UNIVERSIDADE FEDERAL DE JUIZ DE FORA
INSTITUTO DE CIÊNCIAS EXATAS
PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Jacimar Fernandes Tavares

GiveMe Infra: Uma Infraestrutura Baseada em Múltiplas Visões Interativas para Apoiar a Evolução Distribuída de Software

Juiz de Fora

2015

UNIVERSIDADE FEDERAL DE JUIZ DE FORA
INSTITUTO DE CIÊNCIAS EXATAS
PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Jacimar Fernandes Tavares

GiveMe Infra: Uma Infraestrutura Baseada em Múltiplas Visões Interativas para Apoiar a Evolução Distribuída de Software

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação, do Instituto de Ciências Exatas da Universidade Federal de Juiz de Fora como requisito parcial para obtenção do título de Mestre em Ciência da Computação.

Orientador: José Maria Nazar David
Coorientador: Marco Antônio Pereira Araújo

Juiz de Fora
2015

Ficha catalográfica elaborada através do Programa de geração
automática da Biblioteca Universitária da UFJF,
com os dados fornecidos pelo(a) autor(a)

Fernandes Tavares, Jacimar.
GiveMe Infra: Uma Infraestrutura Baseada em Múltiplas Visões Interativas para Apoiar a Evolução Distribuída de Software / Jacimar Fernandes Tavares. -- 2015.
114 f. : il.

Orientador: José Maria Nazar David
Coorientador: Marco Antônio Pereira Araújo
Dissertação (mestrado acadêmico) - Universidade Federal de Juiz de Fora, Instituto de Ciências Exatas. Programa de Pós-Graduação em Ciência da Computação, 2015.

1. Evolução de Software. 2. Colaboração de Software. 3. Visualização de Software. I. Nazar David, José Maria, orient. II. Pereira Araújo, Marco Antônio, coorient. III. Título.

Jacimar Fernandes Tavares

"GiveMe Infra: Uma Infraestrutura baseada em Múltiplas Visões Interativas para Apoiar a Evolução Distribuída de Software "

Dissertação apresentada ao Programa de Pós-graduação em Ciência da Computação da Universidade Federal de Juiz de Fora como requisito parcial à obtenção do grau de Mestre.

Aprovada em 16 de março de 2015.

BANCA EXAMINADORA

Prof. Dr. José Maria Nazar David – Orientador
Universidade Federal de Juiz de Fora

Prof. Dr. Marco Antônio Pereira Araújo - Coorientador
Universidade Federal de Juiz de Fora

Profª. Dra. Fernanda Cláudia Alves Campos
Universidade Federal de Juiz de Fora

Prof. Dr. Glauco de Figueiredo Carneiro
Universidade Salvador

A Deus em primeiro lugar. Aos meus pais e irmãos, noiva e amigos.

AGRADECIMENTOS

Você que lê esse texto pode até não fazer idéia de como é custoso concluir um mestrado em computação, mas talvez saiba como é importante poder contar com pessoas especiais nas jornadas em que se aventura. Comigo não foi diferente. Cada um dos aqui mencionados tiveram um papel fundamental na materialização deste projeto. Vamos a eles:

A Deus sobre todas as coisas. Quem o conhece sabe como seu agir é de infinita bondade e seus planos são perfeitos. Por tudo que vi e vivi, obrigado.

Aos meus pais e irmãos que, sem sombra de dúvida são a base de tudo. Foram eles que me deram todo o suporte e tranquilidade para poder concluir essa empreitada. A cada gesto, a cada palavra, a cada conselho, obrigado.

A minha noiva Sara (e muito em breve esposa), que com muita sabedoria e calma soube lidar com minha ausência em muitos momentos. Lembro-me de cada palavra de incentivo, de cada afago, de cada gesto. Por estar sempre e para sempre comigo, obrigado.

Aos meus amigos pelo apoio e incentivo. Não poderia deixar de citar aqui, como representantes deles, Ronan Rocha, a quem sempre me incentivou e me proporcionou momentos de muita alegria durante essa empreitada e Joabe Geraldo, por me incentivar ouvir e ajudar. Aos amigos que fiz no mestrado, obrigado de coração. Foi um prazer conhecer todos vocês.

Aos meus orientadores, que me deram apoio e incentivo durante todo o mestrado. José Maria, por ter aceitado o desafio em um momento difícil pra mim no mestrado, obrigado. Marco, pela parceria e amizade que já duram alguns anos, obrigado. Este trabalho não seria nada sem vocês.

Ao meu pai e irmãos na fé, que intercederam por mim em oração. Obrigado a todos e que Deus esteja com vocês.

Aos meus professores, que durante toda a minha vida acadêmica me incentivaram e me ajudaram, as vezes, muito além de suas obrigações. Em especial aos professores Marcelo Daibert, Clayton Fraga e Sergio Murilo Stempliuc, pelo incentivo, indicação e amizade.

Aos parceiros de trabalho, Emmanuel, Marcos Miguel e Cláudio, por terem contribuído muito para que este trabalho se tornasse possível. Obrigado.

"De que vale ganhar o mundo
todo e perder a sua alma?
(Mateus 16:26),
portanto...
Combati o bom combate,
completei a corrida, perseverei
na fé! (2 Timóteo 4:7)"
Bíblia Sagrada

RESUMO

Existem diversas tecnologias desenvolvidas nas áreas de manutenção e evolução colaborativa de software associadas à visualização de software, objetivando resolver problemas de empresas e grupos de usuários. Em muitos casos é necessário integrar soluções dessas áreas visando resolver um dado problema relacionado à evolução do software. Este problema se torna mais intenso quando se trata de equipes geograficamente distribuídas. Neste sentido, foi desenvolvida GiveMe Infra, uma infraestrutura para apoio a realização de atividades de manutenção e evolução de software, realizadas por equipes co-localizadas ou geograficamente distribuídas. Tais atividades são apoiadas por diferentes visualizações de software que permitem ao usuário obter diferentes perspectivas sobre as informações disponibilizadas. Um estudo experimental foi realizado objetivando verificar a viabilidade de uso da solução. Os resultados obtidos são apresentados, bem como os trabalhos futuros em relação à infraestrutura proposta.

Palavras-chave: Evolução de Software. Colaboração de Software. Visualização de Software.

ABSTRACT

There are several technologies developed to support collaborative software maintenance and evolution, as well as software visualization, in order to solve problems of companies and group of users. In many cases is necessary to integrate solutions already available in these areas to solve a given problem. These problems become intense when geographically dispersed groups are involved in software maintenance and evolution activities. In this sense, an infrastructure, named GiveMe Infra, was developed to support these activities when performed by co-located or geographically distributed teams. These activities are supported by different software visualizations that allow the user get different perspectives about the provided information. An experimental study were carried out aiming to verify the feasibility of the solution as well as the hypotheses. The obtained results related to the experiments and future works are presented.

Keywords: Software Evolution. Software Collaboration. Software Visualization.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1. INTRODUÇÃO

Ao longo dos anos, diferentes publicações tais como (MAO, 2011) (SYEED et al., 2013) e (LUNGU et al., 2014) surgiram no contexto de engenharia de software, especificamemente em áreas como manutenção, evolução e colaboração de software, trazendo novas técnicas, métodos, ferramentas e teorias, cabendo a profissionais da área a iniciativa de se manterem atualizados quanto as novidades apresentadas.

1.1 MOTIVAÇÃO

Na área de Evolução de Software, surgiram teorias que permeiam todo o ciclo de manutenção de um software ao longo da sua vida e que apoiam desenvolvedores e pesquisadores no entendimento de como se dá a evolução de um produto (LEHMAN, 1996). Não é objetivo deste trabalho discutir o termo Leis, mas sim aproveitar o conhecimento gerado por esses estudos. Na área de Colaboração surgiram elementos como Cooperação, Comunicação e Coordenação, melhorando o entendimento sobre a divisão de tarefas entre equipes distribuídas (FUKS et al., 2003). Já em Visualização de Software surgiram ambientes interativos múltiplas visões (AIMV) que fornecem aos usuários diferentes perspectivas sobre um mesmo conjunto de dados (SILVA et al., 2012). Todas essas áreas são abrangidas pela solução desenvolvida neste trabalho que atua apoiando equipes de manutenção e evolução de software.

Ao mesmo tempo em que surgem novidades nas áreas mencionadas, surgem também novos problemas organizacionais que podem ser resolvidos usando recursos existentes ou combinando recursos de diferentes áreas, tal como é o problema abordado deste trabalho e a solução apresentada. Este trabalho se inicia a partir da detecção de um problema organizacional encontrado em uma empresa de desenvolvimento de software para gestão empresarial, cuja solução apresentada foi possível graças à combinação de recursos das áreas de manutenção, evolução, colaboração e visualização de software.

1.2 PROBLEMA

A realização de uma consultoria mostrou que uma empresa de desenvolvimento possuía uma solução chamada *Service Center*, para registro e controle de Solicitações de Mudanças. A cada solicitação registrada, podendo ser proveniente de um atendimento ao cliente via suporte ou internas à própria empresa, um ou vários desenvolvedores eram alocados para resolução (caso envolvesse diferentes áreas dentro da mesma empresa, como analistas de banco de dados, *designers*, e programadores).

Uma solicitação de mudança, quando finalizada pela equipe responsável, era então repassada ao setor de qualidade que seria responsável por validar as alterações realizadas pelos desenvolvedores, onde poderia aceitar o fechamento da mesma ou rejeitar. Aceitar implicaria em gerar uma nova versão do software. Rejeitar implicaria em devolver a solicitação de mudança à equipe responsável por efetuá-la, para que pudessem então efetuar as mudanças apontadas pelo setor de qualidade.

Uma particularidade encontrada no processo de registro e controle das solicitações de mudança está relacionada às informações de rastreabilidade que são adicionadas sempre que uma solicitação de mudança era finalizada: módulos e componentes alterados. O termo módulo é designado para classificar: (i) software (produtos que a empresa comercializa), (ii) módulos (como módulo financeiro, módulo fiscal, entre outros) e (iii) pacotes de código (*packages*, tais como os criados em ambientes como *NetBeans* e *Eclipse*). Já o termo componente refere-se a: (i) classes ou arquivos de código alterados, (ii) formulários (tais como *jFrame* e *jDialog*), (iii) DLLs (bibliotecas de código e dados) e (iv) *scripts* de banco de dados. A rastreabilidade é dita como as ligações existentes entre, no contexto da empresa observada, módulos e componentes que sofreram algum tipo de manutenção ao longo da resolução da solicitação de mudança. Permite que *links* sejam criados e mantidos entre artefatos gerados e alterados durante o ciclo de vida de um software (WALTERS et al., 2014). Os módulos e componentes alterados são obtidos de sistemas de controle de versão que, tal como dito em (MOHAN et al., 2008), podem fornecer informações úteis para entender a evolução do software. Assim é feito na empresa que foi alvo da consultoria mencionada: módulos e componentes alterados são obtidos de sistemas controladores de versão e são adicionados a sua respectiva solicitação de mudança, quando finalizada, criando assim uma base de dados históricos composta por informações de rastreabilidade e solicitações de mudanças.

O problema encontrado na empresa é que a mesma estava insatisfeita com o número de solicitações de mudanças reprovadas pelo setor de qualidade. As reprovações se davam por dois motivos em especial, segundo apontado pelo gerente da empresa: a falta de controle sobre a evolução do software e a complexidade do projeto de código dos produtos. Ao longo dos anos e dos vários ciclos de manutenção sofridos, os produtos se afastaram de suas arquiteturas iniciais dado que não havia um controle sobre sua evolução. Alterações eram realizadas sem o devido planejamento da evolução do projeto de código. Isso fez com que os projetos de códigos dos softwares se tornassem confusos e de difícil manutenção, dado que nenhuma solução que permitisse melhorar a compreensão sobre o projeto em manutenção era utilizada. Mediante a isso, as equipes que efetuam as solicitações de mudanças tem dificuldade para estimar, por exemplo, quais componentes devem ser mantidos sempre quando outros componentes forem alterados, dado que eles não possuíam nenhum tipo de solução (técnica, metodologia, ferramenta, *framework*, ou modelo) que fornecesse este tipo de indicação. Também há a dificuldade para entender quais componentes tem maior prioridade de manutenção, quando um outro componente é alterado. Em um cenário onde a resolução de uma solicitação de mudança é feita de forma colaborativa entre membros de uma equipe geograficamente distribuída a situação poderia se agravar em alguns casos, dado que, modificações sem controle sobre o código fonte poderiam gerar impactos em outros componentes sobre responsabilidade de outros membros da mesma equipe. Além disso, a coordenação das atividades de manutenção não contava com nenhuma solução para que as informações de rastreabilidade inseridas nas solicitações de mudanças pudessem ser visualizadas (além do próprio formulário que exibe a solicitação de mudança), de forma a facilitar o trabalho do gerente de projetos na coordenação de suas atividades e equipes na tarefa de identificar as ligações existentes entre módulos e componentes alterados. O que se tinha era apenas uma base de dados históricos com informações de alterações, o que não ajudava o setor de qualidade em nada, além da realização de consultas para verificar o que foi alterado a nível de código fonte.

