

UNIVERSIDADE FEDERAL DE JUIZ DE FORA
INSTITUTO DE CIÊNCIAS EXATAS
BACHARELADO EM SISTEMAS DE INFORMAÇÃO

GiveMe Trace
Uma ferramenta de apoio à rastreabilidade de *software*

Cláudio Augusto Silveira Lélis

JUIZ DE FORA
DEZEMBRO, 2014

GiveMe Trace
Uma ferramenta de apoio à rastreabilidade de *software*

CLÁUDIO AUGUSTO SILVEIRA LÉLIS

Universidade Federal de Juiz de Fora
Instituto de Ciências Exatas
Departamento de Ciência da Computação
Bacharelado em Sistemas de Informação

Orientador: Marco Antônio Pereira Araújo
Co-orientador: José Maria Nazar David

JUIZ DE FORA
DEZEMBRO, 2014

Resumo

A rastreabilidade é um importante fator na análise de impacto das mudanças que o *software* sofre ao longo de sua evolução. As abordagens atuais provêem uma análise em termos de artefatos impactados. O presente trabalho apresenta a ferramenta *GiveMe Trace*, desenvolvida como um *plugin* do ambiente Eclipse capaz de, a partir da análise do repositório de versões, gerar rastreabilidade entre os artefatos de código fonte e suas versões evidenciando que houve alterações de método. É apresentada ainda uma integração desta ferramenta com um sistema de acompanhamento de mudanças de modo a relacionar um *ticket* com as modificações realizadas. Espera-se com isso aumentar a precisão na análise de impacto das mudanças e, a partir da integração, auxiliar na tarefa de verificação da mudança implementada.

Sumário

1	Introdução	3
2	Rastreabilidade de Software	6
3	Mapeamento Sistemático	16
3.1	Planejamento	16
3.2	Execução	17
3.3	Análise dos Resultados e Trabalhos Relacionados	18
4	<i>GiveMe Views</i>	24
5	<i>GiveMe Trace</i>	31
5.1	Arquitetura	31
5.2	Modo <i>Standalone</i>	34
5.3	Modo Integrado	39
5.3.1	Integração com <i>Mantis Bug Tracker (MantisBT)</i>	39
5.3.2	Integração com <i>GiveMe Views</i>	41
6	Prova de Conceito	44
6.1	Teste 1	44
6.2	Teste 2	46
6.3	Teste 3	48
6.4	Teste 4	50
7	Considerações Finais e Trabalhos Futuros	52
	Referências Bibliográficas	56

1 Introdução

A rastreabilidade de *software* pode ser entendida como a possibilidade de rastrear os artefatos de *software* que se relacionam em qualquer ponto ao longo do ciclo de vida do *software*. Os benefícios da rastreabilidade são percebidos em várias tarefas chave do desenvolvimento e evolução do *software* como por exemplo, a compreensão, verificação, validação, e principalmente na análise do impacto de mudanças e no teste do *software* (Walters et al., 2014). Um importante aspecto de análise na evolução do *software* é poder registrar as mudanças que ocorrem com os artefatos como, por exemplo código fonte, ao longo do tempo. Um sistema de controle de versão tem este objetivo. Uma versão, por sua vez é representada por um número ou código que relaciona vários ou apenas um artefato a um momento do ciclo de vida do *software*. O *GIT* (Chacon, 2009) ou *Subversion* (Pilato et al., 2008), exemplos de sistemas de controle de versão, gerenciam dependências entre versões do código fonte (Mohan et al., 2008), são capazes de gerar *logs* do histórico destas versões e apresentar a alteração sofrida por cada arquivo ao longo das versões. Na literatura encontram-se diferentes propostas para a geração e recuperação das ligações (*links*) de rastreabilidade entre os diferentes artefatos, seja entre o documento de requisitos e os modelos da fase de projeto (Sengupta et al., 2008), ou entre planos de teste e o código fonte (Wiederseiner et al., 2011), ou ainda entre o código fonte e suas diferentes versões (Kagdi et al., 2007). Para cada proposta existem diferentes abordagens (Rochimah et al., 2007) que podem ser adotadas na geração dos *links* de rastreabilidade. No entanto, as propostas que geram a rastreabilidade de *software* considerando as relações com artefatos de código fonte, não se preocupam com o relacionamento a um nível de granularidade mais baixa, como métodos, atributos e linhas de código. A ferramenta *GiveMe Views* (Tavares et al., em fase de elaboração) é um exemplo que se beneficiaria caso houvesse um recurso que gerasse dados de rastreabilidade em mais baixo nível. Trata-se de uma ferramenta capaz de indicar módulos e componentes que possuem uma probabilidade estatística de serem impactados caso ocorra uma manutenção em outro módulo ou componente. Com dados em um nível de abstração menor, ela poderia prover indicações mais precisas informando

a relação de impacto entre métodos. As lacunas apresentadas pela ferramenta *GiveMe Views* compartilhada por outras ferramentas apresentadas pela literatura justificam o desenvolvimento deste trabalho. Segundo Javed e Zdun (2014), uma abordagem que oferece informações em mais baixo nível, tende a ser mais precisa, ficando clara a importância de sua adoção (Kagdi et al., 2007) (Motta, 2013).

Considerando o contexto apresentado, definiu-se o objetivo geral de elaborar um mecanismo automatizado para geração de informações de rastreabilidade entre artefatos de código fonte e suas diferentes versões considerando uma granularidade mais fina, em termos de método.

A partir do objetivo geral, objetivos específicos foram definidos: (i) desenvolver a ferramenta *GiveMe Trace* que implementa o mecanismo elaborado, como um *plugin* do ambiente Eclipse capaz de, a partir da análise do repositório de versões, gerar *links* de rastreabilidade entre os artefatos de código fonte e suas versões, evidenciando as alterações nos métodos. (ii) Integrar a ferramenta desenvolvida com outras (*Mylyn-Mantis* e *GiveMe Views*) demonstrando seu poder de uso e os benefícios esperados com a integração, (iii) Aplicar uma prova de conceito de forma a verificar os dados gerados pela ferramenta a partir das funcionalidades providas.

A metodologia utilizada durante a execução do trabalho obedeceu uma sequência, iniciada pelo levantamento do referencial teórico, agregando os principais conceitos envolvidos no escopo deste trabalho desde a conceituação de *software* abordando o processo de desenvolvimento permeando os ciclos de vida e artefatos gerados, a evolução do *software* ao longo das manutenções, a disciplina de gerência de configuração com suas atividades de controle de versão e controle de mudanças, e por fim demonstrando que a rastreabilidade tem participação em todos estes pontos da vida de um *software*.

Em seguida, a definição e aplicação de um mapeamento sistemático sobre a rastreabilidade de *software* focando nas ligações entre os artefatos de código fonte e suas diferentes versões armazenadas nos sistemas de controle de versão. O resultado demonstrou uma deficiência de abordagens que contemplem a rastreabilidade com o código fonte a nível de método. Foi então iniciado o processo de análise da ferramenta *GiveMe Views* buscando compreender seu funcionamento, suas origens, características e limitações.

Com base nas limitações identificadas e nos resultados do mapeamento sistemático, foi elaborado um mecanismo automatizado, posteriormente implementado como um *plugin* do Eclipse, chamado *GiveMe Trace*, capaz de gerar *links* de rastreabilidade entre as versões geradas ao longo das modificações e os métodos efetivamente alterados. Para mostrar a capacidade de integração da ferramenta desenvolvida, foram demonstradas integrações feitas com duas ferramentas, a primeira é um sistema de gerenciamento de solicitações de mudança e a outra ferramenta é o *GiveMe Views*. Os benefícios que podem ser obtidos com a utilização da ferramenta *GiveMe Trace* foram também mostrados para cada caso de integração.

Uma prova de conceito foi elaborada e aplicada à ferramenta desenvolvida *GiveMe Trace* para se averiguar a adequação das funcionalidades propostas com as informações geradas de fato. As funcionalidades foram divididas em quatro hipóteses a serem verificadas. A prova de conceito apresentou indícios de que o *GiveMe Trace*, de fato, gera as informações conforme o esperado.

O presente trabalho se divide em sete capítulos. O primeiro capítulo consiste nesta introdução. No segundo capítulo consiste do referencial teórico onde são apresentados os principais conceitos e características presentes na literatura sobre rastreabilidade de *software*. Em seguida, é apresentado um mapeamento sistemático com o intuito de elencar os trabalhos relacionados e estudos que demonstram os benefícios da rastreabilidade durante o processo de manutenção do *software*. No quarto capítulo é apresentada a ferramenta *GiveMe Views* que serviu de inspiração para que a solução fosse implementada na forma de um *plugin*. No quinto capítulo, é apresentada a solução propriamente dita. A ferramenta *GiveMe Trace* é detalhada e sua capacidade de integração é demonstrada. Em seguida, a prova de conceito aplicada à ferramenta e seus resultados são expostos. Por fim, as considerações assinalando as limitações da ferramenta *GiveMe Trace*, as dificuldades encontradas, as limitações da prova de conceito e o que se espera com o uso da ferramenta. Sugere-se ainda, possibilidades de aprofundamento posterior, além de propostas para trabalhos futuros.

2 Rastreabilidade de Software

A definição de *software* é amplamente difundida como uma sequência de instruções escritas para serem interpretadas por um computador com o objetivo de executar tarefas específicas (IEEE, 1990) ou ainda como os programas que comandam o funcionamento de um computador. Neste capítulo serão elucidados os principais conceitos que permeiam o *software* e a sua relação com a rastreabilidade.

A engenharia de *software* é a aplicação de uma abordagem sistêmica, disciplinada e quantificável no desenvolvimento, na operação e na manutenção de *software*, isto é, a aplicação da engenharia ao *software* (IEEE, 1990). Entretanto, segundo Pressman (2011) é preciso disciplina sim mas, também, adaptabilidade e agilidade com foco na qualidade.

A engenharia de *software* é uma tecnologia composta por camadas, onde o foco na qualidade é a base de sustentação desta estrutura, seguida pela camada de processos. O processo define uma metodologia que deve ser estabelecida para a entrega efetiva de tecnologia de engenharia de *software* e possibilita o desenvolvimento de *software* de forma racional e dentro do prazo. As demais camadas são compostas pelos métodos e as ferramentas da engenharia de *software*. Os métodos são responsáveis por fornecer as informações técnicas para desenvolver *software*, envolvendo uma ampla gama de tarefas que incluem a comunicação, a análise de requisitos, a modelagem do projeto, a construção do programa, os testes e então o suporte. As ferramentas fornecem suporte automatizado ou semiautomatizado aos processos e métodos (Pressman, 2011).

De modo geral, processo é um conjunto de atividades, ações, e tarefas realizadas na criação de algum produto de trabalho (Pressman, 2011). Para a engenharia de *software*, trata-se de uma abordagem adaptável e não uma prescrição rígida do desenvolvimento de um *software*. Um processo de engenharia de *software* completo é estabelecido através de uma metodologia de processo, um *framework*, que identifica um conjunto de atividades aplicáveis a todos os projetos de *software* desprezando tamanho ou complexidade dos mesmos. Uma metodologia genérica para engenharia de *software* é formada por cinco atividades: a comunicação cujo intuito é compreender os objetivos e necessidades das

partes interessadas para com o projeto, o planejamento onde se define o trabalho de engenharia de *software* descrevendo as tarefas técnicas a serem conduzidas, cronograma, produtos gerados, recursos necessários e riscos associados. Em seguida, a modelagem onde são criados os modelos para melhor entender as necessidades do *software* e o projeto que irá atendê-las. A próxima tarefa é a construção que combina a geração de código seja manualmente ou automático e os requeridos testes para identificar erros no percurso. E, por fim, o emprego, no qual o *software* é entregue ao cliente, e este pode avaliar e retornar suas impressões. O processo de *software* genérico composto por estas cinco atividades descritas, também recebe o nome de ciclo de vida de *software* (Sommerville, 2008). Ao longo da execução de cada atividade, ao final tem-se a geração dos chamados artefatos de *software* por exemplo documentos, modelos, projetos, planos, listas, *logs*, código fonte, enfim, sendo o *software* muito mais que simplesmente um conjunto de instruções que, quando executadas, retornam informações e desempenho desejados. Em muitos casos o ciclo de vida é aplicado repetidamente produzindo um incremento de *software* a cada rodada, desta forma se torna, com o passar dos ciclos, mais completo (Pressman, 2011).

Desde a primeira fase do processo de *software* onde se elicitam os chamados requisitos dos interessados pelo sistema, é possível iniciar o processo de rastreabilidade de *software*. Ao avançar para as fases seguintes o requisito, este importante artefato de *software*, será o responsável pela geração dos demais artefatos. A capacidade de criar a relação entre os artefatos gerados ao longo do ciclo de vida, é a chamada rastreabilidade de *software* (Sommerville, 2008).

Acompanhando as atividades metodológicas existem as chamadas atividades de apoio, estas ocorrem ao longo do processo de *software* e se concentram principalmente, no gerenciamento, no acompanhamento e controle do projeto. São as chamadas atividades “guarda-chuva”. Dentre elas faz-se indispensável citar a administração de riscos que avaliam os possíveis riscos que venham a afetar a qualidade do produto ou do projeto geral, o controle e acompanhamento do projeto que permite avaliar o progresso do projeto e tomar medidas para cumprir o previsto. Outra atividade é o gerenciamento da configuração de *software* que gerencia as mudanças ao longo da vida do *software* e o impacto dessas mudanças através da rastreabilidade dos artefatos envolvidos na alteração, e a garantia

da qualidade de *software*, responsável por definir, conduzir e manter as atividades que garantem a qualidade do *software*.

O incremento dado ao *software* a cada ciclo de vida, é uma prova de que o *software* cresce e evolui, mas não é só durante o processo de desenvolvimento que esse crescimento é percebido. Após a implantação do *software* ao fim do processo de *software*, ele deve inevitavelmente mudar para continuar útil. Com o passar do tempo, as necessidades e interesses dos clientes mudam, leis podem surgir e os negócios podem tomar rumos diversos levantando a obrigatoriedade de novas mudanças. Melhoria de desempenho, adaptação a uma nova plataforma e erros encontrados durante a operação do *software* também levam a mudanças. A dinâmica da evolução de programas é o estudo de mudanças do sistema (Sommerville, 2008). Portanto, entender como o *software* vai mudando ao longo do tempo, leva à compreensão de sua evolução no período.

Sommerville (2008) define a manutenção de *software* como um processo geral de mudanças de um sistema depois que ele é entregue, e classifica em três os tipos de manutenção de *software*. A manutenção para reparo de defeitos ou também manutenção corretiva é o primeiro deles, e defeitos podem ser tanto erros de codificação, quanto erros no projeto ou ainda erros ao levantar os requisitos dos clientes. O segundo é a manutenção para adaptar o *software* a um ambiente operacional diferente ou simplesmente manutenção adaptativa, é necessária quando o hardware ou o sistema operacional ou outro *software* de apoio se alteram obrigando o sistema a se adequar. Por fim, a manutenção para adicionar funcionalidade ao sistema ou modificá-la que seria a também chamada manutenção evolutiva, ocorrendo quando mudanças organizacionais ou de negócios são detectadas.

Mudanças são inevitáveis quando o *software* é construído. À medida que o sistema cresce ele se torna complexo e difícil de alterar, assim mais difícil de ser compreendido acarretando a inserção de defeitos por parte dos programadores e, como consequência direta, reduzindo a qualidade do *software* (Sommerville, 2008). Não analisar as mudanças antes de serem feitas, não registrá-las antes de serem implementadas e não divulgá-las a quem precisa, agrava ainda mais a situação. A gestão de configuração de *software* (*software configuration management - SCM*) é uma atividade aplicada ao longo de todo o processo

de desenvolvimento de um *software* e está relacionada à gestão da qualidade. Trata-se de um conjunto de atividades de rastreamento e controle iniciadas quando um projeto de engenharia de *software* começa, e só termina quando o *software* sai de operação. Como as mudanças podem ocorrer em qualquer instante, as atividades SCM são desenvolvidas para registrar e processar mudanças de sistema, relacioná-las aos componentes do sistema e os métodos usados para identificar diferentes versões dele. Além de controlar a alteração, assegurar que a implementação da mesma seja feita corretamente, bem como relatar estas alterações aos interessados (Pressman, 2011).

