amparo à Pesquisa do Estado do Pará (FAPESPA) e também do Edital Universal N° 003/2008. Os equipamentos utilizados para a realização deste trabalho foram doados pela Intel. Os autores agradecem também ao programa de bolsas PIBIC / UFPA, pelo apoio financeiro ao primeiro autor.

Referências

AVIS. 2008. AVIS Event Router. Disponível em: http://avis.sourceforge.net. Acesso em: 14/07/2008.

BIEHL, J.T.; CZERWINSKI, M.; SMITH, G.; ROBERTSON, G.G. 2007. FASTDash: a visual dashboard for fostering awareness in software teams. *In*: SIGCHI CONFERENCE ON HUMAN FACTORS IN COMPUTING SYSTEMS, San Jose, CA, EUA, 2007. *Anais*... San Jose, CHI '07. ACM, p. 1313-1322.

BOWMAN, I.T.; HOLT, R. 1999. Reconstructing ownership architectures to help understand software systems. *In*: INTERNATIONAL WORKSHOP ON PROGRAM COMPREHENSION, Pittsburgh, 1999. *Anais*... IEEE Press, p. 28-37.

CATALDO, M.; WAGSTROM, P.A.; HERBSLEB, J.D.; CARLEY, K.M. 2006. Identification of coordination requirements: Implications for the design of collaboration and awareness tools. *In*: CONFERENCE ON COMPUTER SUPPORTED COOPERATIVE WORK, 20, Alberta, 2006. *Anais*... ACM Press, Banff, p. 353-362.

CHENG, L.; HUPFER, S.; ROSS, S.; PATTERSON, J. 2003. Jazzing up Eclipse with collaborative tools. *In:* OOPSLA WORKSHOP ON ECLIPSE TECHNOLOGY EXCHANGE, Anaheim, 2003. *Anais*... Anaheim, p. 45-49.

CONWAY, M.E. 1968. How Do Committees invent? *Datamation*, **14**(4):28-31.

COSTA, J.M.R. 2009. *TransFlow - Uma ferramenta para análise do processo de desenvolvimento de software livre*. Belém, PA. Trabalho de Conclusão de Curso. Universidade Federal do Pará, 98 p.

CVS. 2009. Concurrent Versions System. Disponível em: http://www.nongnu.org/cvs. Acesso em: 10/02/2009.

DE SOUZA, C.R.B. 2005. *On the relationship between software dependencies and coordination: field studies and tool support*. Irvine, CA, USA. Dissertação de PhD. University of California, Irvine, 186 p.

DE SOUZA, C.R.B.; REDMILES, D. 2008. An empirical study of software developers' management of dependencies and changes. *In*: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, 30, Leipzig, 2008. *Anais*... Leipzig, p. 241-250.

DE SOUZA, C.R.B.; DOURISH, P.; REDMILES, D.; QUIRK, S.; TRAINER, S. 2004. From technical dependencies to social de-

pendencies. *In*: WORKSHOP ON SOCIAL NETWORKS FOR DESIGN AND ANALYSIS: USING NETWORK INFORMATION, Chicago, 2004. *Anais...* Chicago, p. 1-8.

DE SOUZA, C.R.B.; REDMILES, D.F.; DOURISH, P. 2003. "Breaking the code": Moving between private and public work in collaborative software development. *In*: INTERNATIONAL CONFERENCE ON SUPPORTING GROUP WORK (GROUP'2003), Sanibel Island, 2003. *Anais...* Sanibel Island, p. 105-114.

DOURISH, P.; V. BELLOTTI. 1992. Awareness and coordination in shared workspaces. *In*: CONFERENCE ON COMPUTER-SUPPORTED COOPERATIVE WORK (CSCW '92), Toronto, 1992. *Anais...* Toronto, p. 107-114.

DOURISH, P. 2006. Implications for design. *In:* SIGCHI CONFERENCE ON HUMAN FACTORS IN COMPUTING SYSTEMS, Montréal, 2006. *Anais...* ACM, New York, p. 541-550.

FITZPATRICK, G.; MARSHALL, P.; PHILLIPS, A. 2006. CVS integration with notification and chat: lightweight software team collaboration. *In*: ANNIVERSARY CONFERENCE ON COMPUTER SUPPORTED COOPERATIVE WORK, 20, Banff, 2006. *Anais...* Banff, CSCW '06. ACM, New York, p. 49-58.

GRINTER, R.E. 2003. Recomposition: Coordinating a web of software dependencies. *Journal of Computer Supported Cooperative Work*, **12**(3):297-327.