A pesquisa por trás deste trabalho se inicia a partir da identificação do cenário descrito, envolvendo a consultoria realizada e a empresa analisada. Foi estabelecida uma parceria com a empresa que, a partir deste ponto e durante todo este trabalho, será identificada como Empresa Parceira 1, por questões de confidencialidade. A motivação para o desenvolvimento deste trabalho surgiu a partir da identificação do cenário encontrado na Empresa Parceira 1 e da necessidade que possui de (i) melhorar a

compreensão sobre o projeto em manutenção (ii) melhorar o processo de manutenção, melhorando o controle sobre a evolução do software, (iii) tratando aspectos de colaboração entre equipes distribuídas, (iv) utilizando recursos visuais que facilitem as tarefas realizadas pela equipe de manutenção, gerência e setor de qualidade.

Foi conduzida uma Revisão Sistemática de Literatura (*GIVEME INFRA*, 2014) com o intuito de levantar publicações que possam indicar soluções que combinem os seguintes temas: (i) manutenção de software; (ii) evolução de software (iii) colaboração em software; (iv) visualização de software. O objetivo foi o de encontrar soluções como técnicas, metodologias, ferramentas, *frameworks,* ou modelos que pudessem auxiliar na resolução do problema encontrado na Empresa Parceira 1. Através da revisão realizada, não foi possível encontrar soluções que englobem todos os temas pesquisados, o que levou à formulação do problema geral tratado neste trabalho, que é: a falta de uma solução (técnica, metodologia, ferramenta, *framework,* ou modelo) que apoie a manutenção e a evolução de software, no contexto de equipes geograficamente distribuídas ou co-localizadas, na tarefa de dar manutenção em produtos de software sobre supervisão de uma gerência, que toma iniciativas com base nas atividades realizadas pelas equipes de manutenção. A partir deste ponto do trabalho, o termo "soluções existentes" será sempre usado referindo-se às soluções (técnicas, metodologias, ferramentas, *frameworks,* ou modelos) encontradas na Revisão Sistemática de Literatura realizada.

1.3 OBJETIVO

Mediante ao problema geral apresentado, definiu-se o seguinte objetivo: desenvolver uma solução, ou integrar soluções existentes, com o intuito de apoiar a manutenção e a evolução de software, no contexto de equipes geograficamente distribuídas ou co-localizadas, na tarefa de dar manutenção em produtos de software sobre supervisão de uma gerência, que toma iniciativas com base nas atividades realizadas pelas equipes de manutenção. O enfoque da solução está na colaboração entre equipes de manutenção de software distribuídas na tomada de decisões e no acompanhamento do processo de manutenção de software com base nas informações oriundas da extração de dados históricos de software.

A solução apresentada neste trabalho, dado o problema geral destacado, foi a de desenvolver uma infraestrutura que integra ferramentas de compreensão de software

com ferramentas desenvolvidas no contexto deste trabalho. Uma infraestrutura conceitualmente é caracterizada pela existência de um conjunto de modelos e arquiteturas, visando formar uma base para especificação e implementação de soluções em um dado domínio ao reutilizar tais conjuntos (BRAGA, 2000).

GiveMe Infra, é uma infraestrutura baseada em múltiplas visões interativas para apoio da evolução distribuída de software. Integra ambientes de compreensão de software interativos baseados em múltiplas visões (AIMVs) com ferramentas que apoiam a colaboração entre equipes geograficamente distribuídas nas tarefas de manutenção e evolução de software. Em (SILVA et al., 2012) é dito que AIMVs fornecem meios para que análises sobre dados sejam feitas a partir da utilização de visualizações, e que tais ambientes devem fornecer meios para que haja a coordenação entre as visualizações. Também é dito que, combinar diferentes visualizações pode permitir a análise de dados em diferentes perspectivas. A interatividade fica por conta da definição de recursos que permitem interagir diretamente nas visualizações fornecidas como, por exemplo, através do envio de mensagens entre membros da equipe.

Mediante ao objetivo deste trabalho, foram estabelecidas (além da parceria com a Empresa Parceira 1) parcerias com o SINAPAD (Sistema Nacional de Processamento de Alto Desempenho) e com outra empresa de desenvolvimento de software, que será chamada neste trabalho de Empresa Parceira 2. O objetivo é, além de poder analisar outros repositórios de dados históricos cedidos pelas parcerias, também verificar a viabilidade de uso da solução em outros locais além da Empresa Parceira 1.

1.4 HIPÓTESE

A seguinte hipótese foi estabelecida para este trabalho após a definição da *GiveMe Infra*:

- H $_{nula}$: *GiveMe Infr*a não é capaz de auxiliar equipes distribuídas ou colocalizadas na tarefa de manter e acompanhar a evolução de projetos de software;
- H $_{alternativa}$: *GiveMe Infra* é capaz de auxiliar equipes distribuídas ou colocalizadas na tarefa de manter e acompanhar a evolução de projetos de software.

1.5 METODOLOGIA

A metodologia adotada neste trabalho é descrita nas etapas a seguir e se inicia após a descoberta do problema na Empresa Parceira 1:

- Etapa (i): análise dos resultados gerados na revisão sistemática de literatura: nesse passo foram analisadas as publicações recuperadas pela revisão sistemática de literatura feita sobre o tema deste trabalho (manutenção, evolução, colaboração e visualização de software) objetivando conhecer o que já foi feito na área em termos de tecnologia e investigar quais eram capazes de apoiar a resolução do problema geral deste trabalho;
- Etapa (ii): definição dos trabalhos relacionados: etapa onde foram documentados os trabalhos relacionados, obtidos com base na revisão sistemática de literatura mencionada na Etapa (i);
- Etapa (iii); desenvolvimento da solução - passo onde foi planejada e desenvolvida a *GiveMe Infra*;
- Etapa (iv): formulação de hipótese - aqui foi definida a hipótese geral deste trabalho, para ser aceita ou rejeitada com base nos resultados do estudo experimental realizado;
- Etapa (v): definição do estudo experimental - passo onde foi planejado o estudo experimental;
- Etapa (vi): execução do estudo experimental;
- Etapa (vii): documentação dos resultados do estudo experimental - etapa onde são documentados os resultados obtidos com a execução do estudo experimental;
- Etapa (viii): validação da hipótese geral - etapa de validação da hipótese geral do trabalho.

  Ao final foi realizada a documentação de todo o conteúdo gerado neste trabalho.

1.6 ESTRUTURA DA DISSERTAÇÃO

Este trabalho está dividido da seguinte forma: o capítulo 1, Introdução, apresenta uma visão geral do problema que está sendo tratado, o objetivo do trabalho, a solução

desenvolvida e a metodologia utilizada. O capítulo 2, Referencial Teórico, apresenta os temas que são base para este trabalho, fornecendo ao leitor uma visão geral sobre a teoria necessária. Além disso, também são apresentados os trabalhos relacionados. O capítulo 3 apresenta a solução desenvolvida: *GiveMe Infra*. O capítulo 4 apresenta a avaliação da solução realizada através da aplicação de um estudo experimental. Por último, o capítulo 5, apresenta as conclusões.

2. REFERENCIAL TEÓRICO

Este capítulo é destinado à apresentação e discussão de temas que são a base deste trabalho: manutenção de software, evolução de software, colaboração e visualização de software. Além de abordar estes temas, este capítulo também destina-se a fornecer uma visão geral das relações existentes entre eles, dado que o problema geral tratado neste trabalho tem como resolução o desenvolvimento de uma solução que combina os temas citados.

Inicialmente serão apresentadas as definições de manutenção de software e os tipos de manutenção explorados neste trabalho. Em seguida, são apresentadas definições de evolução de software. Posteriormente, são apresentados aspectos de colaboração que fazem parte de atividades de manutenção e evolução realizadas de forma distribuída. A sessão seguinte explicita a relação entre os temas discutidos nas seções anteriores objetivando a resolução do problema geral deste trabalho. Depois são apresentados os trabalhos relacionados e as lacunas deixadas por eles e ao final, uma seção de fechamento do capítulo.

2.1 MANUTENÇÃO DE SOFTWARE

Segundo (SOMMERVILLE, 2007), é inevitável que sistemas de software mudem com o passar do tempo. Dentre os motivos estão: a necessidade de correções de defeitos, de adaptação às novas plataformas e melhoramentos, bem como o aumento do desempenho e outras melhorias não funcionais, isto é, que não impactam nas funcionalidades implementadas ao ponto de inserir novas ou remover alguma existente. São apresentados três tipos de manutenções: (i) manutenção corretiva, refere-se ao tipo voltado para os erros encontrados no software, (ii) manutenção adaptativa, que refere-se a modificações visando adequar o software a um novo ambiente, por exemplo, suportando uma nova plataforma e (iii) manutenção evolutiva (ou perfectiva), que significa melhorar o projeto de código existente objetivando simplificá-lo, melhorar seu desempenho, aumentar sua legibilidade ou acrescentar novas funcionalidades. Em (IEEE93, 1993) é apresentada uma definição semelhante. É dito que manutenção de

software refere-se às atividades realizadas após o desenvolvimento e entrega de um produto objetivando a correção de falhas detectadas, a melhoria de desempenho e outros atributos, ou adaptação do produto às novas exigências do ambiente, como por exemplo, mudanças na legislação.

Pressman (2006) define quatros tipos de manutenção de software. São elas: (i) manutenção corretiva, para correção de falhas encontradas após a etapa de testes, (ii) manutenção adaptativa, para adaptações feitas no sistema por conta, por exemplo, de mudanças na legislação, (iii) manutenção evolutiva, objetivando a melhoria do projeto de código existente e (iv) manutenção preventiva, para tratar modificações feitas no software como forma de prevenção de possíveis problemas no futuro. Essa divergência entre os tipos de manutenção foi identificada em (SOMMERVILLE, 2007). Independente da divergência relatada, neste trabalho serão abordados apenas os três tipos de manutenção definidos em (SOMMERVILLE, 2007).

2.2 EVOLUÇÃO DE SOFTWARE

É inevitável que softwares passem um dia por um processo de envelhecimento que poderá culminar na sua extinção (LORGE, 1994). O que se pode fazer nestes casos adiar, tomando atitudes, tais como: obter conhecimento sobre as causas do processo de envelhecimento ao qual o sistema está submetido, realizar algumas ações objetivando limitar as causas identificadas (como refatoração de código), remover, mesmo que de forma temporária, os danos causados e por fim, se preparar para o momento em que a manutenção do software não seja mais possível ou viável. O processo de envelhecimento do software pode ser acompanhado através da análise da sua evolução. O termo evolução de software é designado como sendo a observação sobre as mudanças que ocorreram em um software ao longo do seu ciclo de manutenção, bem como os impactos que tais mudanças causaram, diferenciando-se do termo manutenção, que se refere a somente as atividades de manutenção realizadas em um software (LEHMAN, 1996).

Durante a evolução do software, alterações provenientes de manutenções realizadas podem ser observadas. Em (SNEED, 2001) é definido que alterações no software constituem um ciclo formado pelas fases:

- Requisição de mudanças: período em que são definidas as mudanças a serem realizadas;
- Fase de Planejamento: diz respeito à (i) Compreensão do programa: etapa em que o software a ser alterado é analisado, buscando conhecimento sobre ele; (ii) Análise do Impacto: etapa na qual a alteração a ser realizada é analisada, visando conhecer os riscos de sua efetivação, bem como o que será necessário alterar;
- Implementação da alteração: se relaciona à execução da alteração planejada. É formada pelas seguintes etapas: (i) Reestruturação para a mudança: contempla as alterações necessárias para reestruturar o sistema de modo a acomodar as novas implementações; (ii) Propagação da alteração: teoria fundamental neste trabalho, pois diz respeito à análise de componentes vizinhos a um componente alterado. Cada vez que um componente é alterado, é dito que os componentes que possuem relação com ele (ou seja, vizinhos) devem ser verificados a fim de descobrir se alterações também serão necessárias neles. Alterações em um componente vizinho é denominada alteração secundária;
- Verificação e validação da alteração efetuada;
- Redocumentação: diz respeito à adaptação da documentação existente ou à definição de nova documentação, objetivando registrar as alterações realizadas e a nova visão arquitetural do software, dentre outras coisas;

Em (LEHMAN, 1996) são definidas oito leis sobre a evolução de software que podem ajudar no entendimento de algumas questões neste trabalho, entretanto é importante destacar que não é objetivo deste trabalho discutir o termo Leis, e sim apenas utilizar o arcabouço conceitual por elas apresentado São elas:

1. Alteração contínua: lei que diz que alterações são continuamente realizadas nos softwares dada a existência de *feedbacks* e a necessidade de se adaptar às mudanças no ambiente onde estão inseridos;
2. Aumento da complexidade: diz que a complexidade de um projeto de software tende a aumentar, caso medidas não sejam tomadas para reduzi-la, dado que são realizadas mudanças sobre mudanças, para adaptar às novas exigências estabelecidas;
3. Auto regulação: refere-se a questões que impactam diretamente no software, como interesses organizacionais e *feedbacks*;

4. Preservação da estabilidade organizacional: diz respeito a tomada de decisões em uma empresa. É dito que o esforço para que um software possa crescer e evoluir está ligado à capacidade de tomar decisões no âmbito organizacional;
5. Conservação da familiaridade: lei que aborda a importância de se manter a familiaridade com o que vem sendo alterado em um sistema. É fundamental que todos os envolvidos com o sistema possam se inteirar sobre as mudanças realizadas de modo a acompanhar a evolução do software;
6. Crescimento contínuo: diz que, para manter a satisfação do usuário em relação ao software desenvolvido, o mesmo deve ter seu conteúdo funcional aumentado. Um exemplo seria os aplicativos de redes sociais, que continuamente vem oferecendo novos recursos aos seus usuários, objetivando mantê-los sempre conectados a plataforma;
7. Declínio da qualidade: lei que diz que todo sistema tende a declinar na questão da qualidade ao longo dos anos, a menos que iniciativas sejam tomadas com o intuito de sempre adaptá-lo a mudanças no ambiente operacional em que ele está inserido;
8. Sistema de *feedback*: relata a importância do *feedback* no processo de alteração do sistema. Assim, é possível criar indicadores que podem apontar diferentes rumos na gestão do software;

A evolução de software pode ser analisada de forma colaborativa, onde diferentes envolvidos podem fornecer diferentes perspectivas sobre a informação analisada em conjunto com outras pessoas. A colaboração em software é discutida na próxima seção.