As mudanças levam às novas versões do sistema e a gerência impede que se percam os relacionamentos, também chamados, *links* de rastreabilidade de quais mudanças foram incorporadas em qual versão do sistema. A ineficiência do gerenciamento de configuração definido pode levar ao desperdício de esforços ao modificar uma versão errada do *software*, entregar a versão errada ao cliente ou ainda, perder a rastreabilidade de onde o código fonte está armazenado (Sommerville, 2008).

É responsabilidade dos gerentes de configuração, não apenas manter a rastreabilidade das diferenças entre versões de *software* garantindo que as novas versões sejam derivadas de maneira controlada, mas liberar novas versões para clientes certos no momento certo. Grandes sistemas podem conter módulos com incontáveis linhas códigos, modelos, centenas de casos de teste, enfim, produzidos com nomes similares ou idênticos por pessoas distintas. Manter a rastreabilidade num cenário como este exige um esquema de identificação para todos os itens do SCM. Os itens ou classe de itens a serem controlados é definida na fase de planejamento da gerência de configuração. Esses documentos, uma vez sob controle de configuração, são chamados documentos controlados ou itens de configuração, e eles podem ser por exemplo, especificações, modelos, projetos, massa de dados de teste, enfim tudo que possa ser usado na evolução do sistema futuramente. A estes documentos é atribuído um nome único de fácil compreensão.

Os itens de configuração ficam armazenados em bancos de dados de projeto ou repositórios (Pressman, 2011). Segundo (Sommerville, 2008), este banco de dados é utilizado para registrar todas as informações relevantes sobre as configurações do sistema, itens de configuração e, opcionalmente, dados de clientes, plataformas de execução e mu-

danças propostas, entre outros. Além disso, para auxiliar a avaliação de impacto das mudanças de sistema e a geração de relatórios.

Alteração é um fato normal no desenvolvimento do *software*. Para garantir que as mudanças serão aplicadas ao sistema da maneira correta tomando-se as precauções de controle, é preciso um conjunto de procedimentos de gerenciamento de mudança apoiado por ferramentas. Tais procedimentos, dizem respeito à análise de custo benefício das mudanças propostas, à aprovação de mudanças viáveis e à rastreabilidade de quais componentes do sistema foram alterados (Sommerville, 2008).

O processo de controle de mudança descrito por Sommerville (2008), se aproxima mais da realidade e do enfoque que se quer dar neste ponto do trabalho. É iniciado descrevendo a mudança necessária para o sistema através de um formulário de solicitação de mudança (*CRF – Change Request Form*). Neste formulário registram-se as recomendações sobre a mudança, os custos estimados e as datas de quando ela foi solicitada, aprovada, implementada e validada. Pode ainda incluir uma seção onde o analista descreve como a mudança será implementada. Uma vez recebida a solicitação de mudança, ela deve ser registrada no banco de dados de configuração. O passo seguinte é analisar para verificar se a mudança solicitada é de fato necessária, não é proveniente de um mal-entendido, ou de um defeito já conhecido. Chegando a esta conclusão a solicitação é rejeitada. Será rejeitada também nos casos de solicitação ser julgada inválida, duplicada, ou já tratada. A origem da solicitação então deve ser notificada sobre os motivos da rejeição.

O passo seguinte, considerando as mudanças válidas, é a avaliação da mudança e o custo. O impacto da mudança no restante do sistema deve ser verificado. Isso envolve a identificação de todos os componentes afetados pela mudança usando informações do banco de dados de configuração e do código-fonte do *software*. Se for necessária mudança adicional em alguma parte do sistema, o custo se eleva. Em seguida, avalia-se as mudanças para os módulos do sistema e, finalmente, os custos individuais dos componentes é considerado e o custo total da mudança é estimado (Sommerville, 2008). Durante este processo, é por meio da rastreabilidade entre os artefatos que o impacto da mudança é analisado. É preciso manter a corretude dos *links* de rastreabilidade para que não se encorra em erro ao estimar o custo, por desconsiderar componentes que também serão

afetados mas não possuem o *link* atualizado.

As decisões de aprovação ou rejeição, revisão, análise de impacto e custo, além da priorização das mudanças aceitas são feitas pelo comitê de controle de mudanças (*Change Control Board* - *CCB*). Em projetos pequenos pode ser formado pelo gerente de projeto e um engenheiro não diretamente envolvido no desenvolvimento (Pressman, 2011).

Outra atividade que compõe a gerência de configuração, é a gestão de versões e *releases*, cujo foco é a identificação e a manutenção da rastreabilidade das versões de um sistema. As versões são as instâncias do sistema que se diferenciam umas das outras em algum aspecto, seja um erro corrigido, um aperfeiçoamento no desempenho, funcionalidades ou ainda características de hardware. Versões, também chamadas revisões são criadas para uso interno e não há o costume de serem lançadas para o cliente. Quando o produto é construído para ser entregue ou liberado ao cliente, este produto é o *release*. Cria-se uma versão também, a partir da junção de versões individuais de cada componente, para esta tarefa a técnica mais usada de identificação de versão é a numeração das versões onde cada componente possui como sufixo um número de versão. Apesar desta abordagem ser simples, o nome não informa nada sobre a versão e o motivo de sua criação, com isso é preciso guardar essas informações no banco de dados de configuração. Desta forma, não se perde os dados de rastreabilidade das diferenças entre versões e relacionamentos entre propostas de mudança do sistema e as versões. Em último caso é possível também ligar explicitamente as solicitações de mudança para as diferentes versões de cada componente (Sommerville, 2008).

O apoio de ferramentas é essencial para o gerenciamento de configuração, dada a criticidade do processo. Podem trabalhar integradas ou individualmente. Recursos integrados tendem a ser complexos e dispendiosos fazendo muitas organizações optarem por ferramentas de apoio individuais, mais baratas e simples.

Para apoiar o gerenciamento de mudanças existem as ferramentas mais simples de código aberto como *Bugzilla* e *Mantis Bug Tracker*, e outros mais complexos e integrados. De forma geral, Sommerville (2008) descreve recursos que as ferramentas buscam prover, tais como: (i) um editor de formulários permitindo a criação dos formulários com propostas de mudança; (ii) um sistema de fluxo de trabalho (*workflow*) que permite definir

quem deve processar o formulário de mudança e sua ordem de processamento. Também é função deste sistema informar aos interessados, no tempo e hora certa, o progresso da mudança. Outro recurso é (iii) um banco de dados de mudança, usado para gerenciar todas as propostas de mudança. Seu sistema de consulta permite encontrar uma solicitação de mudança específica, e ainda pode estar ligado, ou não, a um sistema de gerenciamento de versões. Há ainda (iv) um sistema de relato de mudanças que gera relatórios gerenciais.

O gerenciamento de versões lida com grandes volumes de informação, assim o uso de ferramentas também é essencial. Essas ferramentas controlam repositórios de itens de configuração nos quais os conteúdos não podem ser diretamente alterados. Para manipular um item de configuração é preciso retirá-lo do repositório, o chamado *checkout*, e guardá-lo num diretório local de trabalho gerando uma cópia de trabalho do item, o chamado *work-copy*. Depois de realizadas as alterações no item, deve-se devolvê-lo ao repositório, agora o *checkin* ou também chamado de *commit*, e uma nova versão é automaticamente criada. Para completar corretamente este processo as ferramentas devem ter certas capacidades descritas por Sommerville (2008) como a identificação de versões e releases, a capacidade de identificar versões gerenciadas, isso é possível pois estas recebem identificadores únicos quando enviadas ao sistema. Gerenciamento de armazenamento permite que as versões sejam descritas pela sua diferença em relação a uma versão mestre. Isso poupa espaço de armazenamento das várias versões cujas diferenças geralmente são pequenas. Outra capacidade é o registro de histórico de mudanças que de fato registra toda alteração que é feita a um artefato sendo possível listar essas mudanças, por meio delas é possível selecionar uma versão específica do sistema. Num cenário realista de trabalho dois desenvolvedores podem atuar sobre a mesma *release* do sistema. O sistema de gerência de versões mantém a rastreabilidade dos componentes retirados para edição, de modo que as mudanças efetuadas pelos desenvolvedores não interfira no componente controlado. Por fim, sistemas que dão suporte a projetos possibilitam o checkout de todos os arquivos associados a um projeto com todos os dados das versões. Sistemas como *GIT* e *Subversion* suportam essas capacidades.

A rastreabilidade, por acompanhar toda a vida de um *software*, consiste em peça chave trazendo benefícios em seu uso. Segundo Spanoudakis e Zisman (2005), ela oferece

dados para a análise de corretude e completude do *software* além da estimativa do custo envolvido em alterações. A identificação de conflitos entre artefatos, os riscos associados a integrações com outros sistemas e a detecção precoce de erros, são algumas das vantagens conseguidas com a rastreabilidade.

A natureza destas relações de rastreabilidade entre os artefatos pode variar, são oito os tipos mais conhecidos. Segundo Spanoudakis e Zisman (2005) o primeiro deles é a dependência, no qual um artefato A depende do artefato B se o A invoca o B, ou se uma mudança em A reflete em B. Generalização é utilizado para identificar como artefatos podem ser refinados. Na relação de evolução, o artefato A evolui para o artefato B, e A foi substituído pelo B durante o desenvolvimento, manutenção ou evolução do sistema de *software*. Na satisfatibilidade um artefato A satisfaz artefato B, se A atende as expectativas de B ou se A está satisfeito com as condições representadas por B. Na sobreposição, diz-se que um artefato A sobrepõe artefato B, se A e B referem-se à mesma funcionalidade ou ao mesmo domínio. A relação de conflito indica conflito de funcionalidade entre dois artefatos, a análise lógica representa a lógica da criação e evolução dos artefatos e a contribuição demonstra a associação entre os artefatos, e os *stakeholders* que contribuíram para a geração dos requisitos.

A identificação e geração desses relacionamentos de rastreabilidade entre artefatos variam também com base no nível de automação oferecido pelas ferramentas de suporte à geração de rastreabilidade. Segundo o Centro de Excelência em Rastreabilidade de *Software* (COEST), a criação manual do *link* de rastreabilidade depende da ação humana. Isso inclui a criação de rastreabilidade e manutenção usando o arrastar e soltar, métodos que são comumente encontrados em ferramentas de gerenciamento de requisitos atuais. A ferramenta exhibe ao *stakeholder*, pessoa envolvida no projeto, os artefatos a serem rastreados, e assim ele pode arrastar e soltar definindo a relação entre estes artefatos. Embora ferramentas de rastreabilidade manual ofereçam suporte à gestão das relações de rastreabilidade e minimização do tempo necessário para encontrar os *links* desejados, os benefícios da rastreabilidade são reduzidos devido ao enorme esforço para a criação e manutenção de relações de rastreabilidade. Apesar de ser difícil, complexa e dispendiosa, a rastreabilidade manual é útil, por exemplo, quando o julgamento humano é necessário

para estabelecer as ligações de rastreabilidade ou, quando um número limitado de ligações devem ser especificadas porém são difíceis de automatizar.

Um outro nível de automação é a semi-automática, quando a rastreabilidade é estabelecida através de uma combinação de técnicas automatizadas, métodos, ferramentas e atividades humanas. Por exemplo, técnicas automatizadas podem sugerir *links* de rastreamento candidatos ou *links* suspeitos e depois o *stakeholder* pode ser solicitado a verificá-los. O contrário também é possível, o *stakeholder* indica um *link* candidato e uma ferramenta busca informações para conferir a relação. Nessa abordagem ainda é necessária a supervisão humana para analisar os resultados gerados e, possivelmente, para refazer o *link* de rastreamento. O terceiro nível, o automático, é quando a rastreabilidade é estabelecida por métodos, ferramentas e técnicas automatizadas, assim as decisões de como criar, onde criar, e quando criar o *link* e a sua devida manutenção é feita automaticamente. Apesar da não intervenção humana no processo, abordagens automáticas de geração da rastreabilidade tendem a gerar resultados com pouca acurácia e precisão, e a proliferar *links* de rastreabilidade desnecessários.

Além do tipo de relacionamento entre artefatos, do nível de automação oferecido pelas ferramentas de apoio, outra característica importante de variação da rastreabilidade é o nível de detalhe em que um link é estabelecido e mantido. A granularidade de uma relação depende da granularidade do artefato de origem e do artefato de destino. Segundo Javed e Zdun (2014) componentes e classes, por exemplo, possuem granularidade mais grossa, enquanto métodos, funções, parametros e atributos são considerados de mais fina granularidade.

Conforme o *software* evolui, mudam-se as relações existentes entre os artefatos, podem surgir novas relações ou as existentes são eliminadas. Para tratar essa inconstância, a rastreabilidade ainda agrega um processo de manutenção dos *links* definidos. São, na verdade, atividades associadas com a atualização de um único *link* pré-existente mediante a evolução do mesmo. Novos relacionamentos são estabelecidos quando necessários, para manter o *link* relevante e atualizado.

Diferentemente da rastreabilidade de requisito, que é focada na capacidade de seguir e descrever um requisito ao longo de toda sua vida, a rastreabilidade de *software* vai

além, traz uma percepção mais ampla ao conceito, englobando não somente os requisitos mas qualquer artefato da engenharia de *software* e seus inter-relacionamentos.

A relação entre os artefatos de código fonte ao longo do processo de evolução do *software*, suas mudanças registradas pelo sistema de solicitação de mudança e, consequentemente, as diferentes versões geradas na disciplina de Gerencia de Configuração através do sistema de controle de versão é o objeto de estudo deste trabalho.

3 Mapeamento Sistemático

Para a sequência do trabalho e de modo a compreender o estado da arte do cenário de pesquisa escolhido, foi desenvolvido um mapeamento sistemático cujo processo é descrito nos tópicos subsequentes. Segundo Kitchenham e Charters (2007) um mapeamento sistemático é projetado para fornecer uma visão geral de uma área de investigação. Também conhecidos como estudos prévios, podem indicar áreas adequadas para a realização de revisões sistemáticas da literatura (Kitchenham, 2004). O mapeamento sistemático foi aplicado seguindo três etapas: (i) o planejamento onde se elaborou o protocolo com os dados a serem observados, (ii) a execução do mapeamento propriamente dito, e (iii) a apresentação dos resultados.

3.1 Planejamento

O protocolo desenvolvido foi constituído de objetivo, questão de pesquisa, *string* de busca, critérios de inclusão e exclusão e bases de dados de publicações.

O objetivo é descobrir as ferramentas contempladas pela literatura que oferecem suporte à geração de rastreabilidade entre código fonte e suas versões.

Mediante o objetivo apresentado foi definida a questão de pesquisa, com o objetivo de guiar a execução do mapeamento sistemático. A questão de pesquisa definida foi a seguinte:

(Q1) Quais as ferramentas existentes para promover a rastreabilidade de software entre código fonte e suas diferentes versões?

A *string* de busca adotada foi:

(traceability or rastreabilidade) AND software

Os critérios de inclusão e exclusão encontram-se dispostos por tipo na Tabela 3.1.

Tipo (Inclusão/Exclusão)	Descrição
Inclusão	Estudos sobre ferramentas que geram a rastreabilidade de software no código fonte
Inclusão	Estudos que discutem os efeitos da rastreabilidade no código fonte
Inclusão	Estudos publicados a partir do ano de 2004
Exclusão	Estudos relatam informações sobre rastreabilidade em arquiteturas de hardware, requisitos, design de software, ou outros campos que não estão diretamente relacionados a rastreabilidade no código fonte
Exclusão	Estudos relacionados a código fonte mas não, ou só superficialmente, à rastreabilidade
Exclusão	Estudos que não puderem ser acessados completos de forma gratuita pela web
Exclusão	Estudos que tenham mais do que uma descrição publicadas serão incluídas apenas uma vez considerando a mais detalhada e atualizada versão do estudo
Exclusão	Estudos que não tratem da rastreabilidade na área de ciência da computação.

Tabela 3.1: Relação de critérios de inclusão e exclusão

Para executar a string de busca foram consideradas seis bases de dados eletrônicas de publicações com acesso livre para alunos da Universidade Federal de Juiz de Fora (UFJF), acessados via periódicos da Capes, os dados das bases encontram-se registrados na Tabela 3.2.