GRINTER, R.E. 1995. Using a configuration management tool to coordinate software development. *In*: CONFERENCE ON ORGANIZATIONAL COMPUTING SYSTEMS, Milpitas, 1995. *Anais...* Milpitas, p. 168-177.

GRUDIN, J. 1994. Computer-supported cooperative work: History and focus. *IEEE Computer*, **27**(5):19-26.

HALVERSON, C.A.; ELLIS, J.B.; DANIS, C.; KELLOGG, W.A. 2006. Designing task visualizations to support the coordination of work in software development. *In*: ANNIVERSARY CONFERENCE ON COMPUTER SUPPORTED COOPERATIVE WORK, 20, Banff. *Anais...* Banff, p. 39-48.

HERMAN, I.; MELANCON G.; MARSHALL, M.S. 2000. Graph visualization and navigation in information visualization: A survey. *IEEE Transactions on Visualization and Computer Graphics*, **6**(1):24-43.

KAGDI, H.; MALETIC, J.I. 2007. Combining single-version and evolutionary dependencies for software-change prediction. *In*: INTERNATIONAL WORKSHOP ON MINING SOFTWARE REPOSITORIES, 4, Minneapolis. *Anais...* Minneapolis, p. 107-110.

MACCORMACK, A.; RUSNAK, J.; BALDWIN, C.Y. 2004. Exploring the structure of complex software designs: An empirical study of open source and proprietary code. *Management Science*, **52**(7):1015-1030.

MCDONALD, D.W.; ACKERMAN, M.S. 1998. Just talk to me: A field study of expertise location. *In*: CONFERENCE ON COMPUTER SUPPORTED COOPERATIVE WORK (CSCW '98), Seattle, 1998. *Anais...* Seattle, p. 315-324.

MORELLI, M.D.; EPPINGER, S.D.; GULATI, R.K. 1995. Predicting technical communication in product development organizations. *IEEE Transactions on Engineering Management*, **42**(3):215-222.

PARNAS, D.L. 1972. On the criteria to be used in decomposing systems into modules. *Communications of the ACM*, **15**(12):1053-1058.

PERRY, D.E.; WOLF, A.L. 1992. Foundations for the study of software architecture. *ACM SIGSOFT Software Engineering Notes*, **17**(4):40-52.

PERRY, D.E.; HARVEY P.S.; VOTTA, L.G. 2001. Parallel changes in large-scale software development: An observational case study. *ACM Transactions on Software Engineering and Methodology*, **10**(3):308-337.

PREFUSE. 2009. Information Visualization Toolkit. Available at: http://prefuse.org/. Accessed on: 10/02/2009.

REN, X.; SHAH, F.; TIP, F.; RYDER, B.G.; CHESLEY, O. 2004. Chianti: a tool for change impact analysis of java programs. *In*: ACM SIGPLAN CONFERENCE ON OBJECT-ORIENTED PROGRAMMING, SYSTEMS, LANGUAGES, AND APPLICATIONS, 19, New York, 2004. *Anais...* New York, **39**(10):432-448.

SARMA, A.; NOROOZI, Z.; VAN DER HOEK, A. 2003. Palantír: Raising Awareness among Configuration Management Workspaces. *In*: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, 25, Portland, 2003. *Anais...* Portland, p. 444-453.

SOUSA JÚNIOR, S.F.; BAILEIRO, M.; COSTA, J.M.R.; SOUZA, C.R.B. 2009. Multiple social networks study using sargas. *In*: HAWAII INTERNATIONAL CONFERENCE ON SYSTEM SCIENCES, Waikoloa, 2009. *Anais...* Waikoloa, p. 1-10.

SUBVERSION. 2009. Available at http://subversion.tigris.org. Accessed on: 10/02/2009.

TRAINER, E.; QUIRK, S.; DE SOUZA, C.R.B.; REDMILES, D. 2005. Bridging the gap between technical and social dependencies with Ariadne. *In*: Eclipse Technology Exchange, San Diego, 2005. *Anais...* New York, ACM, p. 26-30.

WOHLIN, C.; RUNESON, P.; HÖST M.; OHLSSON M. C.; REGNELL B.; WESSLÉN A. 2000. *Experimentation in Software Engineering: An introduction*. Norwell, Kluwer Academic Publishers, 228 p.

ZIMMERMANN, T.; WEIBGERBER, P.; DIEHLI, S.; ZELLER, A. 2003. Mining version histories to guide software changes. *IEEE Transactions on Software Engineering*, **31**(6):429-445.

Submitted on February 21, 2009.
Accepted on March 19, 2009.