2.3 COLABORAÇÃO EM SOFTWARE

Como já foi dito neste trabalho, Manutenção de Software é um processo de mudanças em um sistema desenvolvido (SOMMERVILLE, 2007). Já a Colaboração pode permitir a grupos (equipes de manutenção, por exemplo) se organizarem de tal forma que haja predominância de elementos como Cooperação, Comunicação e Coordenação (FUKS et al., 2003). A Percepção pode ser considerada um elemento de Colaboração, dado que ela diz respeito às informações das interações realizadas entre membros de uma equipe. Como a manutenção de software pode ser realizada de forma

colaborativa (D'AMBROS et al., 2008), então é possível ter elementos de percepção associados a ela, como por exemplo, um recurso que permita ao usuário obter conhecimento de que uma mensagem enviada por ele foi visualizada pelo destinatário.

Cooperação é a ação em conjunto entre membros de uma equipe que compartilham o mesmo espaço de trabalho, no caso, o mesmo espaço do ambiente de colaboração, objetivando realizarem alguma tarefa em parceria. Um exemplo envolvendo manutenção colaborativa seria a realização de atividades (como reuniões), através da qual há cooperação para se obter entendimento sobre os motivos pelos quais um módulo do sistema apresentou um alto número de defeitos em relação aos demais. O trabalho cooperativo neste caso poderá envolver todas as pessoas responsáveis pelo módulo com maior incidência de defeitos, como desenvolvedores, arquitetos, DBAs (*Data Base Administrators*) e equipe de qualidade.

Comunicação diz respeito à efetividade com a qual uma mensagem é entregue ao destinatário e, ao mesmo tempo, é entendida, gerando concordância ou conflitos necessários. Uma falha na comunicação pode fazer com que o destinatário não compreenda a intenção da comunicação estabelecida e, assim, não consiga firmar compromissos com o remetente. Um exemplo seria a realização de atividades nas quais há troca de informações entre membros da equipe sobre os riscos associados à tarefa de dividir um módulo que apresenta altas taxas de complexidade.

Coordenação é a tarefa de articular um grupo de pessoas trabalhando em um ou vários processos que se conectam de alguma forma. Coordenar, portanto, é articular a pré-realização de uma tarefa, a realização da tarefa e o pós-realização da tarefa. Um exemplo seria a realização de atividades de preparação do tema de uma reunião, convidar os participantes, conduzir a reunião rumo à tomada de decisões e delegar as decisões tomadas aos responsáveis por executá-las.

Em resumo, a colaboração na manutenção pode ser vista, em termos práticos, na análise de pontos de um software que podem se beneficiar com um processo de manutenção corretiva, adaptativa ou evolutiva apoiada pelos elementos de colaboração apresentados.

A colaboração pode existir através do envio e recebimento de mensagens entre membros de uma equipe de diferentes formas, como exemplo através de mensagens de

correio eletrônico ou mensagens inseridas diretamente em visualizações, que é o tema da próxima seção.

2.4 VISUALIZAÇÃO DE SOFTWARE

Visualização é uma importante forma de se obter compreensão e é fundamental na tarefa de se criar um modelo mental sobre as informações. Visualizar uma dada informação pode ser útil no processo de compreensão visando a obtenção de um conhecimento mais profundo sobre o objeto analisado. A forma como informações são disponibilizadas pode caracterizar a necessidade de se utilizar um Ambiente Interativo Múltiplas Visões (AIMV). AIMVs são ambientes que fornecem diferentes visualizações para o mesmo conjunto de informações, permitindo também que dados de interação entre usuários sejam registrados (SILVA et al., 2012). Um exemplo seria a inserção de mensagens de alerta em uma visualização que permita ressaltar que a complexidade de uma classe está muito acima dos limites estabelecidos pela organização. Múltiplas visões também podem evitar que interpretações errôneas possam surgir, se comparadas com a análise realizada sobre um conjunto de dados visualizado apenas em uma única visão. Neste trabalho são utilizados dois paradigmas de visualização: um que implementa um AIMV, ou seja, múltiplas visões para o mesmo conjunto de dados e outro que implementa diferentes visões para diferentes conjuntos de dados.

No paradigma que implementa um AIMV, tem-se as visões propostas por (CARNEIRO, 2011) e (SILVA, 2013). No paradigma que implementa diferentes visões para diferentes conjuntos de dados tem-se visões tais como: *Metric View*, que permite a visualização de um conjunto de métricas processadas com a *GiveMe Infra*. *Tree View*, exibe um conjunto de métodos de uma classe e seus relacionamentos (ou seja, outros métodos que possuem algum tipo de relação com um método selecionado) e *Deep View*, que exibe informações em forma de grafos que podem mudar, de acordo com a interação do usuário, gerando novos nós e arestas.

Visualizações podem ser utilizadas tanto para se obter múltiplas visões sobre o mesmo conjunto de dados, como também pode ser usada para visualizar a evolução desse conjunto de dados ao longo dos anos. Um exemplo seria a visualização da contribuição dada por cada desenvolvedor em um projeto ao longo dos anos. Com uma

visualização que permita acompanhar as variações do conjunto de dados isto se tornará possível (SANTANA et al., 2001).

2.5 COLABORAÇÃO NA MANUTENÇÃO, EVOLUÇÃO E VISUALIZAÇÃO DE SOFTWARE.

A manutenção e a evolução de software podem se beneficiar com o uso de ferramentas de visualização, dado que elas podem fornecer informações sobre artefatos de software que não seriam facilmente conhecidos se examinados diretamente os repositórios que contêm tais informações (STOREY et al., 2008).

A visualização na manutenção e na evolução de software tem sido aplicada de modo que, as implementações existentes sejam capazes de expor informações em diferentes formatos, tais como: imagens, diagramas, animações e textos. Estas informações podem ajudar na tarefa de obter entendimento sobre o projeto de código a ser mantido. Técnicas de visualização podem expor informações sobre artefatos e, quando combinadas com múltiplas visões, podem exibir diferentes aspectos da informação compartilhada. Há ainda a possibilidade de se criar compartilhamentos de informações de forma diferenciada, de tal modo que detalhes da informação possam ser ocultados ou exibidos para um grupo determinado de pessoas. Como exemplo pode-se ter informações a nível gerencial e financeiro que não devem ser visualizadas pelo pessoal que está no nível operacional.

É possível, além de associar visualização com manutenção e evolução de software, associar manutenção e evolução distribuídas com o conceito de visualização. Neste contexto, quando uma informação é compartilhada, deve-se levar em conta que há uma diversidade de autores que irão se deparar com o que foi compartilhado, objetivando ter a mesma compreensão que o autor que a compartilhou (BLY, 1988).

O termo visualização colaborativa surgiu para denominar a fusão das áreas de visualização e colaboração, que por si só já possuíam desafios próprios, e que agora abrem espaço para novos desafios de pesquisa (ISENBERG et al., 2011). Elas possuem natureza interdisciplinar, formada, por exemplo, de áreas como Computação Distribuída, Interação Humano-computador (IHC) e Trabalho Cooperativo Suportado por Computador (*Computer-Supported Cooperative Work* - CSCW). Neste sentido,

pode-se ainda, categorizá-las por espaço e tempo. Espaço se divide entre espaço distribuído e espaço co-localizado. Já o tempo, se divide em assíncrono e síncrono. Distribuído diz respeito a equipes dispersas geograficamente e co-localizado diz respeito a equipes no mesmo espaço geográfico, isto é, próximos de forma a estabelecer comunicação verbal de forma natural. Assíncrono diz respeito a comunicação que não é feita em tempo real. Já o modo síncrono estabelece envio e recebimento de informações em tempo real.

Em (ISENBERG et al., 2011) são apresentadas 4 definições para o termo visualização colaborativa:

- "Melhora a visualização tradicional, reunindo muitos especialistas, de modo que cada um pode contribuir para o objetivo comum de compreensão do objeto, fenômeno, ou dados sob investigação";
- "Refere-se a um subconjunto de aplicações CSCW em que o controle sobre parâmetros ou produtos do processo de visualização científica são compartilhados";
- "Permite que os usuários geograficamente separados acessem um ambiente compartilhado para visualizar e manipular conjuntos de dados para a resolução de problemas sem ter que se deslocarem fisicamente";
- Análise de dados sociais é o termo designado para classificar interações sociais que podem ocorrer numa visualização colaborativa. Alguns trabalhos tratam visualização colaborativa como análise de dados sociais, então a quarta definição apresentada fala a respeito disso: "Análise de dados social é uma versão de análise exploratória de dados que se baseia na interação social como fonte de inspiração e motivação".

A próxima definição de visualização colaborativa foi elaborada por (ISENBERG et al., 2011), por considerar que as definições encontradas em outros trabalhos não descreviam de forma abrangente o significado da área:

- "visualização colaborativa é o uso compartilhado e interativo de representações visuais de dados por mais de uma pessoa, apoiado por computador, com o objetivo comum de contribuir para as atividades de processamento de informação conjunta".

Esta seção visou apresentar temas fundamentais para o entendimento deste trabalho, como manutenção de software, evolução de software, visualização de

software, colaboração de software e o termo mais geral, colaboração na manutenção, evolução e visualização de software. Tais temas são a base deste trabalho e são úteis na formulação da solução apresentada no capítulo 3.

2.6 TRABALHOS RELACIONADOS

Alguns trabalhos na literatura de manutenção e evolução de software abordam o mesmo problema deste trabalho, que é: a falta de uma solução (técnica, metodologia, ferramenta, *framework,* ou modelo) que apoie a manutenção e a evolução de software, no contexto de equipes geograficamente distribuídas ou co-localizadas, na tarefa de dar manutenção em produtos de software sobre supervisão de uma gerência, que toma iniciativas com base nas atividades realizadas pelas equipes de manutenções. Esta seção apresenta os trabalhos relacionados que foram obtidos através da condução de uma Revisão Sistemática de Literatura (KITCHENAM et al., 2007) que está disponível em (*GIVEME INFRA*, 2014). Uma tabela é apresentada ao final onde é estabelecido um comparativo entre as tecnologias encontradas nos trabalhos relacionados com os principais pré-requisitos necessários para que uma solução possa ser considerada para atuar na resolução do problema geral apresentado neste trabalho.

2.6.1 Soluções Existentes

Syeed et al. (2013) mencionam que dados históricos podem dar indicações sobre o processo de evolução no qual um dado software foi submetido e sobre o conhecimento tácito que foi utilizado durante a evolução, quanto fornecer um histórico sobre a comunicação e a colaboração realizadas pelos desenvolvedores ao longo do ciclo de vida do software. Nesse contexto, foi definido um *framework* para automatizar a análise e a visualização da evolução do software, através de dados históricos de projetos *open sources*. São levados em conta fatores sociais, técnicos e sócio-técnicos. Em relação ao fator social, os autores estão interessados em analisar questões como: de que modo a comunidade que desenvolve projetos *open source* tem mudado ao longo da evolução do software? Existe algum padrão nas mudanças ocorridas com a comunidade de desenvolvedores de software? Sobre o fator técnico eles estão interessados em entender como se dá a evolução de software observando, por exemplo, mudanças no número de linhas de código, número de *commits* realizados, comentários inseridos,

complexidade ciclomática, dentre outros. Já em relação aos fatores sócio-técnicos, o objetivo é analisar, por exemplo, como se deu a comunicação entre membros da equipe de desenvolvimento, historicamente falando, nas atividades de manutenção de software realizadas durante os ciclos de evolução do software.