Bases de Publicações
Science@Direct (http://www.sciencedirect.com)
Compendex (http://www.engineeringvillage.com)
IEEE Digital Library (http://ieeexplore.ieee.org)
ACM Digital Library (http://portal.acm.org)
ISI Web of Science (http://www.isiknowledge.com)
Scopus (http://www.scopus.com)

Tabela 3.2: Relação das bases de dados de publicação

3.2 Execução

Após a definição do protocolo, a *string* de busca definida foi executada nas bases de publicações (Tabela 3.2), sendo inicialmente aplicada ao título dos artigos. Dos 423 retornos, uma primeira análise, automática, feita pela ferramenta de apoio *StArt* (StArt, 2014), classificou 150 registros como duplicados, esta classificação foi verificada e aprovada pelo condutor deste trabalho.

Nos 273 restantes observou-se a necessidade de uma nova análise, neste caso manual, em busca de outros resultados duplicados, desta segunda análise foram identificados 118 duplicatas usando como critério de análise a comparação entre os títulos, datas de publicação e as fontes.

A partir dos 154 trabalhos restantes, iniciou-se o processo de análise do resumo de cada trabalho. Da leitura do “*abstract*” foram selecionados 34 artigos, que após a análise do texto completo, 8 foram os resultantes.

Ao final do processo de execução do mapeamento, um trabalho de conclusão de curso (Motta, 2013), não indexado em nenhuma base de dados de publicações, sendo acessível apenas através do portal da instituição ao qual foi submetido, foi incluído manualmente. Seu conteúdo foi considerado relevante por abordar os principais conceitos do desenvolvimento de *plugins*, e estes, terem sido utilizados no desenvolvimento da ferramenta *GiveMe Trace*, além de apresentar como trabalhos futuros, algumas propostas de testes que acabaram sendo aplicadas ao *GiveMe Trace* logo ao término de sua implementação. A análise deste trabalho, e dos demais resultados selecionados é feita a seguir.

3.3 Análise dos Resultados e Trabalhos Relacionados

Em recente estudo, Javed e Zdun (2014) apresentaram uma revisão sistemática da literatura realizada com o objetivo de descobrir abordagens e ferramentas existentes para a rastreabilidade entre arquitetura de software e o código fonte. Após a execução da revisão, os resultados foram divididos em seis categorias principais para distinguir as diferentes abordagens e ferramentas encontradas. As categorias foram as seguintes: (i) abordagens de rastreabilidade entre arquitetura de software e código fonte, (ii) nível de automação das abordagens, (iii) tipo das relações de rastreabilidade, (iv) a granularidade suportada, (v) a direção da rastreabilidade e (vi) representação das informações de rastreabilidade. Como resultado, propôs um esquema de classificação baseado nos vários aspectos da rastreabilidade. Apesar do foco divergir, ele foi considerado pois traz importantes conceitos e uma percepção lúcida e direta das estratégias de solução para tratar a rastreabilidade de software além de um diagnóstico deste cenário de pesquisa em franca expansão.

Em outro trabalho, não recente mas bastante semelhante, Rochimah et al. (2007) utilizaram um *framework* para avaliar as diferentes abordagens existentes para tratar a rastreabilidade. Desta vez o foco era a evolução do software e os resultados mostraram que não há abordagens que satisfaçam completamente as capacidades relacionadas aos requisitos que têm de ser realizados para suportar a evolução do software.

Tanto Rochimah et al. (2007) quanto Javed e Zdun (2014) identificaram as mesmas abordagens. Por um lado, isso preocupa pois em nenhuma delas foi observado suporte

à evolução de software, por outro lado isso mostra que o campo de pesquisa está completamente aberto. Lembrando que os dois estudos tiveram sua seleção endossada por apresentarem forte conteúdo teórico servindo de base para a fundamentação e análise destes resultados.

Apesar das vantagens na utilização de rastreabilidade já serem conhecidas, Mader e Egyed (2011) fizeram um experimento com o objetivo de averiguar se os desenvolvedores se beneficiariam com a ajuda da rastreabilidade durante a navegação pelo código fonte durante as atividades de manutenção. Para isso, desenvolveram um experimento no qual 52 indivíduos deveriam realizar tarefas reais de manutenção em dado projeto. Para metade dos indivíduos, foi fornecido também um suporte de navegação automatizada baseado em informações de rastreabilidade com a capacidade de mostrar onde um requisito afetado pela mudança foi implementado, enquanto a outra metade de indivíduos, não pode contar com este suporte. Uma tarefa de mudança consiste em três atividades, inicialmente entender a requisição de mudança, em seguida procurar no código fonte os pontos relevantes para realizar a mudança e, por fim, implementar a alteração. Segundo Mader e Egyed (2011), a busca no código pelos pontos a alterar, segunda tarefa, poderia se beneficiar muito com a navegação automática baseada em rastreabilidade. Os resultados mostraram que aqueles desenvolvedores cuja navegação em rastreabilidade foi oferecida, desconsideraram o uso da navegação convencional, e o novo tipo de navegação proposta teve impacto profundo na performance, qualidade, precisão e no fluxo de trabalho de como eram trabalhadas as solicitações de mudança. Este trabalho foi considerado também pois demonstra os ganhos de se oferecer dados de rastreabilidade durante o processo de manutenção.

Realizar uma gerência efetiva dos artefatos durante a evolução do software e suas inúmeras mudanças ao longo do tempo é imprescindível, e trata-se de uma das atividades previstas pela Gerência de configuração. A rastreabilidade pode ajudar nesta tarefa. Enquanto a gerência de configuração provê meios de gerenciar, controlar, executar as mudanças e evolução do software, a rastreabilidade ajuda no gerenciamento de dependências entre os artefatos relacionados às diferentes fases do ciclo de vida do desenvolvimento. No entanto, na maioria das vezes essas duas práticas trabalham isoladamente. Mohan et al. (2008) apontam dois problemas nas abordagens de gerenciamento de mudanças, o

primeiro diz respeito a falta de suporte no processo de conhecimento, que pode rastrear dependências entre artefatos em diferentes fases do ciclo de vida de desenvolvimento, e o segundo é a falta de suporte para o gerenciamento do produto e do processo de conhecimento num grau mais fino de granularidade. Então desenvolveram um modelo de referência para integrar a rastreabilidade com o SCM. Numa segunda fase, uma ferramenta foi desenvolvida, denominada *Tracer*, que suporta a aquisição e uso do conhecimento de processo e produto representado no modelo de referência. Esta ferramenta suporta a criação de uma rede de rastreabilidade que representa a associação entre os diversos componentes de conhecimento originados de diferentes ferramentas usadas por desenvolvedores de software. *Tracer* possui integração com o MS Visual SourceSafe®, uma ferramenta de controle de versão de uso geral. Para tratar o problema de granularidade, *Tracer* facilita a documentação de mudanças implementadas em elementos específicos dentro das versões de vários artefatos de software e facilita a ligação entre as mudanças efetivadas e a solicitação que as originou. O relacionamento entre as mudanças pode ser feito em um nível mais fino de granularidade, por exemplo, uma classe específica ou um caso de uso pode ser ligado a um pedido de mudança para versões específicas deste artefato na ferramenta SourceSafe. Os conhecimentos gerados pela integração apresentada, auxilia na retenção de parte do conhecimento tácito que se perde na rotação de funcionários, além de ser útil em outras decisões de projeto semelhantes, acelerando o processo de decisão. Em essência, este conhecimento pode levar ao desenvolvimento de melhores práticas gerais que podem ser compartilhadas entre projetos. A ferramenta *Tracer* ilustra uma integração do controle de versão com rastreabilidade que pode ser generalizada para outras práticas e ferramentas utilizadas para gestão da mudança em SCM. Segundo Mohan et al. (2008), o modelo e a ferramenta desenvolvidos no trabalho, podem ser ampliados para suportar a integração, por exemplo, com uma ferramenta utilizada para acompanhar as solicitações de mudança, personalizando o modelo de rastreabilidade. Apesar desta possibilidade de ampliar o modelo desenvolvido, não fica claro como isso poderia ser efetivamente implementado. O suporte para apenas um sistema de controle de versão é também uma limitação observada. Quanto ao segundo problema levantado, tratando do nível de granularidade, apesar de níveis mais finos de granularidade terem sido alcançados relacionando, por exemplo,

um requisito específico a uma classe, ou a um caso de uso num diagrama UML, o foco é dado na relação com os documentos descritivos e modelos de projeto. Assim os desafios continuam, considerando as diferentes versões de código fonte.

Analisar repositórios das alterações feitas ao código fonte é foco de outras pesquisas também. A extração de *links* de rastreabilidade através das versões dos diferentes artefatos, e consequentemente do software como um todo, é uma realidade. A mineração das informações contidas nos bancos de dados de versões em busca do chamado “padrão de alteração”, foi a base que sustentou o trabalho de Kagdi et al. (2007). Segundo eles estes padrões, ao indicarem quais artefatos foram alterados juntamente com outros repetidas vezes, é possível inferir a existência de algum tipo de relação entre eles. Para demonstrar sua técnica, a mineração do histórico de versões encontradas em repositórios de software que são mantidos sob ferramentas de controle de versão tais como *Subversion* e *CVS*. Foram usadas duas técnicas de mineração, uma para identificar e analisar conjuntos de artefatos que foram commitados juntos, desta forma produzindo os padrões de alteração, e a outra técnica de mineração aplicada para considerar a ordem cronológica entre os diferentes padrões. Assim os padrões recuperados deram a ordem específica na qual os arquivos em um padrão foram alterados. Esta informação ordenada pode ser usada para inferir o tipo, a direção e os artefatos envolvidos na relação de rastreabilidade. Foi possível comprovar a alta precisão na predição dos links caso padrões similares tenham sido encontrados em versões anteriores. Apesar do resultado positivo, o autor salienta que uma análise numa granularidade mais fina, a nível de classe ou método, poderia produzir melhores resultados.

Em grandes empresas preocupadas com a qualidade de seus serviços e produtos, que seguem padrões e normas para estabelecer esta qualidade interna, a rastreabilidade serve como importante parâmetro de controle. Pequenas e médias empresas também devem atentar para a qualidade onde a rastreabilidade pode e deve ser usada. APIS (Neumuller e Grunbacher, 2006) é um ambiente implementado numa empresa Austríaca de pequeno porte, que com técnicas simples, gera rastreabilidade entre artefatos. A empresa já adotava algumas técnicas que favoreceram a implantação do sistema proposto, por exemplo, os desenvolvedores usam os ids dos artefatos armazenados em comentários para

explicar as mudanças sofridas. Além de convenções de nomenclatura, e um repositório central *CVS*. Foi identificado que a implantação do sistema traria significativas melhorias, como a compreensão do software, a análise de impacto das mudanças, e facilidade em se relacionar com sistemas legados. APIS seguiu então duas premissas funcionar como um grande banco de dados (*data warehouse*) aproveitando as técnicas já implantadas, todos os dados já gerados e mantidos pela empresa de modo a adequar a sua realidade e facilitar a integração. A segunda premissa baseou-se na utilização dos links de rastreabilidade através de um sistema de busca simples com suporte a query, cuja interface permite consultar informações de rastreamento e navegar entre os requisitos, documentação, tabelas de banco de dados, o código fonte, versão logs, e testes de unidade. Nas versões de Log é possível com a utilização de hyperlinks navegar até o código implementado. No suporte ao código fonte é possível verificar o código originado por um requisito. Apesar do avanço já alcançado com o uso da ferramenta, no período de publicação do estudo ela não estava finalizada deixando algumas lacunas em aberto. No entanto o autor expõe as lições aprendidas como o uso de ferramentas acompanhadas por processos apropriados, a introdução e evolução gradativa da cultura de rastreabilidade.

O uso de *plugins*, ferramentas integradas aos já consagrados ambientes de desenvolvimento tem se tornado mais comum, como é o caso das ferramentas utilizadas por Walters et al. (2014) para apresentar um algoritmo de geração automática de rastreabilidade. Este algoritmo que determina links entre uma requisição de mudança e entidades de código fonte, a partir de dados do movimento ocular. Sua abordagem consiste em gravar em vídeo, as sessões dos desenvolvedores tratando as solicitações e num segundo momento, os dados gerados são passados ao algoritmo apresentado resultando finalmente em uma entidade de ligação entre a solicitação de mudança e o que efetivamente foi alterado durante o período de resolução. O processo implica que a ferramenta de captura seja inicializada, durante o processo de resolução do caso, o movimento dos olhos do desenvolvedor e o tempo de permanência fixado numa dada posição X,Y da tela eram monitorados por uma ferramenta iTrace que gerava um arquivo condensando estes dados que eram arquivados. Este mapeamento era feito para todos os artefatos tratados pelo desenvolvedor conseguindo um nível de granularidade bem fino. Em seguida, o algoritmo

proposto acessava os dados arquivados e os processava comparando o código fonte depois da alteração com as posições no arquivo nas quais o desenvolvedor fixou mais o olhar. Segundo o autor, são necessárias melhorias na precisão mas os resultados são promissores. Para alcançar o objetivo do trabalho foi necessário o uso de um *software* especial e específico para a captura do movimento dos olhos. Além disso, houve a necessidade de várias sessões para a coleta das informações.

Outra ferramenta integrada ao ambiente Eclipse é apresentada por Motta (2013) que desenvolveu um *plugin* para visualizar o histórico de alterações em arquivos utilizando o sistema de controle de versões *Subversion*. Para alcançar seu objetivo fez uso de uma API específica para manipulação e comunicação com o repositório. Provê três tipos de visualização do histórico, *commit* geral, *commit* por desenvolvedor, e *commit* por período agrupado em meses, sendo os últimos dois informados em porcentagem. Uma limitação encontrada é a exibição das alterações considerando um único arquivo. Uma contribuição apontada pela autora foi a reunião num só documento dos principais conceitos envolvidos no desenvolvimento de *plugins*. Em suas considerações finais, Motta (2013) denota a importância da integração com sistemas de solicitação de mudanças, o que segundo ela, promoveria um controle melhor aos desenvolvedores sobre o que foi alterado no projeto e otimizaria o tempo gasto.

4 *GiveMe Views*

GiveMe Views (Tavares et al., em fase de elaboração) é uma ferramenta que tem como objetivo analisar os dados históricos de projetos de *software*, a fim de indicar os possíveis módulos que podem ser impactados quando uma manutenção corretiva, adaptativa ou evolutiva é realizada. Esta ferramenta também oferece visualizações que transmitem em detalhes, a relação entre os módulos. Sua história começa a partir de uma parceria estabelecida com uma empresa de desenvolvimento de *software* de gestão empresarial. Um processo de consultoria foi aplicado na mesma e um problema foi identificado. Este problema era a incapacidade de estimar de modo correto e com grau de precisão aceitáveis pelo gestor, quais pontos deveriam ser alterados quando uma dada solicitação de mudança era recebida pela equipe.

Essa empresa, chamada de parceira para manter sua identificação em sigilo, utilizava uma solução própria para a gestão de solicitações de mudança. A empresa possuía uma base de dados específica onde eram registradas solicitações de mudança, os componentes alterados (formulários e arquivos de código fonte, além de DLLs) e seus respectivos módulos (que são os produtos de *software* que a empresa comercializa), dentre outras informações. Para fins práticos esta base de dados foi referenciada como Repositório de Mudanças.

O processo de mudanças da empresa parceira consiste na abertura de uma solicitação de mudança, na sua execução e posterior verificação. A execução de uma solicitação de mudança implica na alteração de um ou mais componentes que, por sua vez, também poderá originar novas solicitações de mudanças em outros módulos e componentes. Nem sempre a empresa consegue identificar os módulos da aplicação que serão impactados (isto é, onde poderão ser necessárias novas alterações) para uma dada solicitação de mudança.

A etapa de verificação do processo de mudanças adotado pela empresa, é de responsabilidade do setor de qualidade. Ao receber uma solicitação de mudança para a correção de um defeito, por exemplo, deve-se, além de verificar se o defeito foi correta-

mente solucionado, verificar se novos defeitos não foram inseridos em outros componentes acoplados. Observou-se que grande parte das solicitações de mudança efetuadas quando eram enviados ao setor de qualidade para serem analisados, retornavam porque haviam pontos que não foram alterados quando o desenvolvedor resolveu a solicitação de mudança, ou seja, eles não estavam conseguindo estimar os locais que deveriam ser mantidos.

Foi identificado esse mesmo problema nas estimativas, em uma segunda empresa parceira e um centro de pesquisas. Assim, o trabalho foi iniciado, alicerçado na ideia de que, se existe um controle de dados históricos que mostrem o que foi alterado pela equipe ao longo do tempo, então a análise desses dados históricos podem ajudar a fornecer boas indicações de pontos que devam ser alterados. Ao perceber isso, surgiu um primeiro desafio: como extrair o histórico das bases de dados considerando que cada empresa pode ter seu tipo de base de dado diferente, ou seja, como se daria a extração de dados históricos de bases heterogêneas.

A partir disso, foi conduzido o desenvolvimento de um *framework* constituído por ferramentas capazes de manipular diferentes bases de dados históricos. O *GiveMe Metrics Framework* (Tavares et al., 2014), é um *framework* conceitual para extração de dados históricos sobre a evolução de *software*, capaz de extrair dados de três diferentes tipos de repositórios, que são: (i) repositórios de código fonte, (ii) repositórios de defeitos de *software* e, (iii) repositórios de processos de desenvolvimento de *software*. A Figura 4.1 mostra a arquitetura do *GiveMe Metrics Framework*.

A utilização do *framework* pode ser dividida em três diferentes cenários, os quais Tavares et al. (2014) explica em detalhes. O primeiro cenário objetiva analisar repositórios de código fonte, onde o usuário poderá escolher entre uma ferramenta manipulável via console (através de linha de comando) ou uma ferramenta manipulável via interface gráfica do usuário (GUI). Após a seleção da ferramenta, pode-se manipular a base de dados de código fonte, que significa executar a ferramenta sobre as versões de um projeto que estão sendo versionadas no repositório, e obter determinadas métricas sobre cada uma das versões. Ao final, obtém-se um conjunto de dados sobre o *software*, nesse caso, métricas de código fonte tais como número de classes, número de métodos por classe, número de subsistemas, número de linhas de código fonte, acoplamento entre objetos, falta de coesão entre

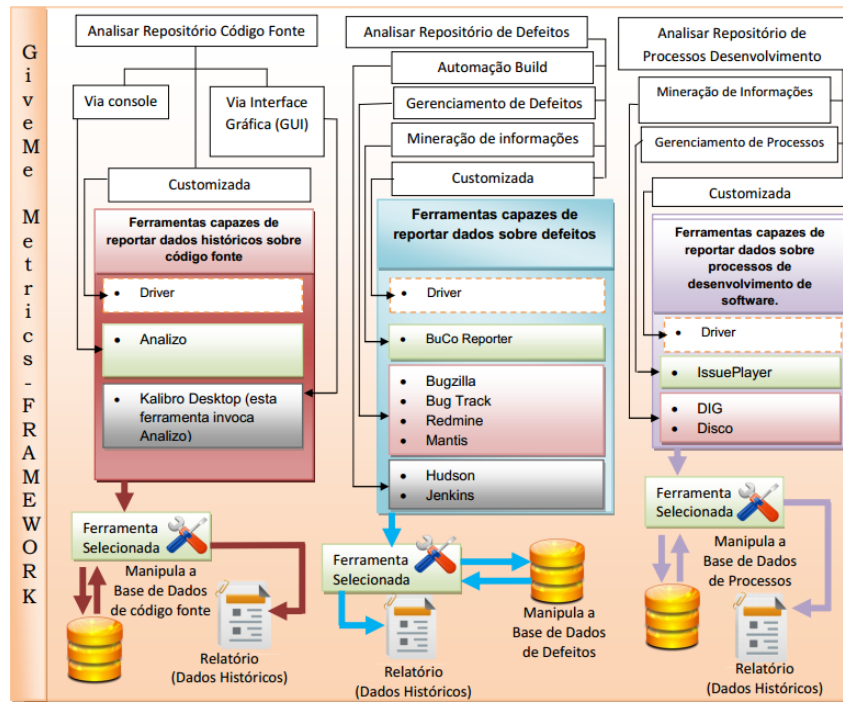


Figura 4.1: Arquitetura *GiveMe Metrics Framework* (Tavares et al., 2014)

métodos e complexidade ciclomática. O segundo cenário objetiva analisar repositórios de defeitos, a extração de dados sobre defeitos pode ser feita usando três diferentes grupos de ferramentas: (i) ferramentas de automação de *build*, (ii) ferramentas de gerenciamento de defeitos e (iii) ferramentas de mineração de informações em repositórios do tipo *Bugzilla*. Após a escolha da ferramenta, a base de dados de defeitos é minerada e dados extraídos dos defeitos são obtidos. Com as ferramentas de gerenciamento de defeitos é possível extrair dados como defeitos em aberto, defeitos resolvidos, autor dos defeitos, descrição do defeito, dentre outros. Com ferramentas de mineração de informações é possível extrair o número de defeitos não resolvidos, número de defeitos resolvidos, tempo médio de correção de um defeito, entre muitos outros. O terceiro cenário busca analisar repositórios de processos de desenvolvimento onde o usuário poderá escolher entre duas opções: (i) ferramentas de mineração de informações ou (ii) ferramentas de gerenciamento de processos. Na primeira opção, os dados possíveis de se obter são, entre outros: frequência absoluta, número máximo de repetições, duração total e a duração máxima. Já na segunda opção, é possível extrair dados como média de tempo dos artefatos no repositório, média de tempo dos defeitos cadastrados pertencentes a um projeto, média de tempo dos arquivos no repositório, média de tempo de construção das funcionalidades de um *software*.

O *framework*, então, foi aplicado nas empresas parceiras extraindo um conjunto de dados históricos. Após a extração dos dados históricos das empresas parceiras, era preciso analisar estes dados. Uma primeira ferramenta foi desenvolvida, em versão Desktop, recebendo o nome de “Analisador” cujo objetivo era, a partir da análise dos dados históricos, oferecer indicações com base em técnicas estatísticas dos pontos que poderiam ser alterados quando um outro ponto fosse alterado. A implementação de um mecanismo, que recebeu o nome de Matriz de Frequências, permitiu responder quais as chances de um dado componente, ao ser alterado, impactar outros módulos e componentes de um software, e até mesmo em outros softwares integrados, considerando o percentual histórico de modificações em que os componentes foram modificados juntos.

No entanto, viu-se a necessidade de melhorar as visualizações que essa ferramenta Analisador podia gerar. Assim, uma versão plugin da ferramenta foi desenvolvida e integrada à ferramenta *Sourceminor* (Carneiro, 2011) que é um *plugin* do ambiente Eclipse capaz de gerar visualizações através da análise de código fonte, desse modo a partir de um mesmo conjunto de dados era possível gerar múltiplas visões. Após a integração, o nome da ferramenta foi alterado de Analisador para *GiveMe Views*. Outras visualizações ao longo do tempo, também foram incorporadas à ferramenta *GiveMe Views* para potencializá-la.

A arquitetura básica da ferramenta *GiveMe Views*, é composta por seis módulos que podem ser descritos, resumidamente, da seguinte forma:

- Módulo *Import* é responsável por gerenciar toda a parte de importação dos dados extraídos do repositório de mudanças e torná-los disponíveis por estruturas de dados e objetos instanciados.
- Módulo *Data Filter* é responsável por conter a implementação dos filtros de dados que permitem selecionar solicitações de mudanças corretivas, adaptativas ou evolutivas.
- Módulo *Calculation of Metrics* é responsável por implementar as funcionalidades de cálculos de métricas.
- Módulo *Statistic Analysis*: módulo que contém a implementação das teorias es-

tatísticas que são a base das indicações feitas pela ferramenta.

- Módulo *Exportation* é o responsável por conter todas as funcionalidades de exportação de dados.
- Módulo *Graphical Visualization*: módulo responsável por implementar as visualizações disponíveis na ferramenta *GiveMe Views*.

Uma representação esquemática da arquitetura descrita para a ferramenta *GiveMe Views* pode ser vista na Figura 4.2.

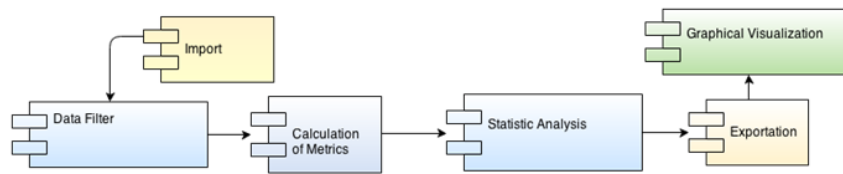


Figura 4.2: Arquitetura em módulos *GiveMe Views* (Tavares et al., em fase de elaboração)

O *GiveMe Views* entre outras funcionalidades, oferece um monitor que permite visualizar os módulos e os componentes que sofreram manutenções ao longo do tempo, com a respectiva quantidade de manutenções sofridas. Outra funcionalidade importante, é o conjunto de visualizações, dentre as existentes citam-se as seguintes: (i) *Visualização XLS* de Frequências, que provêm visualização tabular dos dados estatísticos calculados, (ii) *Graph Visualization*, que exibe um grafo que mostra as relações entre os módulos e componentes de um sistema, (iii) *TreeView Visualization*, visualização em forma de árvore, que permite visualizar componentes e módulos que poderão ser impactados quando um dado componente de um módulo for alterado, e (iv) *DeepView Visualization*, principal visualização, pois permite que o usuário selecione em tempo de execução diferentes componentes de um módulo e visualize os possíveis pontos que serão impactados, bem como as chances estatísticas de que isso ocorra.

A Figura 4.3 mostra um exemplo de visualização da RadialView com a análise estatística relacionando módulos e componentes.

Desde a alteração do nome para *GiveMe Views*, testes foram realizados por meio de provas de conceito e estudos experimentais, atestando sua viabilidade no apoio à avaliação de impacto da solicitação de mudanças (Tavares et al., em fase de elaboração). Na

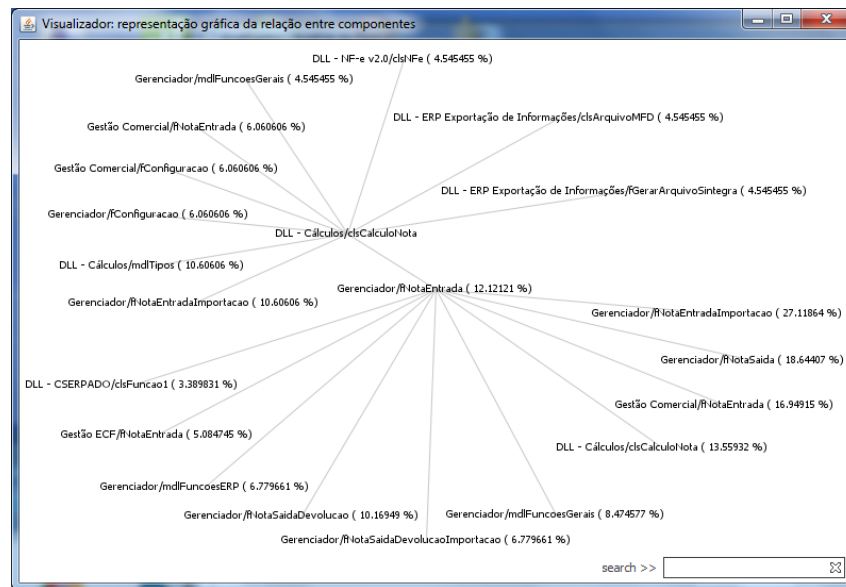


Figura 4.3: Visualização da *DeepView* no nível de componente (Tavares et al., em fase de elaboração)

prova de conceito realizada na empresa parceira, utilizando dados reais e o acompanhamento do gestor, foi possível coletar evidências de que a ferramenta possibilita apoiar a identificação de componentes que de fato foram modificados. Dos resultados qualitativos obtidos, destacam-se: o cálculo de métricas, a atualização da equipe de suporte, o apoio ao setor de qualidade, a análise histórica de dados automatizada, além da percepção positiva no uso da ferramenta e, notadamente, a mudança na forma de armazenar dados históricos, que antes eram armazenados apenas para conferência. Por causa disso, não havia informações com menor granularidade como, por exemplo, quais métodos ou funções de um componente sofreram manutenção.

Prevendo a possibilidade de analisar dados em níveis mais refinados, *GiveMe Views* foi projetada de modo a ser capaz de indicar os componentes de um módulo que poderão ser impactados em uma dada manutenção, e além disso havendo dados como nomes de métodos ou funções que foram modificados neste componente, *GiveMe Views* também indicaria os métodos ou funções possivelmente impactados.

Apesar de já estar apta a indicar possíveis métodos impactados, *GiveMe Views* não é capaz de gerar o dado na granularidade comentada. Para que essa indicação, então, de fato ocorra, a ferramenta dependeria de que um outro recurso computacional, seja uma outra ferramenta externa ou uma biblioteca gerasse e fornecesse o dado atendendo um nível de granularidade mais fina, ou seja, a nível de método ou função.

No capítulo seguinte, uma ferramenta é apresentada capaz de, entre outras funcionalidades, gerar dados de rastreabilidade no código fonte a nível de classe e método, a medida que o software evolui, integrada à *GiveMe Views*.

5 *GiveMe Trace*

GiveMe Trace é uma ferramenta que foi desenvolvida como um *plugin* do ambiente Eclipse capaz de gerar rastreabilidade entre os artefatos de código fonte e suas versões, evidenciando as alterações a nível de método, integrada à *GiveMe Views* que foi apresentada no capítulo anterior. Esta rastreabilidade é feita a partir da análise do repositório de um sistema de controle de versão, chamado aqui de modo simplificado de repositório de versões, de um determinado projeto. Possui suporte para trabalhar com dois dos gerenciadores de versionamento bastante usados hoje em dia, que são GIT (Chacon, 2009) e Subversion (Pilato et al., 2008). Foi projetada para ser utilizada de forma independente, gerando os dados de rastreabilidade a medida que o usuário demandar, ou integrada, estando incorporada a outras ferramentas provendo os dados num formato padrão de modo a facilitar a leitura destes dados e a integração por outros sistemas.

5.1 Arquitetura

Analizando a composição da ferramenta por pacotes e bibliotecas externas, suas funcionalidades, a relação entre eles e os diferentes modos de acesso aos métodos, foi possível definir claramente três camadas na estrutura da ferramenta com responsabilidades e características próprias. Essa divisão fornece uma visão que facilita a compreensão de como a ferramenta funciona e de que modo a questão da integração foi tratada no projeto da ferramenta.

As três camadas definidas foram: (i) camada de Interface, (ii) camada de Integração e (iii) camada de Sistema. A primeira é a camada de Interface, trata a parte de interface gráfica, visual, pela qual o usuário interage e se comunica com o sistema. Esta camada compreende também as ferramentas externas que desejem se conectar ao *GiveMe Trace* e utilizar seus dados gerados. A segunda é a camada de Integração, ela é responsável pela comunicação entre a camada de Interface e a camada de Sistema. Ao proporcionar um isolamento dos processamentos de sistema também é responsável por

facilitar a integração com outros sistemas. A partir desta camada é que a comunicação com a outra ferramenta ocorrerá. A terceira camada é a camada de Sistema, ela é responsável por efetuar operações e gerenciar recursos de mais baixo nível como conexão e processamento dos bancos de dados de versão. A Figura 5.1 mostra uma representação dos pacotes pertencentes a cada camada, e através das setas, o fluxo de dados.

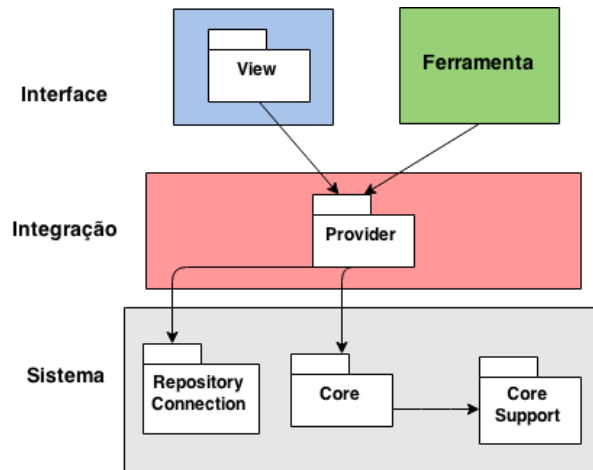


Figura 5.1: Representação da Arquitetura - Pacotes por Camadas

A descrição de cada pacote é feita a seguir:

- *View*: inicializa todas as variáveis de ambiente para o funcionamento do plugin de forma independente e possui ainda a interface gráfica por onde o usuário pode interagir
- *Provider*: responsável por prover um meio de comunicação entre as camadas de Interface e de Sistema. De uma maneira padronizada e clara, ele encapsula as chamadas aos métodos e processamentos complexos da camada inferior. No caso em que o GiveMe Trace é incorporado a outras ferramentas, este pacote faz a comunicação entre a ferramenta e a camada de sistema
- *Repository Connection*: Responsavel por conter as funcionalidades para conexão com os diferentes repositórios suportados. Neste pacote são implementados, o objeto de conexão e o algoritmo de extração dos dados de conexão.
- *Core Support*: responsável por dar o suporte necessário ao funcionamento do pacote Core, implementando métodos utilitários para manipulação de arquivos de texto e

validação de dados, além de conter a definição dos objetos requeridos na execução do pacote Core.