O *framework* apresentado não é capaz de apoiar, por exemplo, atividades de manutenção colaborativa, fornecendo informações que apoiem na tomada de decisões rumo a, por exemplo, manutenções a serem realizadas quando uma dada alteração no nível de código é conduzida.

Em (LUNGU et al., 2014) é apresentada a ferramenta *Softwarenaut*. Atua na recuperação de arquiteturas de software com base na análise de projetos de código. A ideia central é mostrar, através de visões específicas (como *treemaps*, grafos e estruturas de árvores) as relações existentes entre as entidades presentes no software, isto é, a relação entre módulos, classes e métodos.

Softwarenaut se diferencia da solução proposta neste trabalho em diversos pontos. Em destaque, pode-se citar a questão relacionada à análise de dados históricos objetivando apoiar a tomada de decisões rumo a novas manutenções mediante as indicações de pontos a serem alterados. *Softwarenaut* não tem como foco fornecer tais indicações, e sim apenas mostrar um panorama sobre a evolução do software. Isto de fato pode contribuir para a tomada de decisões rumo a novas manutenções, mas não de forma direta como é proposto na solução deste trabalho que fornecerá informações estatísticas sobre as chances históricas de se ter que alterar, por exemplo, um método, quando outro método for alterado.

Em (ANSLOW et al., 2013) é apresentada a *SourceVis*. Uma ferramenta que fornece visões sobre a evolução de software com base na análise de código fonte. O objetivo é fornecer várias visualizações colaborativas para apoiar o trabalho de equipes distribuídas nas tarefas de entendimento sobre a evolução do software. Tais tarefas podem ser de manutenção mas não é o foco exclusivo e nem principal de *SourceVis* (ANSLOW et al., 2010). Entre as informações que podem ser visualizadas estão métricas de código como número de acoplamentos, linhas de código, número de classes, dentre outras.

A solução proposta neste trabalho deve ser capaz de fornecer visualizações sobre a evolução de software com base na análise de código fonte com suporte à colaboração. Diferentemente de *SourceVis*, além da visão no nível de código, fornecendo métricas e visualizações sobre a relação entre entidades de código como classes e métodos,

também apresentará uma visão sobre dados históricos. Tem como foco principal o apoio a atividades de manutenção, dando indicações de pontos a serem alterados mediante, por exemplo, a necessidade de se corrigir um defeito no sistema.

Em (DAMBROS, et al., 2010]) é apresentado *Churrasco*, um framework para apoiar, de forma distribuída e colaborativa, a análise da evolução de software com base em dados históricos extraídos de três diferentes fontes. São elas: (i) um único sistema gerenciador de *bugs*, o *Bugzilla* (BUGZILLA, 2013); (ii) o histórico de mudanças provenientes de repositórios de código fonte, (SVN e CVS); e (iii) o projeto de código fonte do software que se pretende analisar a evolução. *Churrasco* oferece uma variedade de visualizações. Entre elas, destacam-se: (a) *Correlation View*, capaz de indicar a quantidade de linhas de código por classes, os métodos que pertencem a uma dada classe e o número de defeitos que afetaram uma classe ao longo de sua evolução (b) *Complexity View*, que exibe a complexidade de um *package* (pacote de código fonte Java), medida pela quantidade de linhas de código e o número de métodos contidos.

Churrasco possui semelhanças com a solução proposta para este trabalho, como capacidade de obter automaticamente dados históricos de repositórios de código fonte como SVN e a capacidade de apoiar equipes distribuídas, permitindo a colaboração entre membros, em atividades de compreensão sobre a evolução do software. Entretanto, há lacunas deixadas pela ferramenta, tais como: (i) *Churrasco* não é capaz de manipular dados históricos provenientes de diferentes tipos de repositórios de solicitações de mudança, tais como *Redmine* (REDMINE 2013) e *Mantis Bugtracker* (MANTIS, 2013) ou repositórios customizados, como o *Service Center*, desenvolvido pela Empresa Parceira 1 e apresentado no capítulo de introdução deste trabalho; (ii) não dá indicações diretas e estatísticas sobre pontos que podem ser impactados, dada uma solicitação de mudança.

Em (TELEA et al., 2005) é apresentada uma técnica que permite a construção de mecanismos para explorar a evolução de projetos de código fonte disponíveis em bases de dados de código fonte CVS. Os autores apresentam o conceito de *Visual Code Navigator* (VCN), no qual são definidos meios com os quais é possível navegar, de forma visual entre artefatos de código. Este conceito é utilizado na técnica apresentada em forma de visões. Dentre elas estão: *Syntactic View* e *Evolution View*. *Syntactic View* que permite que o usuário visualize em miniatura, um conjunto selecionado de arquivos de código fonte disponíveis em um projeto. Por exemplo, pode-se selecionar 5 classes

pertencentes a um software e analisá-las, ao mesmo tempo (por ficarem pareadas e em formato reduzido) permitindo analisar parte das classes (seus métodos e atributos) apenas utilizando o recurso de *zoom*. *Evolution View* fornece uma linha do tempo para acompanhar as mudanças sofridas em uma classe ao longo das versões de um software, historicamente. Através dela, é possível visualizar, por exemplo, como uma dada classe evoluiu, em termos de número de linhas de código, ao longo das versões do software.

A principal diferença entre a técnica apresentada em (TELEA et al., 2005) e a solução proposta para este trabalho está na forma como a evolução de um dado software pode ser acompanhada. A solução proposta para esse trabalho deve processar as informações e as disponibilizar em visualizações específicas para cada informação processada. Já a técnica anterior necessita que sejam construídas consultas (*queries*) seguindo uma representação própria (que se assemelha a funções matemáticas) para que uma dada informação seja disponibilizada em uma visualização existente. Além disso, a técnica não prevê a comunicação entre equipes distribuídas, fundamental para a resolução do problema geral deste trabalho.

Em (MAO, 2011) é apresentado um *framework* para análise de arquivos de código fonte provenientes de repositórios. Tem como objetivo fornecer ao usuário do *framework* visualizações sobre as dependências existentes entre as entidades de código presentes nos repositórios de código fonte, como classes. A ideia principal é fornecer uma arquitetura de *framework* que possa ser evoluída de acordo com a necessidade do usuário, por exemplo, implementando suporte a repositórios de código fonte de preferência. Nesse trabalho é apresentado um mecanismo que permite que os arquivos de código fonte obtidos sejam transformados em um formato específico para que possam ser interpretados pelo *framework.* Após a interpretação, torna-se possível visualizar as dependências, bem com um conjunto de métricas calculadas. Um exemplo delas é a *Network Diameter*, que exibir o cálculo da distância entre entidades de código, como classes. Para entender melhor essa métrica é preciso entender o princípio utilizado pelo *framework*, denominado rede de dependência. Uma rede de dependência é formada por grafos desconectados ou não, representando, por exemplo, pacotes de código com suas classes. Um pacote seria representado por círculo, tendo as classes contidas no pacote ligadas entre si, se houver acoplamento entre elas. *Network Diameter* permite calcular o caminho entre duas classes, estando elas em um mesmo pacote ou não. Esta métrica pode oferecer algum indício de pontos que podem ser impactados quando uma

dada classe é alterada, mas existem alguns fatores que devem ser considerados, como por exemplo: (i) não é o foco do *framework* apoiar atividades de manutenção colaborativa; (ii) a visualização *Dependence Network* apenas exibe a relação entre entidades como módulos (pacotes de código fonte) e classes, não apoiando a exibição de relações entre entidades como métodos de uma classe, e (iii) as análises realizadas consideram apenas uma única versão do código fonte, não estabelecendo um comparativo baseado na evolução do software.

Em (TELEA et al., 2008) assume-se que código fonte, versionado em sistemas de controle de versão como SVN e CVS, pode oferecer indícios sobre a evolução de software, pois armazenam alterações realizadas durante todo o ciclo de vida de um software. Com base nisto, foi desenvolvido um método, chamado *Code Flows*, para auxiliar na tarefa de analisar a evolução de um software em um contexto onde são observadas mudanças a que um projeto de código foi submetido ao longo do tempo.

Code Flows oferece um conjunto de visualizações para explorar os dados sobre a evolução de software. Em uma delas é possível analisar o contexto histórico de um bloco de código (seja ele um método, uma classe, um conjunto de linhas ou uma única linha de código). Na prática, isto quer dizer que *Code Flows* é capaz de indicar que um bloco de código que estava em uma classe A na versão X de um software, agora, na versão X + 1 encontra-se na classe B ou em outro ponto dentro da mesma classe A. Em outra visualização fornecida é possível rastrear o mesmo bloco de código e seu destino final, mas com uma representação visual em formato de árvore, através da qual, por exemplo, um método A1 que pertencia a classe A (classe A então é representada como sendo um nó pai na árvore e o método A1 como sendo um nó filho de A) em uma outra versão do software analisado passou a pertencer a classe B (portanto agora A1 é nó filho de B).

Assim com outras ferramentas já apresentadas neste Capítulo, *Code Flows* atua fornecendo meios para que se possa analisar a evolução de software olhando exclusivamente para o contexto de código fonte, mas não é capaz de permitir análises sobre a evolução de software no contexto de dados históricos de solicitações de mudanças.

Em (VOIGT et al., 2009) é apresentada uma técnica desenvolvida para apoiar, por exemplo, em atividades de manutenção, a descoberta da rastreabilidade entre

entidades de código fonte orientado a objetos, como métodos de uma classe. Neste sentido, se está interessado em analisar o grau de dependência entre as entidades objetivando aprender sobre os acoplamentos existentes. A técnica desenvolvida acompanha visualizações que permitem explorar graficamente a rastreabilidade descoberta. Um exemplo é a matriz de visualização que exibe a rastreabilidade calculada para um conjunto de entidades de código.

A técnica apresentada em (VOIGT et al., 2009) é capaz de fornecer informações sobre acoplamentos existentes entre métodos de uma classe para apoiar atividades de manutenção. Em contrapartida, a técnica não contempla atividades de evolução, realizadas de forma distribuídas com a colaboração de membros geograficamente distribuídos o que é fundamental na resolução do problema geral apresentado neste trabalho.

Em (KEVIC et al., 2014) é apresentada uma técnica que permite a análise de defeitos cadastrados em um sistema de *bug tracking* objetivando encontrar similaridades entre eles. Ela foi desenvolvida para ser utilizada de forma colaborativa, através da qual desenvolvedores serão capazes de encontrar similaridades entre defeitos e assim conseguirem resolver um novo defeito. Primeiramente, um novo defeito é cadastrado e fica aguardando até que possa ser resolvido. A partir desse momento, um conjunto de desenvolvedores, através de uma ferramenta de software, irão analisar outros defeitos que já tenham sido resolvidos anteriormente e selecionar um ou mais defeitos que julgarem semelhantes ao defeito a ser resolvido. Em seguida, um recurso de software é acionado tornando-se possível observar as mudanças ocorridas, no nível de código, nos defeitos selecionados, indicando que, o defeito a ser resolvido poderá impactar aqueles mesmos trechos de código, dada a similaridade entre eles.

A técnica apresentada em (KEVIC et al., 2014) atua considerando dados históricos de defeitos e dados históricos de alterações no nível de código fonte. Possui ainda a capacidade de permitir que equipes de manutenção trabalhem de forma colaborativa em atividade de manutenção e evolução de software.

As principais diferenças entre a solução proposta para este trabalho e a técnica apresentada em (KEVIC et al., 2014) estão:

- na forma como indicações de possíveis alterações para a resolução de um *bug* são fornecidas: a técnica apresentada necessita de um conjunto de

desenvolvedores atuando de forma colaborativa e utilizando conhecimento tácito para descobrir quais *bugs* são semelhantes ao *bug* a ser resolvido. Essa tarefa é feita analisando apenas os dados cadastrais dos *bugs*, como tipo, data de abertura e principalmente descrição textual do *bug* encontrado. A solução proposta para este trabalho irá calcular as indicações automaticamente e necessitará de apenas um desenvolvedor para dar início na execução do processo.

- nos tipos de solicitações de mudanças abrangidas: a técnica em (KEVIC et al., 2014) apresenta os possíveis pontos no código fonte que podem ser impactados apenas quando a atividade de manutenção a ser realizada corresponde a uma solicitação de mudança do tipo corretiva (para correção de defeitos) visto que apenas dados de *bugs* são analisados, não contemplando outros tipos de solicitações de mudança como corretivas, adaptativas e evolutivas;
- no formato das indicações: a técnica em (KEVIC et al., 2014) fornece indicações de possíveis pontos a serem alterados no código fonte, para a resolução de um *bug*, exibindo trechos de código fonte alterados (visão fornecida por *diffs* (SVN BOOK, 2015) nos defeitos semelhantes, mas não fornece uma lista de métodos, classes e módulos a serem alterados, diretamente, deixando a cargo da equipe de manutenção a tarefa de verificar manualmente quais métodos e classes podem ser impactados.

Em (SHRESTHA et al., 2013) é apresentada a *Storygraph*. Uma ferramenta visual que analisa dados históricos de software, como *logs* provenientes de repositórios de código fonte, objetivando fornecer visualmente informações que podem ser usadas na gerência de projetos desenvolvidos colaborativamente. Fornece visualizações onde é possível que a gerência do projeto acompanhe quem mais está colaborando na equipe geograficamente distribuída. Ela fornece um gráfico com as coordenadas geográficas de cada membro da equipe e exibe em um mapa as regiões que mais colaboraram.

A solução apresentada em (SHRESTHA et al., 2013) de fato é capaz de analisar dados da evolução de software e apoiar a gerência de projetos mantidos de forma distribuída, mas não com o objetivo de apoiar atividades de manutenção. O foco está somente na gerência de projetos mantidos de forma distribuída. Aliás, antes de permitir a gerência de projetos mantidos de forma distribuída, é necessário que uma solução, para resolver o problema geral deste trabalho, apoie o compartilhamento de informações entre membros da equipe distribuída (isto é, colaboração através do envio/recebimento

de mensagens), já que essa interação é necessária para que as atividades de manutenção possam ser realizadas.

2.6.2 Comparativo entre Soluções Existentes

Nesta seção é estabelecido um paralelo (Tabela 1) entre as soluções existentes apresentadas nos trabalhos relacionados com um conjunto de pré requisitos que uma tecnologia precisa apresentar para resolver o problema geral apresentado neste trabalho. Os pré requisitos são:

1. apoiar atividades de manutenção de software em equipes distribuídas;
2. permitir a colaboração entre membros de uma equipe geograficamente distribuída;
3. ser capaz de analisar dados históricos sobre a evolução de software, provenientes de repositórios de solicitações de mudanças e de repositórios de código fonte;
4. fornecer visualizações sobre a evolução de software;
5. fornecer indicações sobre quais métodos de uma classe podem ser impactados quando um dado método for alterado;

Além da lista de pré-requisitos, serão utilizados indicadores que mostram o grau de aderência a cada um dos pré-requisitos:

- TOTAL: indica que a tecnologia apoia por completo a resolução do problema geral deste trabalho no pré-requisito em questão;
- PARCIAL: indica que a tecnologia apoia parcialmente a resolução do problema geral deste trabalho no pré-requisito em questão;
- X: indica que a tecnologia não contempla o pré-requisito em questão;

Para ser considerada uma solução válida para o problema geral deste trabalho, a mesma deve possuir somente indicador TOTAL em todos os pré-requisitos definidos.

Tabela 1: Comparativo entre as tecnologias dos trabalhos relacionados com os pré-requisitos definidos.

Tecnologia apresentada em	Pré requisitos				
	1	2	3	4	5
(SYEED et al., 2013)	X	X	PARCIAL	TOTAL	X
(LUNGU et al., 2014)	X	X	X	TOTAL	PARCIAL
(ANSLOW et al., 2013) e (ANSLOW et al., 2010)	PARCIAL	PARCIAL	PARCIAL	PARCIAL	X
(DAMBROS et al., 2010)	X	TOTAL	PARCIAL	PARCIAL	X
(TELEA et al., 2005)	X	X	PARCIAL	PARCIAL	X
(MAO, 2011)	X	X	PARCIAL	PARCIAL	X
(TELEA et al., 2008)	X	X	PARCIAL	PARCIAL	X
(VOIGT et al., 2009)	PARCIAL	X	PARCIAL	PARCIAL	PARCIAL
(KEVIC et al., 2014)	PARCIAL	PARCIAL	PARCIAL	PARCIAL	PARCIAL
(Shrestha et al., 2013)	X	PARCIAL	PARCIAL	X	X

2.7 CONSIDERAÇÕES FINAIS SOBRE O CAPÍTULO

Este capítulo apresentou uma discussão de temas que são a base deste trabalho, tais como: manutenção de software, evolução de software, colaboração e visualização de software. Além de abordar esses temas, foi fornecida uma visão geral das relações existentes entre eles, dado que o problema geral tratado neste trabalho tem como resolução o desenvolvimento de uma solução que combina os temas citados. Foram apresentadas as definições de manutenção de software e os tipos de manutenção. Também foram apresentadas informações conceituais sobre evolução de software e sobre aspectos de colaboração que fazem parte de atividades de manutenção e evolução realizadas de forma distribuída.

3. *GIVEME INFRA*: UMA INFRAESTRUTURA BASEADA EM MÚLTIPLAS VISÕES PARA A EVOLUÇÃO DISTRIBUÍDA DE SOFTWARE

Como dito na introdução deste trabalho, o problema encontrado pela consultoria realizada na Empresa Parceira 1 deu origem ao problema geral deste trabalho que é: a falta de uma solução (técnica, metodologia, ferramenta, *framework*, ou modelo) que apoie a manutenção e a evolução de software, no contexto de equipes geograficamente distribuídas ou co-localizadas, na tarefa de dar manutenção em produtos de software sobre supervisão de uma gerência, que toma iniciativas com base nas atividades realizadas pelas equipes de manutenções.

GiveMe Infra é a solução desenvolvida com base no problema geral apresentado. Trata-se de uma infraestrutura para apoio a realização de atividades de manutenção e evolução de software, realizadas por equipes co-localizadas ou geograficamente distribuídas. Tais atividades são apoiadas por diferentes visualizações de software que permitem ao usuário obter diferentes perspectivas sobre as informações disponibilizadas.

Com base no objetivo geral deste trabalho e no cenário encontrado na Empresa Parceira 1, foram definidos requisitos funcionais que são contemplados pela *GiveMe Infra.* A infra estrutura deve permitir:

1. manter solicitações de mudanças;
2. associar informações de rastreabilidade à solicitação de
3. Mudança;
4. manter atividades de manutenção e evolução de software;
5. apoiar tomada de decisão sobre alterações em nível de código;
6. analisar o impacto de alterações em outros
7. módulos/componentes;
8. manter equipes de manutenção;
9. manter dados históricos de projetos;
10. potencializar as atividades de compreensão, manutenção e evolução de software;

Como requisito não funcional tem-se:

- Integração com ambiente de compreensão e colaboração.

Os requisitos funcionais de número 2, 4, 5 e 7 são provenientes da integração com ambientes de compreensão e colaboração de software, já os demais, desenvolvidos contemplados pelas ferramentas desenvolvidas neste trabalho.

GiveMe Infra atua de forma integrada ao ambiente de desenvolvimento *Eclipse*. Constituída de diferentes *plugins*, é capaz de atuar desde o cadastramento da solicitação de mudança até a entrega da versão que contempla a mudança solicitada. Ela ainda suporta atividades de gerenciamento e execução da manutenção, pois conta com recursos que dão indícios do que deve ser alterado no código fonte, de forma direta, diferentemente de outras tecnologias que visam apenas relacionar manutenções realizadas com uma dada manutenção a ser efetuada. A Figura 3.1 fornece uma visão geral das principais linhas de atuação da *GiveMe Infra*, separadas por duas categorias, sendo uma delas composta por recursos disponíveis graças a integração com ambiente de compreensão e colaboração. Os demais foram desenvolvidos neste trabalho.

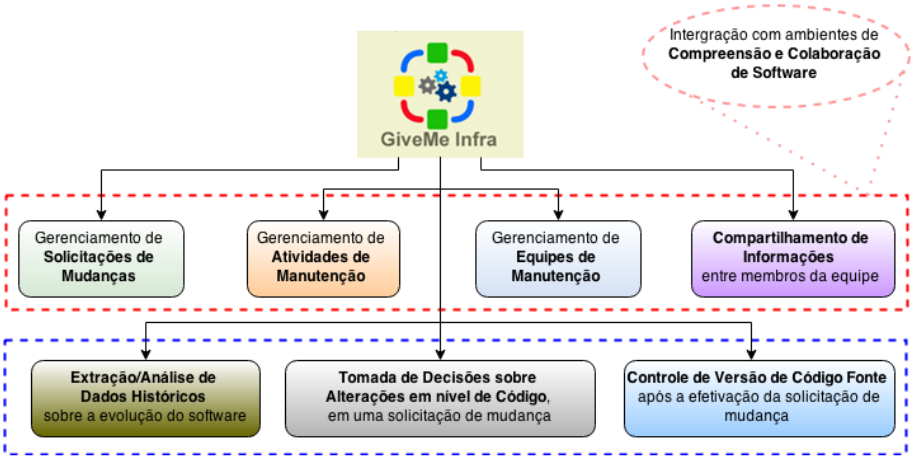

Figura 3.1: Visão geral das principais linhas de atuação da *GiveMe Infra*.

Para suportar as linhas de atuação apresentadas na Figura 3.1, foi desenvolvido um conjunto de ferramentas e recursos. A seção 3.1 apresenta todas as ferramentas que foram desenvolvidas neste trabalho, sendo uma delas conceitual e uma ferramenta piloto. Todas as demais trabalham de forma integrada com o ambiente *Eclipse*, compondo a *GiveMe Infra*.

A seção 3.2 apresenta os recursos disponíveis. Neste sentido, serão apresentadas as funcionalidades disponíveis bem como cenários onde podem ser utilizadas.

A seção 3.3 apresenta o processo de desenvolvimento da solução, que contempla desde as ferramentas implementadas e as integrações realizadas entre elas até as parcerias estabelecidas que viabilizaram este trabalho. Por último, são apresentadas as considerações finais na seção 3.4.

3.1 FERRAMENTAS DESENVOLVIDAS

Inicialmente nesta seção são apresentadas algumas convenções adotadas para este trabalho, que se referem a *GiveMe Infra* (subseção 3.1.1). Em subseções seguintes são apresentadas a ferramenta conceitual desenvolvida, (o *framework GiveMe Metrics* - subseção 3.1.2) e a ferramenta piloto *Analisador* (subseção 3.1.3). As demais subseções apresentam as ferramentas que compõem a *GiveMe Infra.*

3.1.1 Convenções adotadas

A primeira convenção adotada refere-se a dois diferentes contextos em que uma ferramenta da *GiveMe Infra* pode estar associada:

I. Contexto Atual: essa é a classificação dada ao contexto em que as ferramentas estiverem analisando o projeto de código de um software disponível no espaço de trabalho do ambiente *Eclipse* (*workspace*). Quando uma ferramenta está habilitada nesse contexto, automaticamente passará a analisar a versão atual de um software, fornecendo ao usuário informações extraídas do projeto de código fonte, como acoplamento entre classes e número de linhas de código. Um exemplo de ferramenta que trabalha neste contexto é a *Sourceminer* (2011);
II. Contexto Histórico: classificação dada ao contexto em que ferramentas estiverem analisando dados históricos de um projeto, como dados provenientes de registros de solicitação de mudanças e arquivos de *log* de repositórios de código fonte. Nesse contexto, se está interessado em analisar informações obtidas ao longo de todo o ciclo de manutenção de um software e não somente de um período específico ou de uma versão atual, como no Contexto Atual.

Além de pertencer a um dos contextos citados, uma ferramenta que pertença à *GiveMe Infra* pode disponibilizar recursos de colaboração. Durante todo o texto, o termo Recursos de Colaboração será utilizado para dizer que uma dada ferramenta é capaz de:

- enviar e receber mensagens de usuário tipo conversação (*chat*);
- enviar e receber mensagens de usuário sobre uma entidade. Uma entidade pode ser uma classe, um módulo ou um método de um sistema. Esse recurso pode ser usado para trocas de informações técnicas e de impressões obtidas através da análise das entidades;
- criar e visualizar mensagens de sistema: são mensagens que podem ser geradas automaticamente quando uma ação na ferramenta é executada. Um exemplo são mensagens de sistema cadastradas automaticamente quando um membro da equipe efetua *login* na ferramenta.

Todas as ferramentas que compõem a *GiveMe Infra*, ou foram desenvolvidas neste trabalho, ou pertencem a terceiros e, através de parcerias estabelecidas, modificadas e integradas à infraestrutura. Dentre as parcerias realizadas com outras equipes, duas em especial disponibilizaram ferramentas que foram modificadas e integradas ao *GiveMe Infra*. São elas:

- equipe *Sourceminer* + AIMV: equipe representada pelo professor Glauco Carneiro da Universidade Salvador, e atualmente ligado ao Grupo de Pesquisa de Engenharia de Software e Aplicações;
- equipe *Collaborative Sourceminer*: equipe representada pelo doutorando Carlos Fábio Conceição, em Ciências e Tecnologia da Informação pelo Instituto Universitário de Lisboa, Portugal.

Uma atividade de manutenção, no contexto da *GiveMe Infra*, refere-se a atividades realizadas por equipes co-localizadas ou distribuídas, visando efetuar uma solicitação de mudança que irá gerar alterações no projeto de código de um software. A comunicação entre membros da equipe deve contar com recursos de colaboração, como envio de mensagens de conversação e de colaboração. A gerência, por sua vez, deve ser apoiada por recursos que permitam o acompanhamento do que está sendo efetuado por cada membro da equipe.