- *Core*: responsável por conter todas as funcionalidades de extração e geração dos logs de histórico das versões, a nível de classe ou a nível de método.

Algumas funcionalidades são acessíveis às ferramentas externas e à camada de Interface, somente via *Provider*. Entre elas estão: (i) gerar arquivo de *log* do histórico de versões evidenciando as modificações a nível de método, (ii) gerar arquivo de *log* do histórico de versões evidenciando as modificações a nível de classe, (iii) listar classes alteradas precedido pelo número da versão correspondente, (iv) listar métodos alterados também precedido pelo número da versão correspondente e (v) listar métodos alterados no *workcopy* em comparação ao *head* do repositório, que representa o último *commit* realizado. Assim, nestas funcionalidades, destaca-se a relação entre a versão do código fonte e o método, ou classe, alterada dependendo de qual funcionalidade será utilizada.

Com relação ao pacote *Core*, por ser o mais complexo e ter relação direta com bibliotecas externas, será detalhado nos próximos parágrafos. É composto por duas classes, cada uma representando um sistema de controle de versão que o *GiveMe Trace* suporta, GIT ou Subversion. A Figura 5.2 mostra uma visão ampliada do pacote *Core* com as relações internas entre as classes e as bibliotecas externas SVNkit (SVNkit, 2014), jGIT (jGIT, 2014) e JavaParser (JavaParser, 2014).

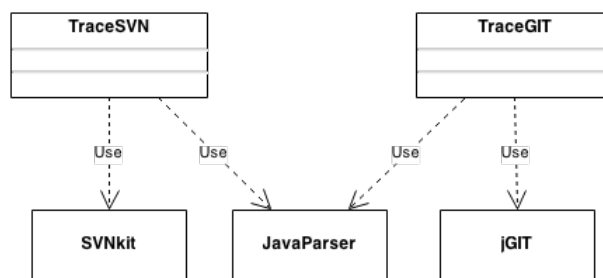


Figura 5.2: Pacote Core em detalhe

Dentre os artefatos destacados na Figura 5.2, está o JavaParser, trata-se de uma biblioteca Java, capaz de analisar qualquer trecho de código Java e permite acesso aos dados resultantes dessa análise como, por exemplo, nome de método, nome de classe, comentários, javadoc, nome do pacote, importações, entre outras. Dentre os dados de

análise, três informações são fundamentais para o algoritmo desenvolvido, são elas: (i) nome do método, (ii) número da primeira linha do método e (iii) número da linha final do método.

Outros dois artefatos apresentados pelo diagrama da Figura 5.2 são: SVNkit e jGIT. Tratam-se de bibliotecas Java que implementam mecanismos de manipulação de repositórios Subversion e GIT, respectivamente. Segundo a documentação de ambos (SVNkit, 2014)(jGIT, 2014), esta é uma definição muito simplista mas atende o contexto desta explicação. As bibliotecas provêm conexão aos repositórios além de atividades típicas de sistemas de controle de versão, tais como: realizar *check-out*, *commit* e exibir histórico das versões, entre outras.

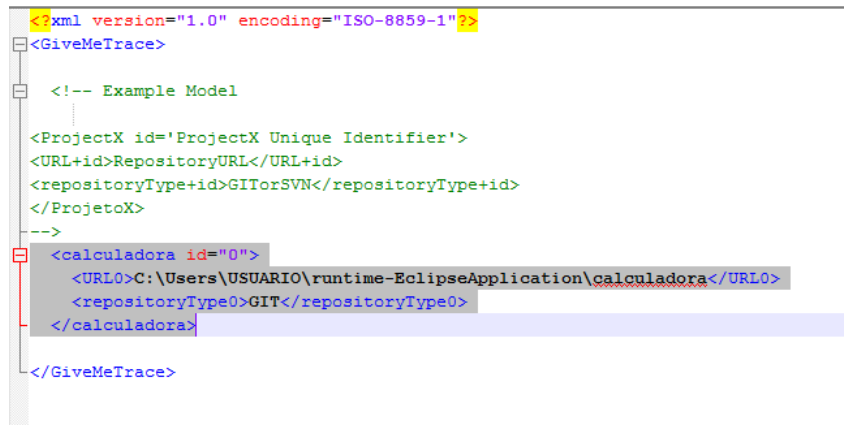
Finalmente, as classes *TraceSVN* e *TraceGIT* mostradas pelo diagrama, são as responsáveis por prover internamente à ferramenta *GiveMe Trace*, os métodos de geração de rastreabilidade já comentados que são chamados via *Provider*. Cada classe chama a respectiva biblioteca de acesso ao repositório de versão e o *JavaParser* onde for necessário.

Tanto a classe *TraceSVN* como a *TraceGIT* possuem um algoritmo base para gerar a lista de rastreabilidades entre versões e métodos alterados. Este algoritmo base se inicia recebendo como parâmetro, um artefato a ser analisado, *classeX.java* e a versão X correspondente. Em seguida, utiliza-se as bibliotecas externas SVNkit e jGIT dependendo do caso, para recuperar o mesmo artefato porém na versão X-1 e a diferença entre as duas versões. Neste momento, o *JavaParser* analisa o artefato na versão X e retorna: (i) nomes dos métodos e (ii) intervalo de linhas que cada método ocupa na classe. Com a informação da diferença entre as versões, foi possível, processar e extrair uma lista L de linhas alteradas. A partir da lista L, e do intervalo de linhas de cada método, foi possível identificar se as linhas alteradas pertencem a algum intervalo que representam um método, refletindo uma alteração no método. Desta forma é possível apontar métodos que foram criados, alterados ou possivelmente deletados.

5.2 Modo *Standalone*

Como já foi dito anteriormente, o *GiveMe Trace* pode funcionar como um *plugin* independente gerando *logs* para o próprio usuário sempre que ele solicitar por meio de uma

interface gráfica disponibilizada para essa tarefa. Mas para que esta geração dos *logs* se conclua depois de chamada via interface, é preciso preencher o documento de configuração *GiveMeTrace.xml*, em formato *XML*, com os dados de conexão ao repositório de versões, pois é a partir deste documento que o pacote *Repository Connection*, vai extrair as informações necessárias para o acesso ao repositório. O arquivo de configuração possui um padrão de preenchimento tal qual mostrado na Figura 5.3.



```
<?xml version="1.0" encoding="ISO-8859-1"?>
<GiveMeTrace>
  <!-- Example Model
  ...
  <ProjectX id='ProjectX Unique Identifier'>
    <URL+id>RepositoryURL</URL+id>
    <repositoryType+id>GITorSVN</repositoryType+id>
  </ProjectX>
  <-->
  <calculadora id="0">
    <URL0>C:\Users\USUARIO\runtime-EclipseApplication\calculadora</URL0>
    <repositoryType0>GIT</repositoryType0>
  </calculadora>
</GiveMeTrace>
```

Figura 5.3: Padrão dos dados de conexão a um repositório

Inicialmente o cabeçalho necessário para todo arquivo no formato *XML* (*eXtensible Markup Language*). Em seguida, a identificação de abertura e fechamento do arquivo com a tag *GiveMeTrace*. Na figura 5.3, acima do trecho destacado, está um modelo de preenchimento dos dados de um projetoX e destacado em cinza o preenchimento dos dados de um projeto cujo nome é calculadora, versionado por *GIT*. Ao cadastrar um projeto a ser analisado pela ferramenta *GiveMe Trace*, a primeira tag recebe o nome do projeto seguido do atributo *id* que é um identificador numérico sequencial para cada projeto configurado no documento. Aninhada à tag do projeto é preciso especificar duas tags cujos nomes são *URL+id* e *repositoryType+id*. A expressão *+id* indica que os nomes das tags recebem o valor do identificador do projeto, por exemplo *URL0*, tal qual é destacado em cinza na figura. Deve-se então, especificar na tag *URL*, o endereço para o repositório e na tag *repositoryType*, qual o tipo de repositório utilizado, *GIT* ou *Subversion*, indicado por *SVN*. Vale lembrar que o arquivo deve ser salvo na pasta *C:/GiveMeRepository/Brain/* com o nome *givemetrace.xml*.

Um fluxo básico de atividades necessárias à utilização do *pluginGiveMe Trace* de modo *Standalone*, é descrito a seguir.

Ao utilizar o *plugin* independentemente, o usuário inicialmente seleciona o projeto a ser analisado, que deve estar presente no *workspace* do Eclipse. Depois informa, qual o número do *commit* que ele deseja analisar e, em seguida, se é uma análise no nível de classe ou no nível de método.

Com relação ao número do *commit* a ser analisado, usando notações específicas, o *GiveMe Trace* é capaz de analisar tanto um único *commit*, como um intervalo de *commits* ou, ainda, uma combinação dos dois. *Commits* únicos são separados por ponto e vírgula e intervalos de *commits* são separados por um traço, símbolo de subtração. A notação “0” (zero) é utilizada para indicar o último *commit* feito, sendo chamado de *head* do repositório. Por exemplo, supondo que se deseje analisar os *commits* do primeiro ao terceiro, o sexto *commit* e do décimo ao último *commit* feito no repositório, a notação ficaria: 1-3;6;10-0.

Por outro lado, a escolha do nível da análise não é feita de modo manual, ela é interpretada ao clicar no botão *Start* correspondente ao nível desejado. A Figura 5.4 mostra a interface gráfica do *plugin* preenchida com um exemplo de análise prestes a ser feita.

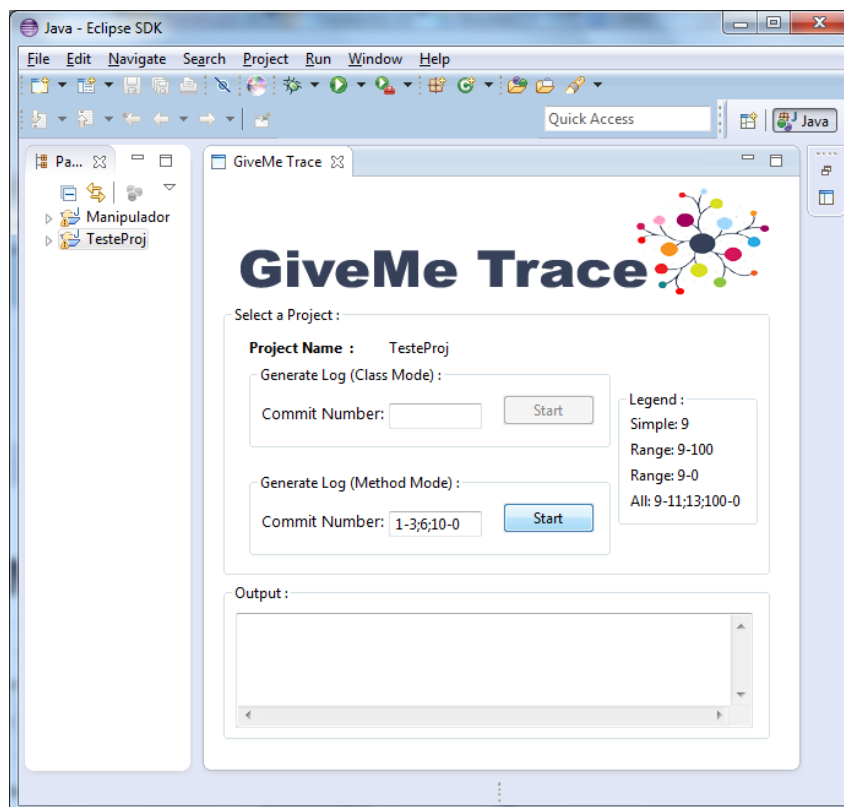


Figura 5.4: GiveMe Trace: tela inicial

Levando em consideração o exemplo retratado na Figura 5.4, após clicar no botão *Start* o processamento é iniciado e uma instância da classe *MasterProvider* é criada. A partir disso, a camada de Sistema entra em operação, extraíndo do arquivo *XML* já configurado, os dados de conexão ao repositório de versões para o projeto “TesteProj” selecionado inicialmente. A sequência a partir deste ponto do processo é determinada pelo tipo de sistema que controla as versões de “TesteProj”, *GIT* ou *Subversion*. Se “TesteProj” for versionado por um sistema *GIT*, então a classe *TraceGIT* é chamada, caso seja versionado por *Subversion*, então *TraceSVN* é chamado. O passo seguinte leva em consideração o nível de análise escolhido, pelo exemplo da Figura 5.4, o “número de *commit*” foi especificado para o modo de método, assim o resultado levará em conta os métodos alterados entre as versões. Ao término do processamento, o usuário é notificado por uma caixa de dialogo (Figura 5.5) sobre o sucesso da operação e a localização para o arquivo resultante. Esse arquivo pode mostrar dados diferentes dependendo do nível de análise escolhido pelo usuário, os dados comuns tanto ao nível de classe quanto ao nível de método são: (i) o número do *commit*, (ii) data e hora da realização do *commit*, (iii) autor do *commit* e (iv) mensagem atribuída ao *commit*. Caso tenha sido escolhida uma análise no nível de classe, logo após a mensagem é exibida uma listagem com as classes alteradas naquele *commit*. Esta listagem divide as classes por linha e apresenta a informação na forma: M, indicando a ocorrência de uma modificação na classe, seguida pela sequência de diretórios do projeto que identificam unicamente a referida classe, e por fim o nome da classe propriamente. No exemplo da figura 5.4, foi considerado o nível de método, portanto, a listagem a ser exibida se refere aos métodos alterados, assim a informação é apresentada na forma: M, indicando método alterado, a identificação da classe a qual o método alterado pertende, seguido por um sinal de *pipe*, e por fim, o nome específico do método alterado. O arquivo resultante apresenta esses dados em sequência, na ordem cronológica de realização dos *commits*, formando um *log* do histórico destes *commits*.

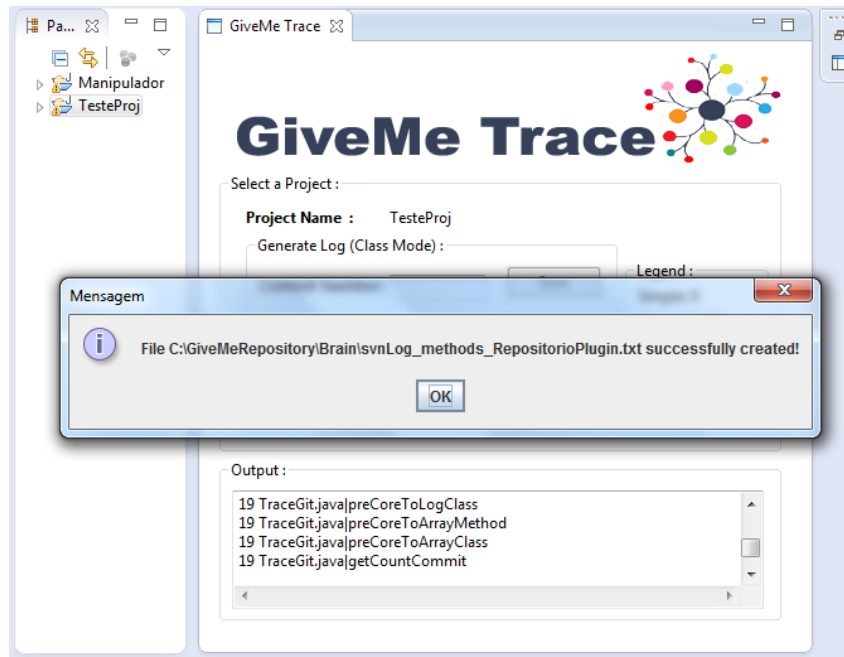


Figura 5.5: Aviso de sucesso ao finalizar a geração da rastreabilidade

A Figura 5.6 representa um trecho do arquivo gerado pelo exemplo de execução descrito.

```
Commit: 19
Autor: USUARIO
Date: Tue Nov 11 13:35:37 BRST 2014

Mensagem: final da implementação dos Ranges;
Resolvi warnings;

Fechamento do projeto;

M /givemetrace/src/givemetrace/implementations/TraceGit.java|preCoreToLogMethod
M /givemetrace/src/givemetrace/implementations/TraceGit.java|preCoreToLogClass
M /givemetrace/src/givemetrace/implementations/TraceGit.java|preCoreToArrayMethod
M /givemetrace/src/givemetrace/implementations/TraceGit.java|preCoreToArrayClass
M /givemetrace/src/givemetrace/implementations/TraceGit.java|getCountCommit

Commit: 20
Autor: USUARIO
Date: Tue Nov 18 18:25:43 BRST 2014

Mensagem: tratamento dos url nos métodos do provider para evitar exception por
não reconhecer que o url é um repositório

M /givemetrace/src/givemetrace/views/vpGiveMeTraceMain.java|createPartControl
```

Figura 5.6: Trecho do arquivo de rastreabilidade gerado a nível de método

Com o resultado neste formato, fica em destaque a rastreabilidade entre uma versão do *software* e os métodos que foram alterados para formar aquela versão, e isso para todas as versões informadas pelo usuário. Os passos subsequentes, não contam com a participação do usuário, sendo desempenhados pela camada de sistema, mediante a requisição especificada num método da classe *MasterProvider*. Sendo executado independentemente ou incorporado a outra ferramenta, esses passos subsequentes feitos pela

camada de sistema são os mesmos. Assim, o comportamento esperado pela ferramenta quando integrada não deve mudar, mantendo o padrão na chamada e no resultado obtido.

5.3 Modo Integrado

A ferramenta *GiveMe Trace* pode ser incorporada a outras ferramentas, tal qual ocorreu com as ferramentas *Mantis* e *GiveMe Views*. Nesse cenário, a ferramenta integrada, por exemplo o *GiveMe Views*, passa a ser responsável pela interação com a camada de Integração, e esta, proverá os métodos, funções e variáveis necessários à obtenção dos recursos oferecidos pela camada de Sistema.

O principal objetivo em facilitar a integração, encapsulando o processamento feito pela camada de Sistema, padronizando a geração e exibição dos resultados, é possibilitar que outros desenvolvedores possam alimentar suas ferramentas visuais com os dados gerados pelo *GiveMe Trace*.

Nas sessões seguintes é mostrada uma integração da ferramenta *GiveMe Trace* com a ferramenta de solicitação e acompanhamento de mudanças *Mantis Bug Tracker*, e ainda, a integração feita com a ferramenta já apresentada *GiveMe Views*, evidenciando a posição do *GiveMe Trace* na arquitetura geral do *GiveMe Views*, e a contribuição do presente trabalho que é fazer as indicações do *GiveMe Views* aparecerem no nível de método alterado.

5.3.1 Integração com *Mantis Bug Tracker* (*MantisBT*)

Mantis Bug Tracker é um gerenciador de solicitação de mudanças para web, que permite acompanhar o chamado caso, ou ticket, desde sua criação até sua resolução completa ou sua rejeição. Apesar de ser originalmente construído para a plataforma *web*, existe uma ferramenta *plugin* para o Eclipse que trabalha como um conector ao *MantisBT* e possibilita a gerência de todos os *tickets* dos projetos gerenciados através do ambiente Eclipse auxiliando o trabalho do desenvolvedor ao tratar a mudança solicitada. Esse *plugin* é o *Mylyn-Mantis*, disponível *open-source* (Mylyn-Mantis, 2014). Foi com ele que ocorreu a integração do *GiveMe Trace*.

Os gerenciadores de solicitação de mudança de modo geral, incluindo *MantisBT*, permitem o cadastro de uma solicitação de mudança, montando um guia para o desenvolvedor do que deveria ser alterado. Assim, um fluxo básico da resolução dos tickets poderia ser descrito da seguinte forma: o desenvolvedor realiza a alteração no código fonte e faz o *commit* das alterações no repositório de código fonte, junto dos artefatos submetidos a controle de versão, que geralmente é diferente do banco de dados onde ficam armazenadas as solicitações de mudança ao longo do tempo. A forma tradicional de gerenciar das ferramentas conhecidas de acompanhamento de mudanças não permitiam que se fizesse um relacionamento direto, de forma automática, entre o caso solucionado e o que foi efetivamente alterado. Nesse sentido, foi desenvolvida uma adaptação ao *plugin Mylyn-Mantis*, por meio da integração com o *GiveMe Trace*, de modo a permitir que quando um caso for fechado, seja possível associar a esse caso, a partir da informação do *commit*, as alterações efetivamente realizadas no código fonte.

Para que a integração entre o *plugin* do *Mantis* com o *GiveMe Trace* pudesse ocorrer com êxito, foi preciso efetuar alterações, acrescentando algumas informações à versão web da ferramenta. Foram acrescentados dois campos personalizados, o primeiro deles para que o desenvolvedor pudesse informar um número de *commit*, e o segundo para que fosse exibida a lista dos métodos alterados no *commit* informado. Um manual detalhado explicando como adaptar o servidor *MantisBT* de modo a integrar com o *GiveMe Trace*, foi feito e pode ser acessado por meio eletrônico (Lelis, 2014). Também foi preciso fazer alterações à versão *plugin*. Na verdade, foi na versão *plugin* que a integração de fato ocorreu e os métodos da classe *MasterProvider* puderam ser invocados.

A adaptação implementada consistiu em identificar o ponto no código que era feita a submissão do formulário de solicitação ao banco de dados, mas somente quando uma alteração ao formulário fosse constatada. Em seguida, identificar o parâmetro que representa o campo do formulário que o desenvolvedor informou o *commit*. Com esse dado em memória, e o nome do projeto para o qual a solicitação foi direcionada, a classe *MasterProvider* do *GiveMe Trace* pode ser chamada. Obtendo como resultado do *MasterProvider* a lista de métodos alterados, foi preciso atribuir este resultado ao parâmetro que representa o campo “métodos modificados” do formulário onde se queria mostrar os

resultados obtidos.

Da forma como foi realizada a integração, basta que o desenvolvedor indique qual *commit* foi necessário para a resolução do caso, que o *GiveMe Trace* gera automaticamente a rastreabilidade entre o caso, ou ticket, e os métodos alterados por causa daquela solicitação de mudança. Os modos de informar o *commit* seguem o padrão já apresentado anteriormente, onde o traço indica intervalo entre *commits* e ponto e vírgula separação de *commits* independentes. Assim, permite que o desenvolvedor atualize o caso e a cada atualização vá acrescentando os intervalos e números de todos os *commits* que forem necessários para a resolução completa do caso.

A Figura 5.7 mostra a tela do *plugin Mylyn-Mantis* já integrado ao *GiveMe Trace* destacando, o resultado dos métodos alterados depois de uma atualização no formulário com o fornecimento de um número de *commit*.

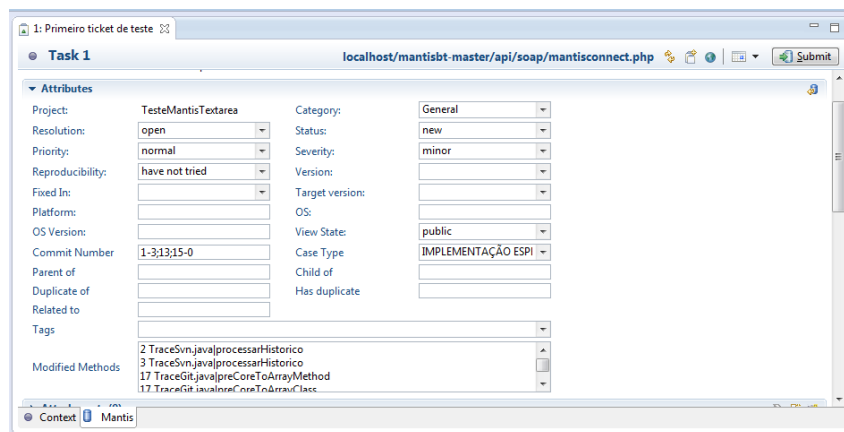


Figura 5.7: *Mylyn-Mantis* integrado com *GiveMe Trace*

5.3.2 Integração com *GiveMe Views*

A ferramenta *GiveMe Views* não conseguia fazer indicações de possíveis métodos impactados, considerando que outro método sofreu uma mudança. Por outro lado, a ferramenta que foi desenvolvida *GiveMe Trace*, possui funcionalidades que geram os dados de alteração num grau de granularidade mais fino, a nível de método. Além disso, possui uma estrutura que facilita a integração com outros sistemas. Aproveitando a integração já feita com a ferramenta *Mantis*, e a possibilidade dada pela *GiveMe Trace* de solucionar a limitação comentada do *GiveMe Views*, foi proposta então uma integração entre

GiveMe Trace e *GiveMe Views*. Como o *Mantis* já estava operando com o *GiveMe Trace* incorporado a ele, integrou-se também o *Mantis* à ferramenta *GiveMe Views*.

O processo de integração se deu de modo semelhante ao ocorrido com o *Mantis*. O *GiveMe Trace* foi incorporado como um componente adicional junto ao *GiveMe Views* e através de um objeto instanciado da classe *MasterProvider*, suas funcionalidades eram acessadas.

Após a integração das ferramentas, a arquitetura geral do *GiveMe Views*, sofreu uma alteração devido a contribuição oferecida pelo *GiveMe Trace* e o *Mantis* já adaptado. A Figura 5.8 mostra a arquitetura do *GiveMe Views* atualmente após a integração das ferramentas. A ligação com o *GiveMe Views* permite a todas as visualizações acessarem o *GiveMe Trace* e utilizarem as funcionalidades oferecidas por ele.

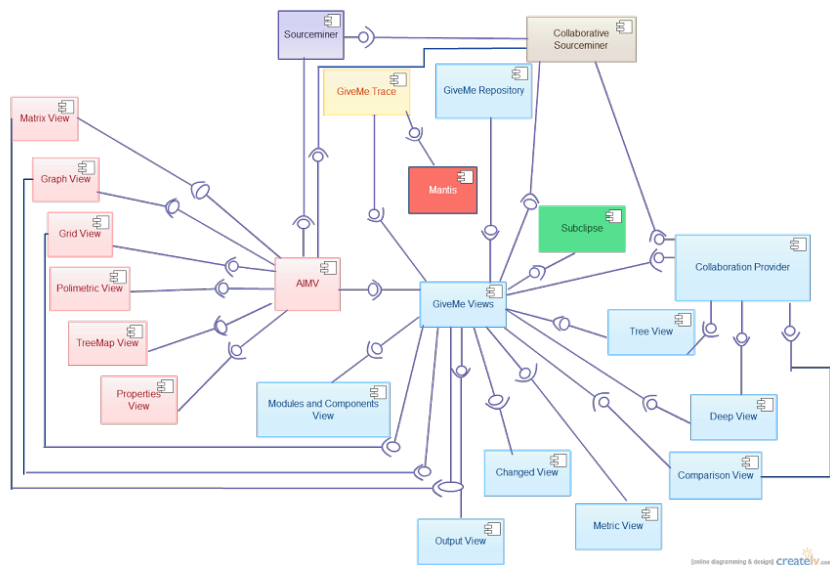
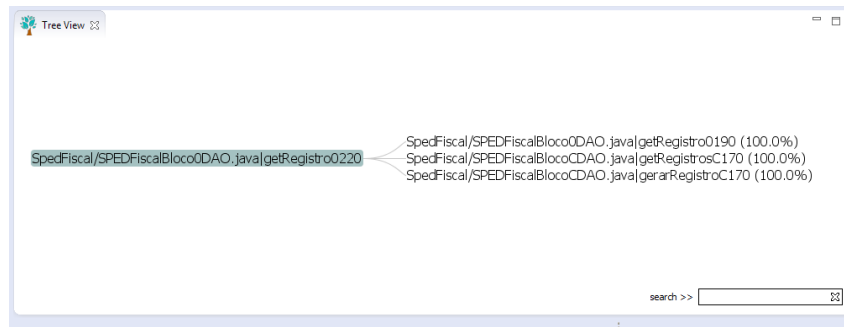


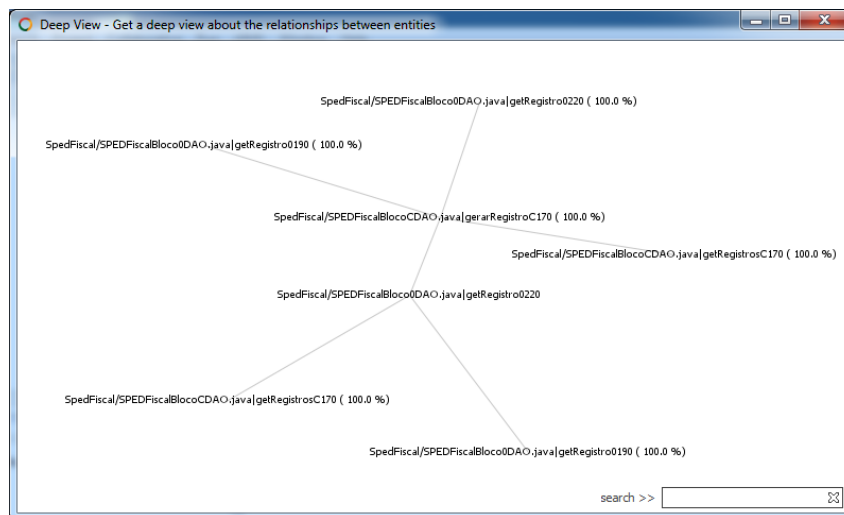
Figura 5.8: *GiveMe Trace* integrado ao *GiveMe Views*

Para demonstrar algumas perspectivas diferenciadas que foram permitidas a partir do *GiveMe Trace*, as Figuras 5.9 e 5.10, mostram visualizações providas pelo *GiveMe Views* porém agora contemplando a contribuição fornecida pelo *GiveMe Trace*.

A visualização *TreeView* (Figura 5.9) demonstrava quais componentes poderiam ser impactados quando uma dada manutenção fosse realizada num outro componente especificado para a análise. Além dos componentes, era informado a porcentagem de chances do impacto ocorrer. Agora, a análise é feita num nível mais baixo, especificando os métodos a serem impactados e suas respectivas chances de ocorrência.

Figura 5.9: Visualização da *TreeView* a nível de método

A visualização *Deep View* (Figura 5.10), permite que o usuário selecione em tempo de execução diferentes componentes de um módulo e visualize os possíveis pontos que serão impactados, bem como as chances estatísticas de que isso ocorra. A representação após a integração com *GiveMe Trace*, permite que o usuário selecione um método e avalie o impacto também em outros métodos.

Figura 5.10: Visualização da *DeepView* a nível de método

6 Prova de Conceito

Esta seção descreve uma prova de conceito com o objetivo de analisar a viabilidade da ferramenta *GiveMe Trace* para detectar as classes e os métodos alterados entre versões do código fonte, estando a versão no repositório ou ainda na cópia de trabalho (*workcopy*) do desenvolvedor. As funcionalidades da ferramenta foram divididas em quatro testes através dos quais sua viabilidade deveria ser analisada. São eles:

- Teste 1: Geração da lista de métodos que foram alterados num dado *commit*;
- Teste 2: Geração da lista de métodos que foram alterados no *workcopy*;
- Teste 3: Geração dos arquivos de log com as alterações de todas as versões: (i) no nível de classe e (ii) no nível de método;
- Teste 4: Recuperar via *Mylyn-Mantis*, a lista de métodos alterados tendo como origem uma solicitação de mudança.

A prova de conceito foi executada em dois cenários reais, o primeiro utilizando um projeto chamado *SpedFiscal*, desenvolvido por uma empresa parceira, situada em Juiz de Fora que desenvolve sistemas para gestão empresarial, com o qual foram realizados os testes 1 e 2. A execução destes testes ocorreu no ambiente da empresa parceira, e a apresentação dos resultados destes testes, foi feita utilizando visualizações providas pela ferramenta *GiveMe Views*, após concluído o processo de integração com o *GiveMe Trace*. O segundo cenário utilizou o projeto de uma calculadora (Calculadora, 2014) versionado em *GIT*, com o qual foram realizados os testes 3 e 4.

As sessões seguintes apresentam a execução de cada teste em detalhes.

6.1 Teste 1

Este teste foi desenvolvido com o objetivo de analisar a ferramenta *GiveMe Trace* na geração da lista de métodos que foram alterados num dado *commit*. Lembrando que este

teste foi executado sobre o projeto da empresa parceira, um desenvolvedor que conhece o projeto e também conhece o repositório, acompanhou a execução deste teste. O *commit* selecionado foi o de número 2 onde o arquivo *SpedFiscalBloco0DAO* sofreu uma alteração na linha 774 pertencente ao método *getRegistro0460*. Segundo o desenvolvedor, esta foi a única alteração referente a versão escolhida. A Figura 6.1 que destaca a linha antes e depois da alteração, foi gerada a partir do *software TortoiseSVN* (TortoiseSVN, 2014), utilizado para gerenciar versões do sistema *Subversion*.