Outra importante informação sobre uma atividade de manutenção é que ela pertence aos dois contextos: atual e histórico. Isso, na prática, quer dizer que, durante sua execução, o membro da equipe pode trocar de contexto, obtendo com isso diferentes perspectivas sobre o projeto em manutenção.

3.1.2 O framework GiveMe Metrics

GiveMe Metrics (TAVARES et al., 2014) é uma ferramenta conceitual (*framework*) criada no início deste trabalho, quando o problema da Empresa Parceira 1 foi identificado. Surgiu da necessidade de se extrair métricas e dados históricos de código fonte de repositórios como o da Empresa Parceira 1. Um *framework* pode ser representado como um conjunto de classes abstratas e pela forma como suas instâncias interagem (JOHNSON, 1997). A Figura 3.2 mostra o *framework*.

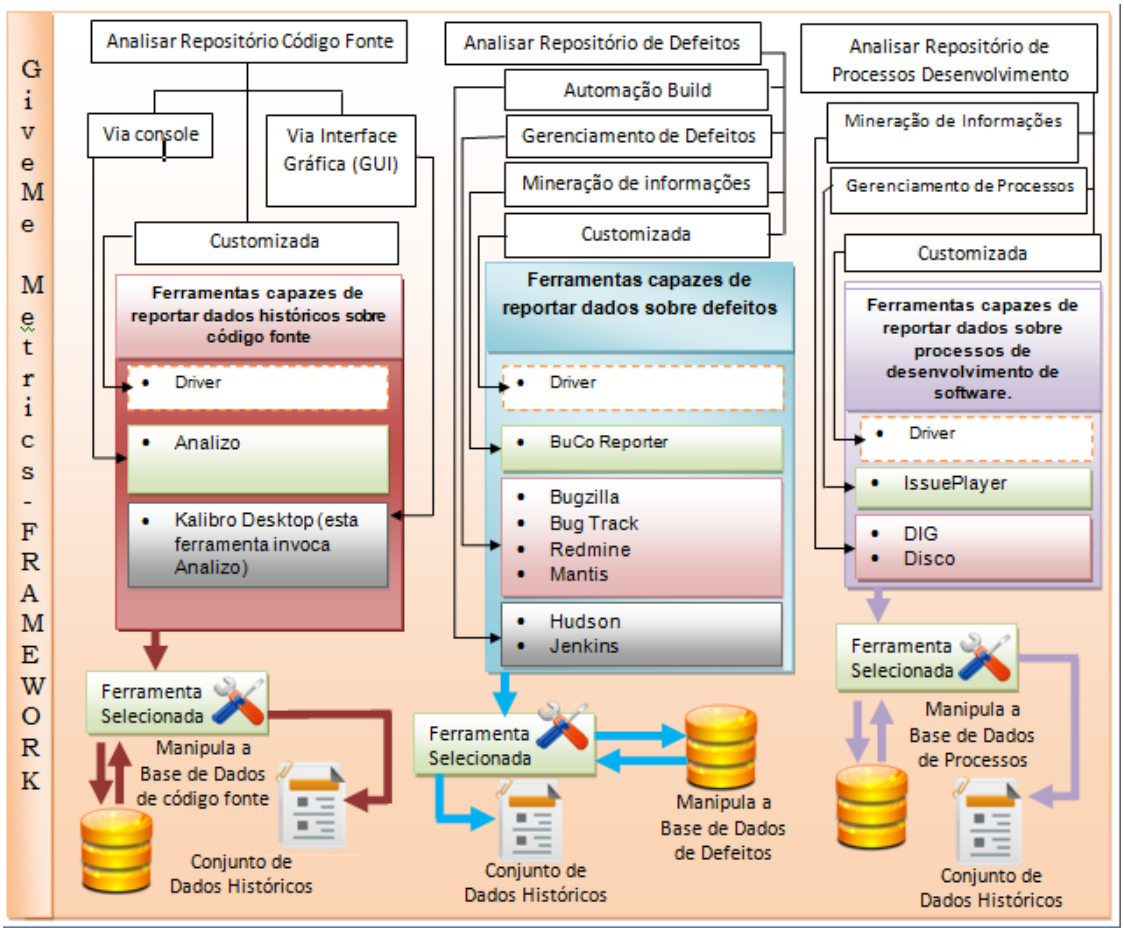

Figura 3.2: *GiveMe Metrics Framework*

Em (TAVARES et al., 2014) foram definidos diferentes cenários de utilização para o *framework*, bem como exemplos práticos da condução de extrações de conjuntos de dados históricos. No contexto deste trabalho, *GiveMe Metrics* proporcionou que dados históricos sobre solicitações de mudanças mantidas no sistema de gerenciamento de solicitações de mudanças *Service Center* da Empresa Parceira 1 (como apresentado no capítulo de Introdução) fossem extraídos.

Todos os conjuntos de dados históricos extraídos pelo *GiveMe Metrics* ficam disponíveis para serem manipulados pela *GiveMe Infra* em um repositório de dados históricos chamado *GiveMe Repository* (subseção 3.1.7).

3.1.3 Analisador

Trata-se de uma ferramenta piloto desenvolvida neste trabalho para implementar o *Statistical Analysis Engine* (SAE) ou Motor de Análise Estatística, parte fundamental e inicial na concepção da solução proposta para este trabalho. A seguir é apresentado o SAE, juntamente com os passos dados rumo à sua definição.

3.1.3.1 SAE - *Statistical Analysis Engine*

Após a definição do problema encontrado na Empresa Parceira 1, como relatado no capítulo de Introdução, foi conduzida a extração dos dados históricos. Posteriormente, foi conduzida uma análise onde foi possível observar que os dados referentes a uma solicitação de mudança continham informações de rastreabilidade, como componentes e módulos alterados. Foi observado também que um dado componente do software era sempre alterado quando outro dado componente também era alterado. Isso pode ser um indício de que exista um acoplamento entre tais componentes de forma que alterações em um impactem diretamente no outro (LÉLIS, 2014). Uma questão foi levantada: historicamente falando, qual o percentual em que um dado componente era modificado sempre que um outro componente de escolha fosse também modificado? Para responder tal questão foram consideradas teorias estatísticas de Distribuição de Frequência Simples e Conjunta (MEYER, 1983). Com isso, foi possível implementar um motor (SAE), junto ao Departamento de Estatística da Universidade Federal de Juiz de Fora, que permite responder a seguinte questão: em

termos de porcentagem, quais as chances de um dado componente de um módulo, que será alterado, impactar em cada um dos outros componentes e módulos de um software e até mesmo em outros softwares integrados a ele? A Figura 3.3 ilustra a definição do SAE.

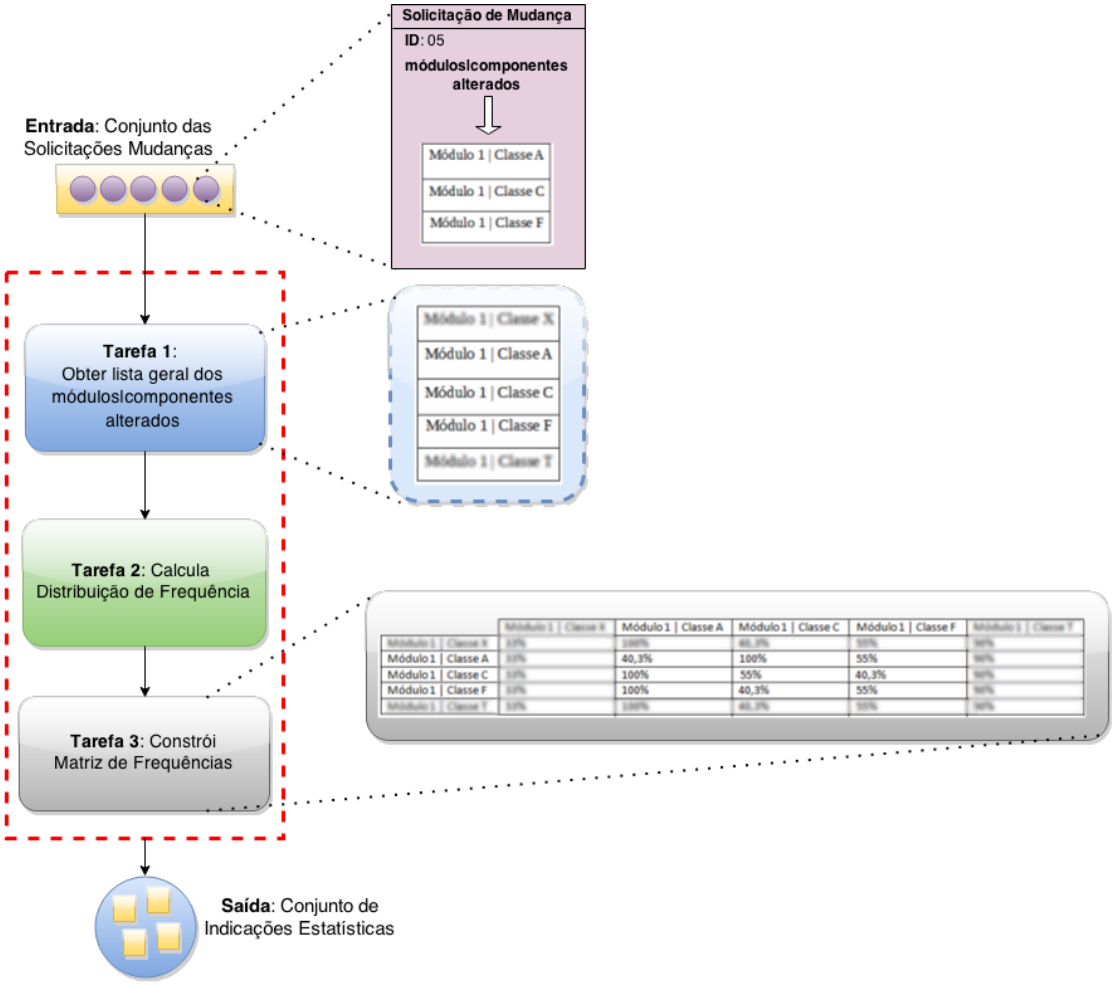

Figura 3.3: Sequência de tarefas que compõem SAE.

Os módulos da Figura 3.3 que estão dentro da linha pontilhada representam o SAE. Como entrada, tem-se um conjunto de solicitações de mudanças. Cada solicitação é única, e possui um conjunto de informações, como classes e módulos alterados para resolução daquela solicitação de mudança, como pode ser visto no *zoom* aplicado na solicitação de mudança de ID 05. A primeira tarefa dentro do SAE é obter a lista geral dos módulos e componentes alterados, sem repetições entre todos os conjuntos de informações. Em seguida, são executados os cálculos estatísticos (Tarefa 2) e construída

a matriz que possui as frequências calculadas. O *zoom* na Tarefa 3 mostra a matriz com as estatísticas obtidas. Como saída do SAE, tem-se um conjunto de estatísticas sobre qualquer método de um software, onde é possível avaliar o impacto em qualquer ponto do software.

3.1.4 Sourceminer

Trata-se de um *plugin* para o ambiente *Eclipse*, proposto por (CARNEIRO, 2011) para apoiar atividades de compreensão de software, tanto no desenvolvimento quanto na manutenção.

Sourceminer trabalha realizando análises de projetos de código fonte disponíveis no espaço de trabalho do *Eclipse* (*workspace*), e gerando informações relevantes sobre o projeto, como métricas de código, que auxiliam na compreensão.

No contexto deste trabalho, *Sourceminer* permite que as atividades de manutenção a serem realizadas contêm duas diferentes formas de compreensão do que deverá ser modificado, tais como: (i) a compreensão baseada na análise do Contexto Atual, fornecida pelo processamento do projeto de código, e (ii) a compreensão baseada na análise do Contexto Histórico.

3.1.5 *Toolkit* AIMV

É um *toolkit* formado por vários *plugins* para o ambiente *Eclipse*, criado por (SILVA, 2013), para apoiar construção de ambientes interativos de múltiplas visões. AIMV trabalha de forma integrada com o *Sourceminer* proporcionando ao usuário realizar atividades de compreensão de software.

Neste trabalho, AIMV é responsável por fornecer visualizações que são carregadas com informações, ora do Contexto Atual, ora do Contexto Histórico, aumentando a gama de visualizações disponíveis na *GiveMe Infra*.

3.1.6 *GiveMe Views*: uma ferramenta de suporte a evolução de software baseada na análise de dados históricos.

Trata-se de um *plugin* para o ambiente *Eclipse* desenvolvido neste trabalho, com dois objetivos em específico:

- realizar o processamento estatístico dos dados históricos: o SAE, que foi implementado na ferramenta piloto *Analisador*, agora ganha sua versão final implementado no módulo Análises Estatísticas;
- persistir as informações: todas as informações originadas do processamento estatístico ficam armazenadas no *GiveMe Views* através de um módulo de persistência, podendo serem acessadas por visualizações ou por outras ferramentas.

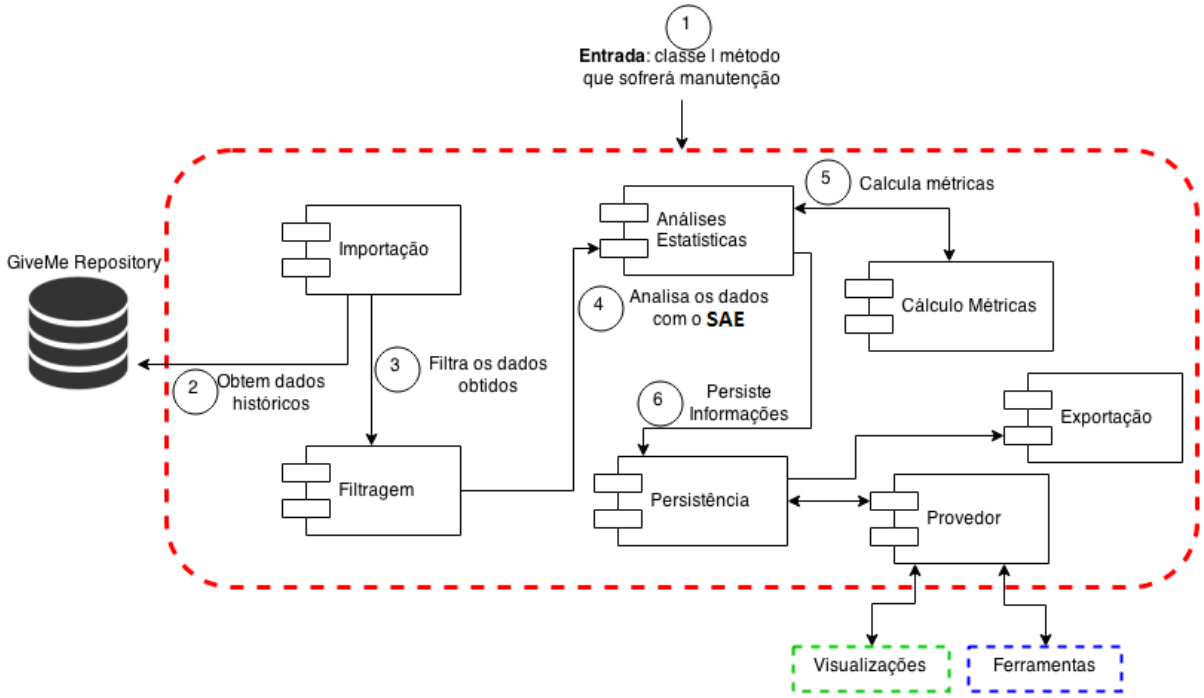

Figura 3.4: Relação entre módulos do GiveMe Views

Considera-se a Figura 3.4, que mostra a relação entre módulos do *GiveMe Views*, contidos na área pontilhada. São eles:

- Importação: responsável por selecionar no repositório, o conjunto de dados históricos referentes ao projeto em manutenção;

- Filtragem: realiza a filtragem dos dados importados, desconsiderando, por exemplo, dados de solicitações de mudanças que não foram concluídas e, portanto, não afetaram nenhum trecho de código do projeto em manutenção, tornando-se irrelevantes no processamento estatístico;
- Análises Estatísticas: responsável por implementar o SAE. Além disso, é responsável por formatar as informações nos formatos necessários para os diferentes tipos de visualização da *GiveMe Infra;*
- Cálculo Métricas: calcula todas as métricas sobre dados históricos que são disponíveis na *GiveMe Infra.* A lista completa das métricas será apresentada na seção Recursos;
- Persistência: módulo responsável por manter em memória (solução que utiliza *singleton*) todas as informações estatísticas processadas, bem como as métricas e as informações de sistema, como usuário autenticado e dados de sessão;
- Provedor: único ponto pelo qual visualizações e ferramentas integradas à *GiveMe Infra* podem obter informações sobre os dados históricos processados e mantidos em memória;
- Exportação: módulo responsável por implementar os relatórios disponíveis para exportação na *GiveMe Infra* no formato de planilhas do *Microsoft Excel.*

Uma entrada para a ferramenta *GiveMe Views* é a seleção de um método que se deseja dar manutenção. Em seguida, os dados históricos referentes ao projeto alvo da manutenção são importados do repositório e filtrados, para posteriormente serem analisados estatisticamente, bem como calcular as métricas disponíveis. Toda a informação gerada é então persistida, ficando disponível para exportação ou para acesso de visualizações e ferramentas, via provedor. Mais detalhes sobre a ferramenta são apresentados na subseção Recursos.

Relação *drivers* e dados históricos de software: um importante conceito implementado no *GiveMe Views* é o de *driver.* São implementações que permitem a manipulação dos dados históricos que estão no repositório, independente do formato, que está restrito aos suportados pelo *GiveMe Metrics*: *.xml*, *.txt*, *.html*, *.csv* e *.xls*. Um conjunto de dados históricos, em termos práticos, pode ser formado, por exemplo, por um arquivo *.csv* e um *.txt*, ou somente um *.xls*.

Outra informação importante sobre um *driver* é que ele é implementado para ler conjunto de dados exportados por cada uma das ferramentas que compõem o *framework*

GiveMe Metrics. Na versão atual do *GiveMe Views*, há suporte para leitura de dados exportados pela ferramenta *Mantis* e dados exportados do repositório customizado da Empresa Parceira 1, mas é importante destacar que eles podem ser desenvolvidos para suportar a leitura de diferentes fontes de dados, e não somente as previstas no *GiveMe Metrics.*

3.1.7 GiveMe Repository

É um *plugin* desenvolvido neste trabalho para automatizar a construção do repositório de dados históricos da *GiveMe Infra*, chamado *GiveMe Repository*. A função desse *plugin* é eliminar a responsabilidade do usuário na criação de toda a estrutura de pastas pertencente ao repositório e por implementar toda a lógica de manipulação do *GiveMe Repository*. A Figura 3.5 ilustra a estrutura do repositório *GiveMe Repository* gerada pelo *plugin.*

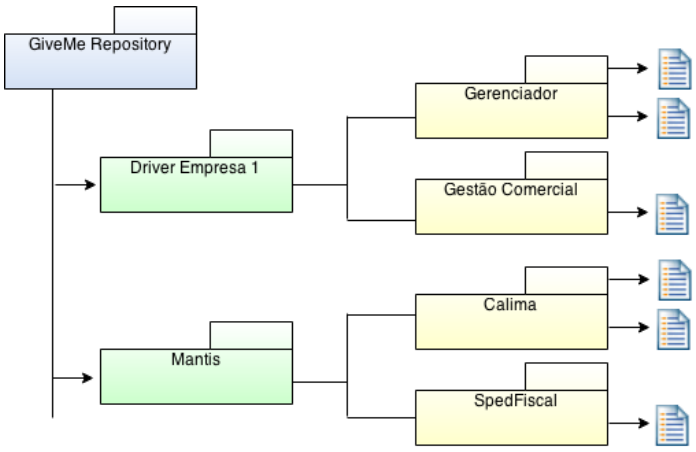

Figura 3.5: Estrutura do repositório gerado pelo plugin GiveMe Repository

A Figura 3.5 ressalta que há espaço para armazenamento de dados para dois *drivers* diferentes: o *Driver* Empresa 1 e o *driver* desenvolvido para a ferramenta *Mantis.* Dentro de cada espaço, há outros espaços que são específicos para cada projeto cadastrado na *GiveMe Infra.* No espaço para o Driver Empresa 1 há dados históricos de dois projetos: Gerenciador e Gestão Comercial. No espaço *Mantis* há dados históricos de outros dois projetos: Calima e *SpedFiscal*. Todos os quatro projetos mencionados são reais e foram disponibilizados através de parcerias com empresas.

Para gerar a estrutura do *GiveMe Repository*, o usuário da *GiveMe Infra* deve apenas clicar no botão indicado na Figura 3.6 e informar o caminho físico de sua escolha para que o *GiveMe Repository* seja criado. A partir de então, todas as vezes que alguma ferramenta da *GiveMe Infra* necessitar analisar um conjunto de dados históricos pertencentes a um projeto, saberá onde o repositório foi criado e o acessará.

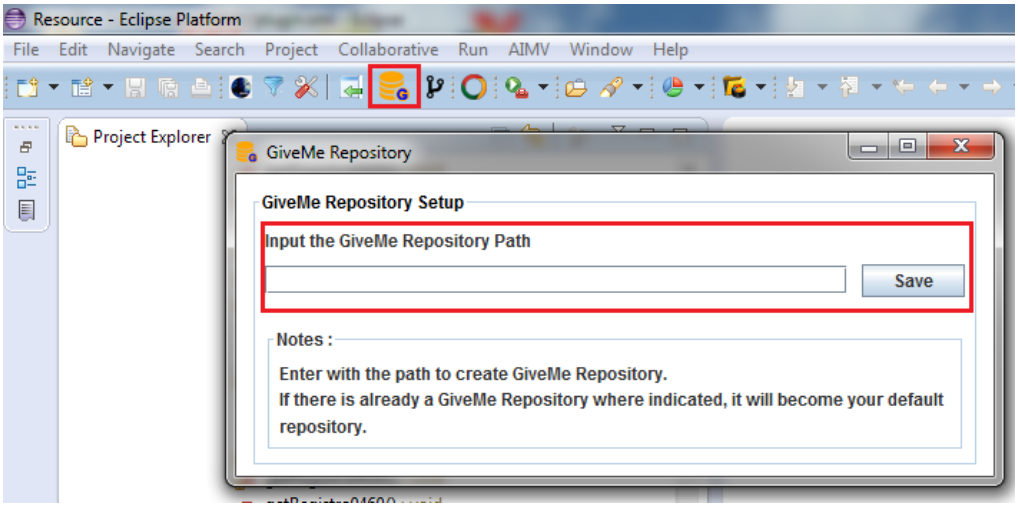

Figura 3.6: Definindo o caminho para a criação do GiveMe Repository

3.1.8 Collaborative Sourceminer

É o *plugin* para o *Collaborative Sourceminer* (CONCEIÇÃO, 2012). Através dele é possível analisar todas as informações geradas pelo *Sourceminer* + AIMV e, através dos recursos de colaboração implementados, permitir a equipes distribuídas a inserção de mensagens de colaboração diretamente nas visualizações, permitindo a interação diretamente nas visualizações.

A integração do *Collaborative Sourceminer* à *GiveMe Infra* permitiu que todos os recursos de colaboração disponíveis para o Contexto Atual, também pudessem ser utilizados no Contexto Histórico. A lista completa de todas as funcionalidades que o *Collaborative Sourceminer* possui e que estão integrados a *GiveMe Infra* será apresentada na seção 3.2.

3.1.9 *Collaboration Provider*

Trata-se de um *plugin* desenvolvido neste trabalho para encapsular toda a lógica de acesso e manipulação do *web service* do *Collaborative Sourceminer*, responsável por fornecer os recursos de colaboração. Seu desenvolvimento permitiu que toda a lógica de implementação dos recursos de colaboração ficasse concentrada em um único módulo (*provider*), que por sua vez acessa código de acesso/manipulação ao *web service* disponível em outro módulo (*core*). A Figura 3.7 ilustra a relação de acesso aos recursos de colaboração por parte dos demais *plugins* que os consomem na *GiveMe Infra*.

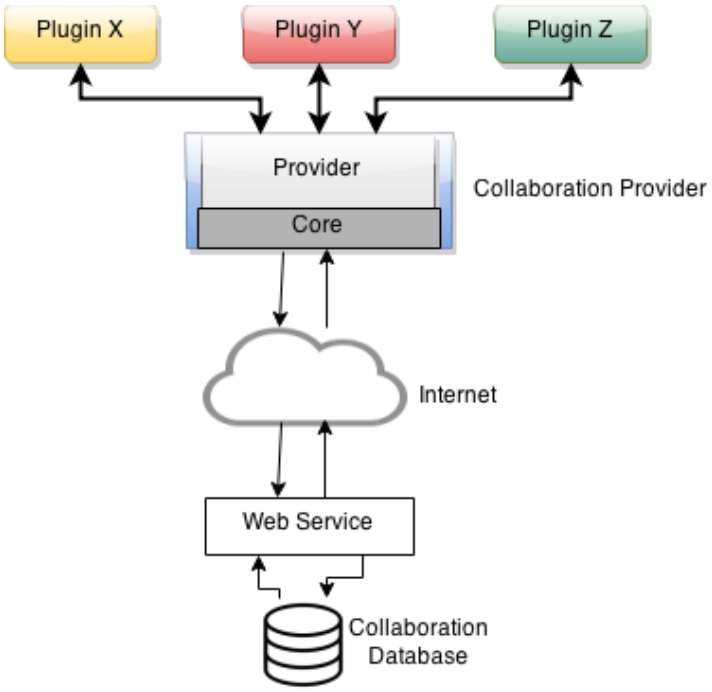

Figura 3.7: Relação dos plugins com o web service via Collaboration Provider

3.1.10 *GiveMe Trace*: uma ferramenta de apoio a rastreabilidade de software

É um *plugin* para o ambiente *Eclipse*[1]. Atua gerando dados históricos sobre a rastreabilidade de software (LÉLIS, 2014).