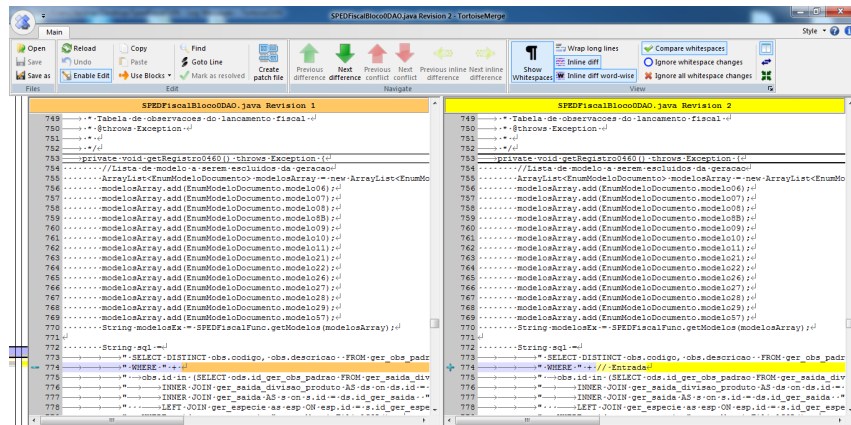


Figura 6.1: Alteração na versão selecionada

Nesse contexto, foi definida a hipótese de que a ferramenta *GiveMe Trace* é capaz de gerar a lista de métodos que foram alterados na versão 2 do projeto.

A Figura 6.2 mostra a visualização *Comparison View* gerada a partir da execução da ferramenta *GiveMe Views* integrada ao *GiveMe Trace*, considerando as indicações feitas antes da alteração em comparação ao que de fato foi alterado no *commit* 2, selecionado para a análise. Na figura é possível perceber que a lista gerada pela ferramenta *GiveMe Trace* foi passada para a tabela *Modifications* e mostra que o método *getRegistro0460* foi alterado.

Assim, este teste descrito e seus resultados, apresentaram indícios de que a ferramenta é capaz de gerar a lista de métodos que foram alterados na versão do projeto selecionado pelo desenvolvedor da empresa parceira.

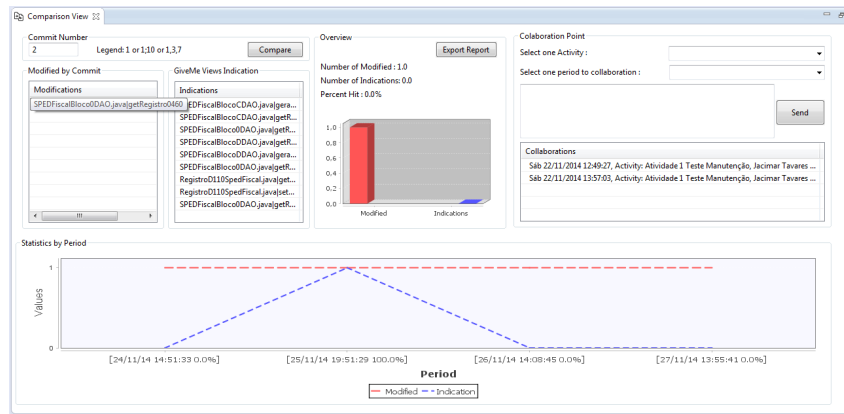


Figura 6.2: Visualização *Comparison View* da alteração na versão selecionada

6.2 Teste 2

Este outro teste foi desenvolvido com o objetivo de analisar a ferramenta *GiveMe Trace* na geração da lista de métodos que foram alterados mas ainda não geraram uma nova versão no repositório, ou seja, ainda estão no *workcopy* do desenvolvedor. Lembrando que este teste também foi executado sobre o projeto *SpedFiscal* da empresa parceira, e novamente o desenvolvedor que conhece o projeto e também conhece o repositório acompanhou a execução.

Foram conduzidas mudanças nos métodos *gerarRegistro0000*, *getRegistro0190*, *getRegistro0200*, *getRegistro0220*, *getRegistro0305*, *getRegistro0460*, todos pertencentes a classe *SpedFiscalBloco0DAO*. Como não julgou-se necessárias outras modificações, uma hipótese foi definida de que a ferramenta *GiveMe Trace* é capaz de gerar a lista de métodos alterados presentes ainda no *workcopy*.

Foi solicitado ao *GiveMe Trace* que gerasse a lista de métodos alterados no *work-copy*, cujo resultado pode ser visto na Figura 6.3. A figura mostra a visualização *ChangedView* gerada a partir da execução da ferramenta *GiveMe Views* integrada ao *GiveMe Trace*. Esta visualização é capaz de mostrar o que foi indicado pelo *GiveMe Views*, o que o desenvolvedor já realizou de alteração na cópia de trabalho e os métodos que ele precisa ainda verificar.

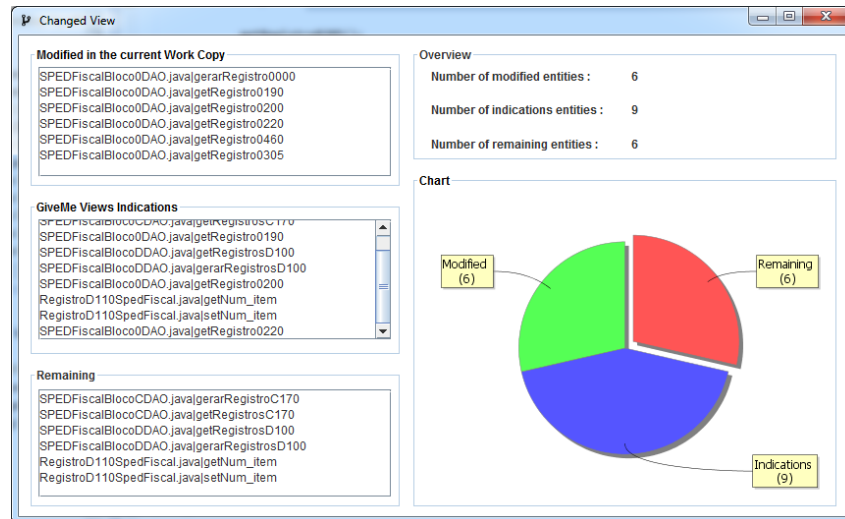


Figura 6.3: Visualização *ChangedView* da alteração no *workcopy*

O desenvolvedor pôde, então, proceder com o *check-in* das alterações no repositório, como pode ser visto na Figura 6.4. A figura mostra a tela do *software TortoiseSVN* no momento de fazer *commit* no repositório, evidenciando que as alterações foram de fato feitas apenas na classe *SpedFiscalBloco0DAO*, conforme dito anteriormente.

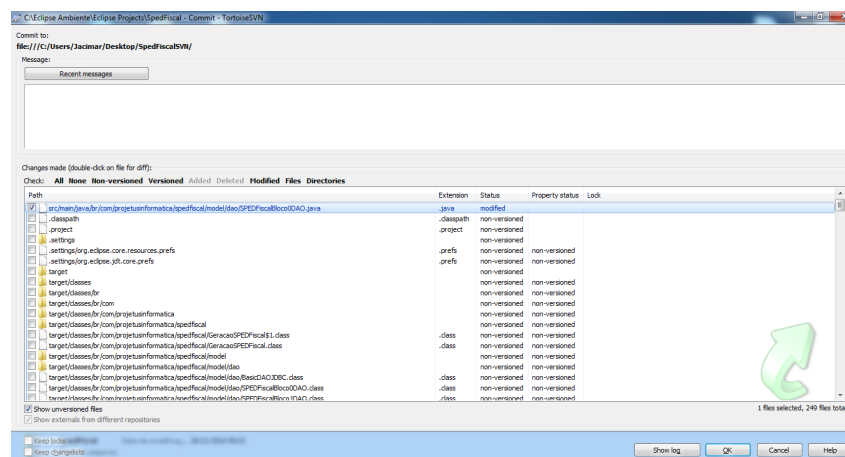


Figura 6.4: Classe alterada no *workcopy*

Caso seja dado dois cliques na classe modificada, uma tela de diferenças (*Diff*) é exibida e as alterações poderão ser vistas nos métodos que a *ChangedView* acusou como alterados. Assim, este segundo teste descrito e seus resultados, apresentaram indícios de que a ferramenta é capaz de gerar a lista de métodos alterados presentes ainda no *workcopy* do desenvolvedor da empresa parceira.

6.3 Teste 3

Este terceiro teste foi desenvolvido com o objetivo de analisar a ferramenta *GiveMe Trace* na geração dos arquivos de *log* completo das versões, nos dois níveis (i) de classe e (ii) de método. Para cada nível foi definida uma hipótese diferente, (i) a ferramenta *GiveMe Trace* é capaz de gerar o arquivo de *log* de todas as versões no nível de classe, e (ii) a ferramenta *GiveMe Trace* é capaz de gerar o arquivo de *log* de todas as versões no nível de método.

Para a execução deste teste foi utilizado o projeto Calculadora, cujas versões são controladas via *GIT*. Foi solicitado ao *GiveMe Trace* que gerasse o arquivo de *log* de todas as versões do projeto Calculadora primeiramente no nível de classe, cujo resultado pode ser visto na Figura 6.5. A figura mostra ainda no lado esquerdo um histórico recuperado pelo *software GIT Bash* (GIT Bash, 2014), uma ferramenta nativa do *Git*, e que portanto apresenta resultados ideais para a comparação. Apesar da ferramenta indicar pela ordem cronológica inversa das versões, é possível verificar a correspondência dos *commits* com o que foi alterado, por exemplo, o *commit* em destaque no arquivo gerado, à esquerda, identificou duas classes Java sendo elas *Basico* e *Visor*. No histórico, à direita, o mesmo *commit* aponta os dois arquivos aos quais as classes pertencem.

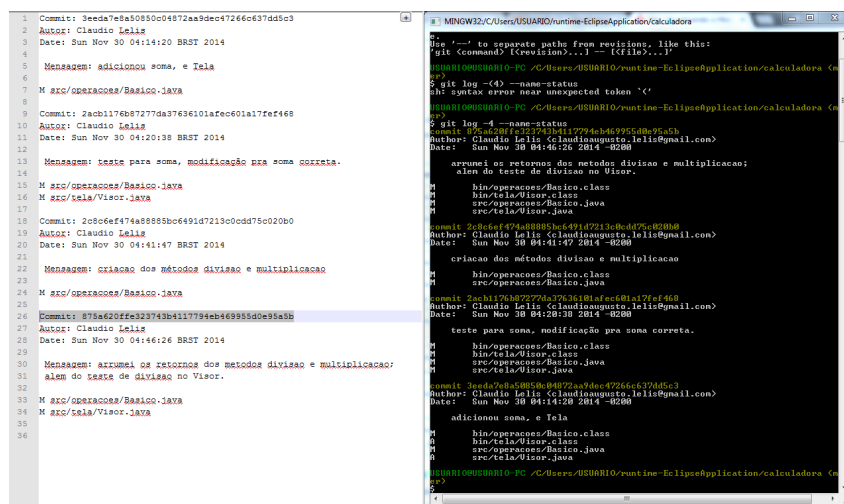


Figura 6.5: Arquivo de *log* no nível de classe

Assim, a análise das imagens apresentaram indícios de que a ferramenta é capaz de gerar o arquivo de *log* de todas as versões do projeto Calculadora a nível de classe.

Num segundo momento, a requisição foi feita para que a ferramenta gerasse o

arquivo de *log* a nível de método. O resultado foi confrontado com as linhas alteradas informadas pelo próprio *GIT*, como pode ser visto na Figura 6.6. A figura mostra o arquivo gerado, a esquerda com um *commit* em destaque onde os métodos *soma* e *main* foram alterados, o que é mostrado à direita são as linhas alteradas no mesmo *commit*, vê-se que as linhas são dos métodos identificados pelo *GiveMe Trace*.

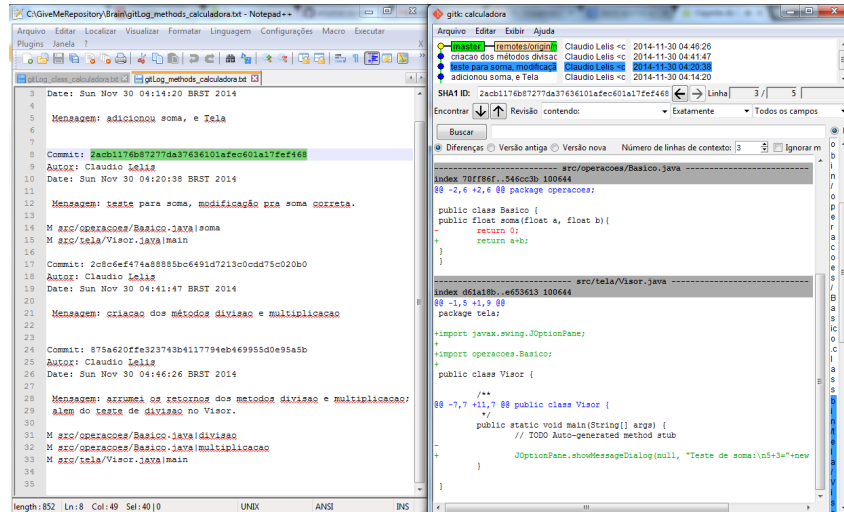


Figura 6.6: Arquivo de *log* no nível de método - destaque na versão 2

O mesmo foi feito com a última versão presente no repositório, em destaque na Figura 6.7. A figura mostra à direita as linhas alteradas no ultimo *commit* feito, pertencentes aos métodos identificados pela ferramenta cujo *log* gerado se encontra à esquerda na figura.

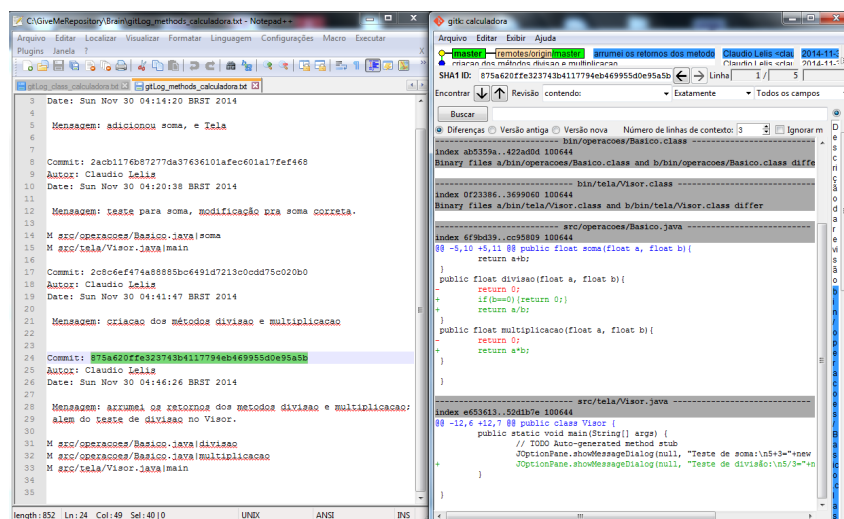


Figura 6.7: Arquivo de *log* no nível de método - destaque na versão 4

A ferramenta apresentou indícios de que também é capaz de gerar o arquivo de

log das versões do projeto Calculadora no nível de método.

6.4 Teste 4

Esse último teste foi desenvolvido com o objetivo de analisar a ferramenta *GiveMe Trace* ao recuperar via *Mylyn-Mantis*, a lista de métodos alterados tendo como origem uma solicitação de mudança. Para este teste foi novamente utilizado o projeto Calculadora, cujas versões são controladas via *GIT*.

Foi simulada a abertura de um *ticket*, uma solicitação de mudança via *Mantis*, requisitando alterações nos métodos *divisao* e *multiplicacao*. O projeto então foi aberto e as alterações foram realizadas nos dois métodos, além disso viu-se a necessidade de alterar o método *main* da classe *Visor* acrescentando uma caixa de diálogo. As alterações nos três métodos foram enviadas ao repositório gerando o *commit* destacado na Figura 6.8.

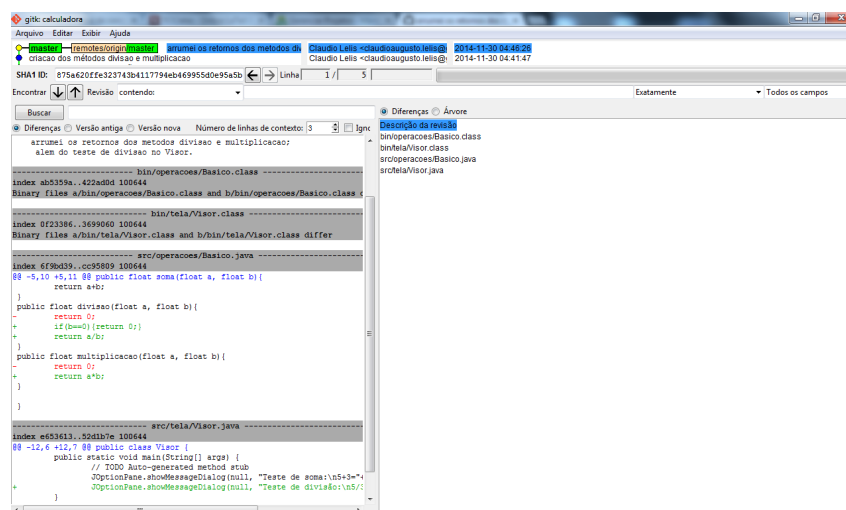


Figura 6.8: Versão originada pela solicitação de mudança

Para este cenário, esperava-se que a ferramenta *GiveMe Trace*, a partir da integração com o *Mylyn-Mantis*, fosse capaz de recuperar a lista de métodos alterados num *commit*, tendo como origem uma solicitação de mudanças registrada via *plugin* do *Mantis*.

Após as alterações terem gerado uma nova versão, os dados da solicitação de mudança no *Mantis* foram atualizados com o número do *commit* gerado e as indicações de *ticket* resolvido. O resultado da submissão do formulário atualizado pode ser visto na Figura 6.9. A figura mostra no campo *Modified Methods* a lista recuperada pelo *GiveMe Trace* para o *commit* referente às alterações originadas por aquela solicitação, evidenciando

os três métodos *divisao*, *multiplicacao* e *main*, conforme o esperado.

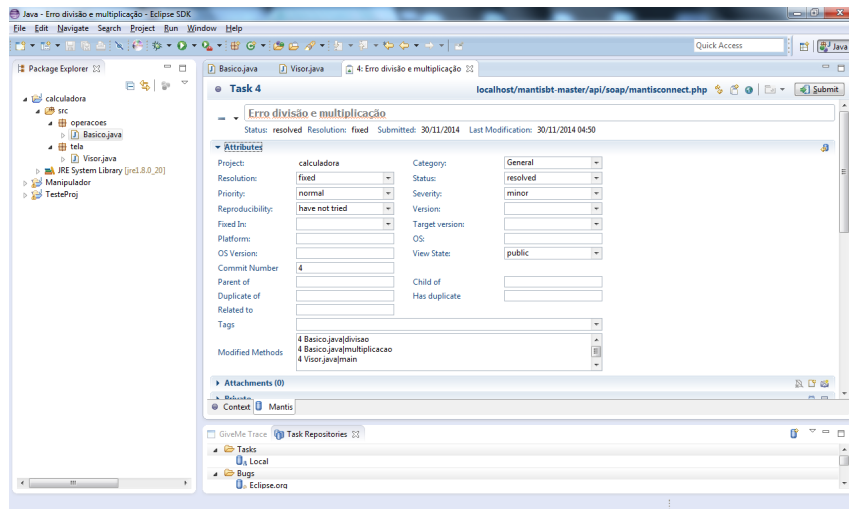


Figura 6.9: Formulário da Solicitação destacando os métodos alterados

Assim, este quarto teste da prova de conceito apresentou indícios de que a ferramenta *GiveMe Trace*, integrada ao *plugin* do *Mantis*, também é capaz de recuperar a lista de métodos alterados num *commit*, com origem numa solicitação de mudança.

7 Considerações Finais e Trabalhos Futuros

A rastreabilidade é um importante fator na análise de impacto das mudanças que o *software* sofre ao longo de sua evolução. A ferramenta *GiveMe Trace* foi desenvolvida com o objetivo de que essa rastreabilidade pudesse ser analisada entre um *commit* realizado no repositório de código fonte, e as classes e métodos alterados durante a atividade de manutenção que originou este *commit*.

Em seguida, a possibilidade de integração do *GiveMe Trace* a outras ferramentas foi explicado, utilizando como base os processos de integração com duas ferramentas, *Mylyn-Mantis* e *GiveMe Views*, destacando os benefícios resultantes destas duas integrações.

Mas a simples integração das ferramentas não traria benefícios sem que as funcionalidades do *GiveMe Trace* fossem averiguadas quanto a sua eficácia. Para tanto, uma prova de conceito foi executada com o objetivo de avaliar se a ferramenta desenvolvida era capaz de detectar classes e métodos alterados entre versões de um dado projeto de código fonte, estando ele versionado ou ainda na cópia de trabalho do desenvolvedor. As funcionalidades da ferramenta *GiveMe Trace* foram analisadas, divididas em quatro testes. Após a execução, a prova de conceito apresentou indícios de que a ferramenta é capaz de gerar o resultado esperado para cada funcionalidade testada.

Ao longo do processo de desenvolvimento da ferramenta *GiveMe Trace*, foram identificadas algumas limitações. Oferecer suporte para repositórios hospedados em servidores externos ou em nuvem é uma delas, a ferramenta apresenta suporte, por enquanto, a apenas repositórios locais. Outro ponto interessante seria a possibilidade de analisar outras extensões de arquivo além do Java, através de um interpretador adequado a cada linguagem, tal qual se utilizou o *JavaParser*. Apesar do processo de desenvolvimento da ferramenta ocorrer de modo gradual e ainda possuir limitações, ela apresenta um potencial de uso.

A prova de conceito apresentada possui as limitações de ter sido realizada em cenário controlado onde se tinha consciência prévia das relações entre artefatos, além de

não ter se repetido os casos em outros projetos.

As dificuldades encontradas estão relacionadas à documentação limitada da ferramenta *Mylyn-Mantis*, sendo necessária a compreensão manual de toda lógica implementada, para a partir de então realizar a integração. Encontrar repositórios reais livres no sistema *Subversion* para a ferramenta *GiveMe Trace* poder analisar e também, encontrar interpretadores de trechos de código em outras linguagens, tal qual o *JavaParser* provê dados para Java, no intuito de aumentar a abrangência de análise da ferramenta.

Espera-se de modo geral, baseado nos processos de integração com as ferramentas *Mylyn-Mantis* e *GiveMe Views* descritos no Capítulo 5, que outras ferramentas também integrem a solução proposta pela ferramenta *GiveMe Trace* como um recurso adicional para enriquecer suas análises e resultados apresentados. Neste sentido, tem-se a expectativa de que a integração da ferramenta *GiveMe Trace* ao *plugin Mylyn-Mantis*, responsável por gerenciar as solicitações de mudanças, e seu uso pelas equipes de desenvolvimento e qualidade, ofereça subsídios mais precisos, a nível de método para verificar se o defeito ou a mudança solicitada foi corretamente implementada com base nos métodos que foram alterados. Por outro lado, ao integrar a ferramenta *GiveMe Trace* ao *GiveMe Views* é esperado aumentar o nível de detalhamento da análise de impacto das mudanças, passando a considerar o impacto entre os métodos alterados.

O problema que a ferramenta *GiveMe Trace* buscou solucionar não afeta apenas as ferramentas com as quais se propôs a integração. Equipes de desenvolvedores e equipes de qualidade que não possuem dados de rastreabilidade que contemplem métodos alterados, ou ferramentas específicas para geração e recuperação de *links* de rastreabilidade como as apresentadas no Capítulo 3, cujas informações geradas não consideram um nível menor de granularidade como os métodos, são candidatos a se tornarem usuários da ferramenta *GiveMe Trace*.

Como trabalhos futuros tem-se a expansão dos princípios da ferramenta *GiveMe Trace* avaliada sob a ótica de diferentes contextos. Numa linha de produto de *software* a possibilidade de estimar os métodos envolvidos na criação de um novo produto da linha. Neste ponto pode-se analisar a real necessidade de determinados métodos para o novo produto, ou a necessidade de uma refatoração e divisão em métodos menores, diluindo

a complexidade do método de origem, além disso gerenciar quais métodos e em quais versões seriam mais adequados para o novo produto da linha também seria possível. Caso outros dados fossem agregados à análise, como tempo gasto de desenvolvimento, artefatos envolvidos no processo e os tempos de cada manutenção seria possível. inclusive, estimar os recursos para implementação do novo produto utilizando estes dados anteriores.

Outra oportunidade para trabalhos futuros é com relação a testes de desempenho da própria ferramenta *GiveMe Trace* na geração de dados de rastreabilidade em repositórios com grandes volumes de dados. Para tal, primeiramente, será necessário classificar os repositórios a serem testados por meio de métricas tais como número de *commits* realizados, número de classes alteradas por *commit*, número de métodos alterados por classe e as linhas alteradas por método. A classificação dos repositórios poderia ser entre pequeno, médio e grande, e o resultado também seria expresso em métricas como tempo de geração dos dados e grau de acurácia dos dados gerados, apresentando a porcentagem de acerto na análise do *GiveMe Trace* confrontando com o que os *softwares* nativos dos sistemas de controle de versão, mostram como alteração por *commit*, tal qual feito na prova de conceito apresentada no Capítulo 6. A aceitação dos testes seria alcançada pela determinação de graus de tolerância tanto a erros quanto ao tempo de geração especificados para cada categoria criada, já que representam a eficácia e a eficiência com que a ferramenta executa suas funcionalidades.

Existe ainda outra possibilidade que seria a partir da integração com a ferramenta *GiveMe Views*, analisar o comportamento do índice de retrabalho da equipe de desenvolvimento nos casos de solicitação de mudança reabertos para correção de erros detectados no *software* em produção. Uma vez que as indicações da ferramenta *GiveMe Views* forem feitas relacionando os métodos impactados por uma alteração em outro método, e ainda, a indicação dos métodos que devem ser observados durante um processo de manutenção de outro método, diminui-se as chances de algum ponto importante passar despercebido pelo desenvolvedor. A equipe de qualidade tem condições de realizar uma verificação mais precisa da mudança implementada confrontando com o que foi solicitado, e ao mesmo tempo a equipe de teste pode prever casos de testes antecipadamente, relacionando os métodos a serem mantidos. Para que este estudo ocorra seria interessante que dados de retrabalho

da equipe tenham sido coletados com o tempo para o confronto com os novos dados e realização das devidas análises apontando o comportamento do índice de retrabalho a partir das indicações do *GiveMe Views* feitas com os dados gerados pelo *GiveMe Trace*.

Referências Bibliográficas

- Lelis, C. A. S. **Calculadora**. <https://github.com/ClaudioLelis/calculadora>. Site, acessado 30 nov. 2014.
- Carneiro, G. **Sourceminer: um ambiente integrado para Visualização multi-perspectiva de software**. 230 f. 2011. Tese de Doutorado - Universidade Federal da Bahia, Instituto de Matemática, Doutorado Multiinstitucional em Ciência da Computação.
- Chacon, S. **Pro Git**. Books for professionals by professionals. Apress, 2009.
- GIT. **Git bash**. <http://git-scm.com/>. Site, acessado 29 nov. 2014.
- IEEE Standard Glossary of Software Engineering Terminology**. Technical report, 1990.
- Gesser, J. V. **Javaparser**. <https://github.com/matozoid/javaparser>. Site, acessado 29 nov. 2014.
- Javed, M. A.; Zdun, U. **A systematic literature review of traceability approaches between software architecture and source code**. In: 18th International Conference on Evaluation and Assessment in Software Engineering (EASE 2014), May 2014.
- Kagdi, H.; Maletic, J. I. ; Sharif, B. **Mining software repositories for traceability links**. In: Program Comprehension, 2007. ICPC'07. 15th IEEE International Conference on, p. 145–154. IEEE, 2007.
- Kitchenham, B. Procedures for performing systematic reviews. **Keele, UK, Keele University**, v.33, 2004.
- Kitchenham, B. A.; Charters, S. **Guidelines for performing Systematic Literature Reviews in Software Engineering**. Technical Report EBSE-2007-01, Keele University, 2007.
- Lelis, C. A. S. **Manual detalhado para integrar o plugin mantisbt ao plugin giveme trace**. <http://implementarsolucoes.com.br/givemetrace.aspx>. Site, acessado 29 nov. 2014.
- Mader, P.; Egyed, A. **Do software engineers benefit from source code navigation with traceability?—an experiment in software change management**. In: Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering, p. 444–447. IEEE Computer Society, 2011.
- Mohan, K.; Xu, P.; Cao, L. ; Ramesh, B. Improving change management in software development: Integrating traceability and software configuration management. **Decision Support Systems**, v.45, n.4, p. 922–936, 2008.

- Motta, A. A. d. **Desenvolvimento de um plugin eclipse para visualizar o histórico de alterações em arquivos utilizando o svn**, 2013. Trabalho de Conclusão de Curso (Graduação em Sistemas de Informação)–Escola de Informática Aplicada, Universidade Federal do Estado do Rio de Janeiro, Rio de Janeiro.
- Mylyn-Mantis. **Mylyn-mantis-connector**. <https://github.com/Mylyn-Mantis/mylyn-mantis>. Site, acessado 29 nov. 2014.
- Neumuller, C.; Grunbacher, P. **Automating software traceability in very small companies: A case study and lessons learned**. In: Automated Software Engineering, 2006. ASE'06. 21st IEEE/ACM International Conference on, p. 145–156. IEEE, 2006.
- Pilato, C.; Collins-Sussman, B. ; Fitzpatrick, B. **Version Control with Subversion**. O'Reilly Media, 2008.
- Pressman, R. **Engenharia de Software**. McGraw Hill Brasil, 2011.
- Rochimah, S.; Kadir, W. W. ; Abdullah, A. H. **An evaluation of traceability approaches to support software evolution**. In: Software Engineering Advances, 2007. ICSEA 2007. International Conference on, p. 19–19. IEEE, 2007.
- Software, T. **Svnkit**. <http://svnkit.com/>. Site, acessado 29 nov. 2014.
- Sengupta, S.; Kanjilal, A. ; Bhattacharya, S. **Requirement traceability in software development process: An empirical approach**. In: Rapid System Prototyping, 2008. RSP'08. The 19th IEEE/IFIP International Symposium on, p. 105–111. IEEE, 2008.
- Sommerville, I.; Melnikoff, S.; Arakaki, R. ; de Andrade Barbosa, E. **Engenharia de Software**. Pearson Prentice Hall, 2008.
- Spanoudakis, G.; Zisman, A. Software traceability: a roadmap. **Handbook of Software Engineering and Knowledge Engineering**, v.3, p. 395–428, 2005.
- Lapes. **Start**. http://lapes.dc.ufscar.br/tools/start_tool. Site, acessado 29 nov. 2014.
- Tavares, J. F.; Braga, R.; David, J. M. N.; Araújo, M. A. P. ; Campos, F. Giveme metrics-um framework conceitual para extração de dados históricos sobre evolução de software. **X Simpósio Brasileiro de Sistemas de Informação (SBSI), Londrina/PR**, 2014.
- Tavares, J. F.; Braga, R.; David, J. M. N.; Araújo, M. A. P.; Campos, F. ; Carneiro, G. d. F. Giveme views: uma ferramenta para análise visual de evolução de software a partir de repositórios de dados. a publicar.
- Team, T. **Tortoisesvn**. <http://tortoisesvn.net/>. Site, acessado 29 nov. 2014.
- Walters, B.; Shaffer, T.; Sharif, B. ; Kagdi, H. **Capturing software traceability links from developers' eye gazes**. In: Proceedings of the 22nd International Conference on Program Comprehension, p. 201–204. ACM, 2014.

Wiederseiner, C.; Garousi, V. ; Smith, M. **Tool support for automated traceability of test/code artifacts in embedded software systems**. In: Trust, Security and Privacy in Computing and Communications (TrustCom), 2011 IEEE 10th International Conference on, p. 1109–1117. IEEE, 2011.

Eclipse. **jgit**. <https://eclipse.org/jgit/>. Site, acessado 29 nov. 2014.