Desenvolvido para se conectar automaticamente a repositórios SVN e GIT, é capaz de reportar informações em dois formatos diferentes, basicamente:

- *Arrays* de *strings:* dado um número ou um intervalo de *commits* (um número de *commit* é a identificação dada para cada versão de um software controlada por um repositório de código fonte), *GiveMe Trace* retorna uma lista dos métodos alterados por *commit*. Cada uma das *strings* retornadas segue o mesmo padrão: nomeDaClasse.java|nomeDoMetodo (nome da classe é concatenado com um *pipe* e com o nome do método alterado). Em termos práticos, um *array* retornado contém um conjunto de métodos de todas as classes de código fonte que foram modificadas num dado *commit,* ou um intervalo deles.
- Arquivos texto (txt): dado um número ou intervalo de *commits*, *GiveMe Trace* retorna um arquivo de *log* de alterações realizadas no repositório de código fonte, idêntico ao fornecido pelo próprio gerenciador de código fonte (SVN ou GIT). A diferença consiste na granularidade das informações. A Figura 3.8, extraída de (LÉLIS, 2014), mostra um exemplo de arquivo de *log* gerado na análise dos *commits* de número 19 e 20 (informados pelo usuário da ferramenta). Nela é possível ver que todas a modificações realizadas estão registradas a nível de métodos alterados, separados das classes as quais pertencem por um *pipe*.

[1]Desenvolvido como parte de um trabalho de conclusão de curso de um aluno do curso de Bacharelado em Sistemas de Informação da Universidade Federal de Juiz de Fora

```
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 repositorio

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

Figura 3.8: Trecho do arquivo de rastreabilidade gerado a nível de método

Considera-se o seguinte cenário: identificar quais pontos do código fonte de um projeto poderão ser impactados quando uma alteração for realizada em outro dado ponto do código. Anteriormente, a integração do *GiveMe Trace* à *GiveMe Infra* somente permitia identificar tais pontos a nível de classe, ou seja, quais classes poderiam ser impactadas quando uma outra dada classe fosse alterada. Essa granularidade de informação não era suficiente para resolver o problema identificado na Empresa Parceira 1 dado que eles possuem classes com número de linhas de código superior a 15 mil, tornando-se vaga a informação de que aquela classe poderia ser impactada. *GiveMe Trace* contribuiu para que as indicações sejam calculadas a partir de então a uma granularidade de método.

3.1.11 Mylyn Mantis

É um *plugin* livre e de código aberto disponibilizado em (MYLYN-MANTIS, 2014), que atua de forma integrada com o ambiente *Eclipse*, permitindo que desenvolvedores e demais usuários cadastrem defeitos encontrados em seus projetos de código fonte.

Na *GiveMe Infra*, *Mylyn Mantis* permite que solicitações de mudanças sejam mantidas pela equipe de suporte ou a própria equipe de desenvolvimento. Cada uma das solicitações cadastradas darão origem a uma nova Atividade de Manutenção na *GiveMe Infra* e que posteriormente será efetuada pela equipe distribuída. Ao final, dados de alterações realizadas no código fonte, para a efetivação da solicitação de mudança, serão associadas à respectiva solicitação de mudança. Isso significa que todos os métodos, classes e módulos alterados serão automaticamente identificados (através do *plugin GiveMe Trace*) e associados ao registro da sua respectiva solicitação de mudança. Este recurso será apresentado em detalhes na seção 3.2.

3.1.12 Subclipse

É um *plugin* livre e de código aberto desenvolvido por *CollabNet* e disponibilizado em (SUBCLIPSE, 2014), que atua de forma integrada com o ambiente *Eclipse* suportando atividades de controle de versão de código fonte em repositórios *Subversion*.

Na *GiveMe Infra*, *Subclipse* permite que as alterações realizadas pela equipe de manutenção e evolução de software sejam enviadas ao repositório de código fonte do projeto (*commit*).

Um novo recurso foi inserido na versão do *Subclipse* que foi integrada a *GiveMe Infra*. Detalhes desse recurso serão apresentados na seção 3.2.

3.2 RECURSOS DISPONÍVEIS

GiveMe Infra disponibiliza para seus usuários uma gama de recursos que podem ser usados nas diferentes atividades suportadas pela infraestrutura. A Figura 3.9 ilustra o uso dos recursos da *GiveMe Infra.*

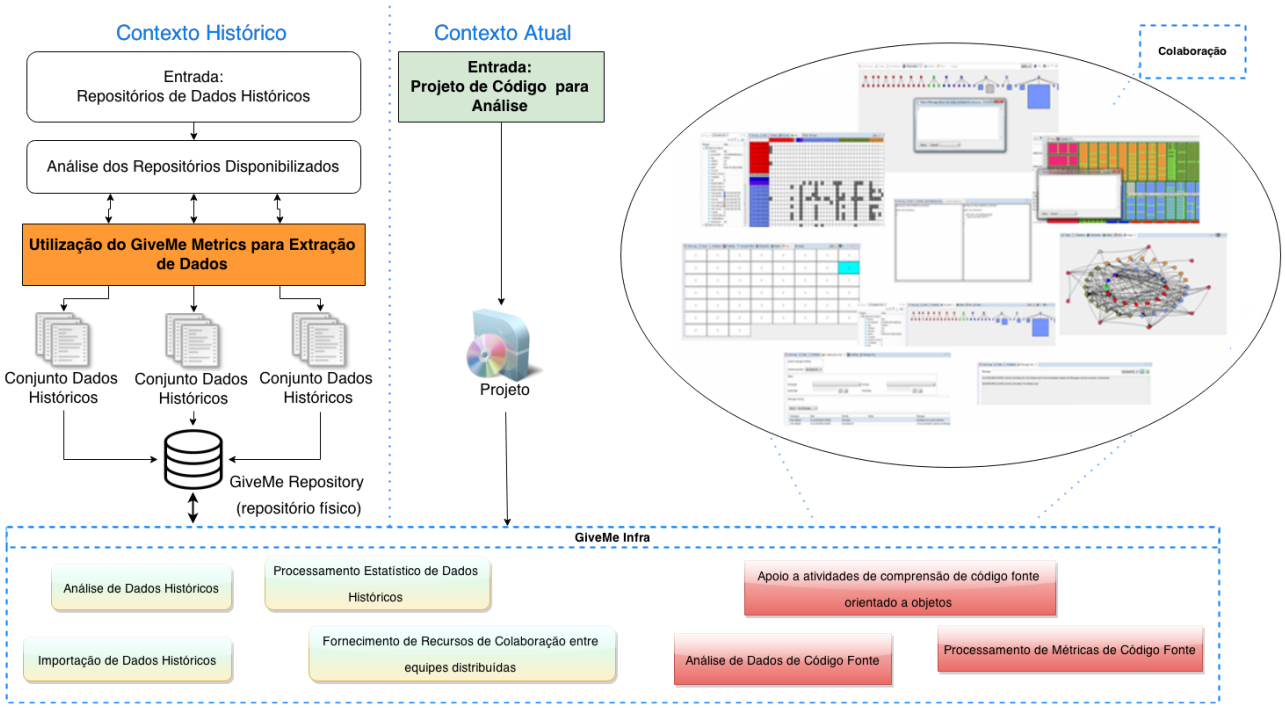

Figura 3.9: Uso dos recursos da GiveMe Infra

Dependendo do contexto escolhido, dados históricos de software, ou dados de código fonte, serão a entrada para a *GiveMe Infra* (Figura 3.9), que irá processá-los gerando informações que alimentarão as diferentes visualizações disponíveis (podendo haver colaboração diretamente nelas), como pode ser visto no balão da Figura 3.9.

Os recursos disponíveis na *GiveMe Infra* visam apoiar diferentes atividades de manutenção, estejam elas sendo executadas observando o Contexto Atual ou o Contexto Histórico. No contexto atual, os recursos disponíveis visam apoiar a compreensão do software analisado permitindo que, através de visualizações, seja possível entender o projeto de código, bem como os relacionamentos existentes entre classes, métodos e módulos. Entretanto, como o contexto atual é implementado através das ferramentas de terceiros, os recursos pertencentes a ele não serão aqui abordados, pois estão apresentados detalhadamente em (CARNEIRO, 2011), (CONCEIÇÃO, 2012) e (SILVA, 2013).

A Figura 3.9 mostra que, além da parceria com a Empresa Parceira 1 e com o SINAPAD foi estabelecida também uma parceria com outra empresa de desenvolvimento de software que será chamada de Empresa Parceira 2, que cedeu acesso a seus repositórios de código fonte e de solicitações de mudanças. O projeto *SpedFiscal* pertence a Empresa Parceira 2 e foi cedido para a execução da *GiveMe*

Infra. Trata-se de um software que atua na extração de dados e pela montagem de arquivos que contêm informações de interesse da Receita Federal do Brasil e de demais órgãos envolvidos na prestação de contas relativas a impostos sobre operações realizadas pelo contribuinte, como operações de compra e venda na indústria e comércio. Ele possui 15 pacotes de código fonte que totalizam 224 classes Java.

A partir deste ponto serão apresentados os principais recursos que a *GiveMe Infra* disponibiliza, observando-se o Contexto Histórico.

A Figura 3.10 mostra a tela de cadastro de solicitações de mudanças disponibilizada pelo *plugin Mylyn Mantis*. O campo *Modified Methods* diz respeito ao registro dos métodos que sofreram manutenção para que a solicitação de mudança fosse realizada. Essas informações são obtidas ao se colocar um número de *commit*, ou um intervalo de números, no campo *Commit Number*. Depois, a ferramenta *GiveMe Trace* obtém os métodos modificados e os associa à solicitação de mudança.

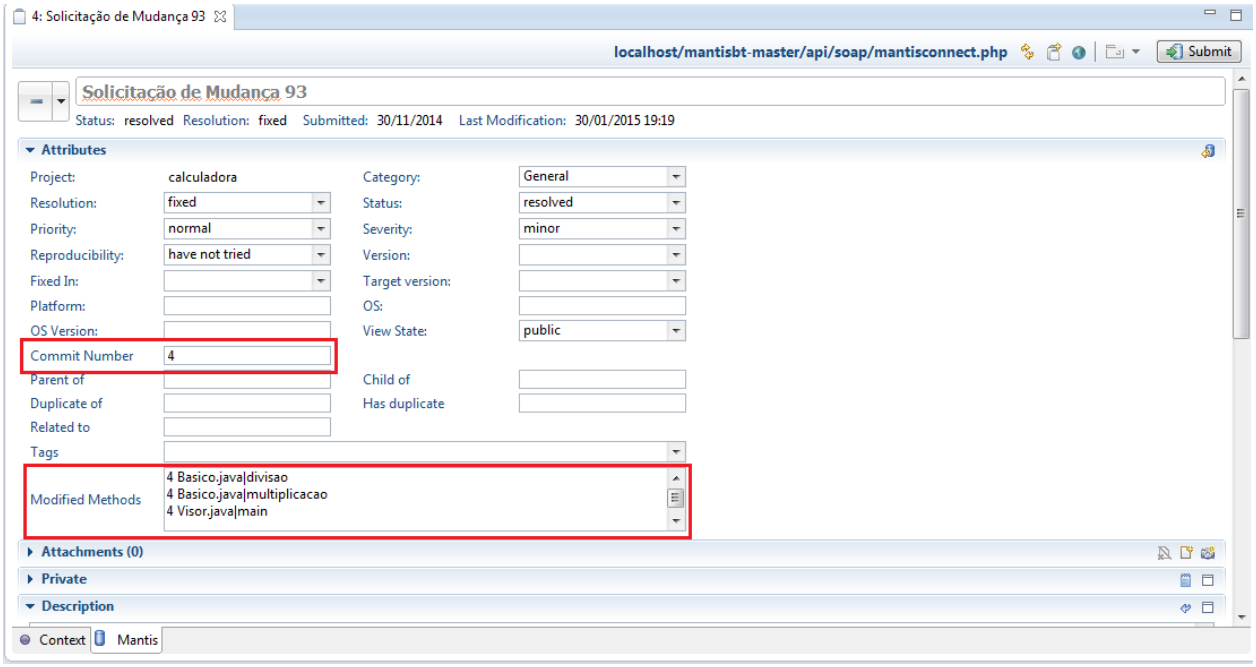

Figura 3.10: Tela cadastro de solicitação de mudança

Quando uma solicitação de mudança é cadastrada, o campo *Modified Methods* fica vazio até que a solicitação de mudança seja finalizada mas, para que isso aconteça, é necessário que uma equipe de manutenção dê início a uma atividade de manutenção. As Figuras 3.11, 3.12 e 3.13 mostram as telas dos recursos disponíveis para o gerente ou responsável pela equipe de manutenção gerir, desde as atividades de manutenção, até os

desenvolvedores que compõem a equipe de manutenção, bem como os projetos que podem ser alvo de manutenção, proporcionando a coordenação.

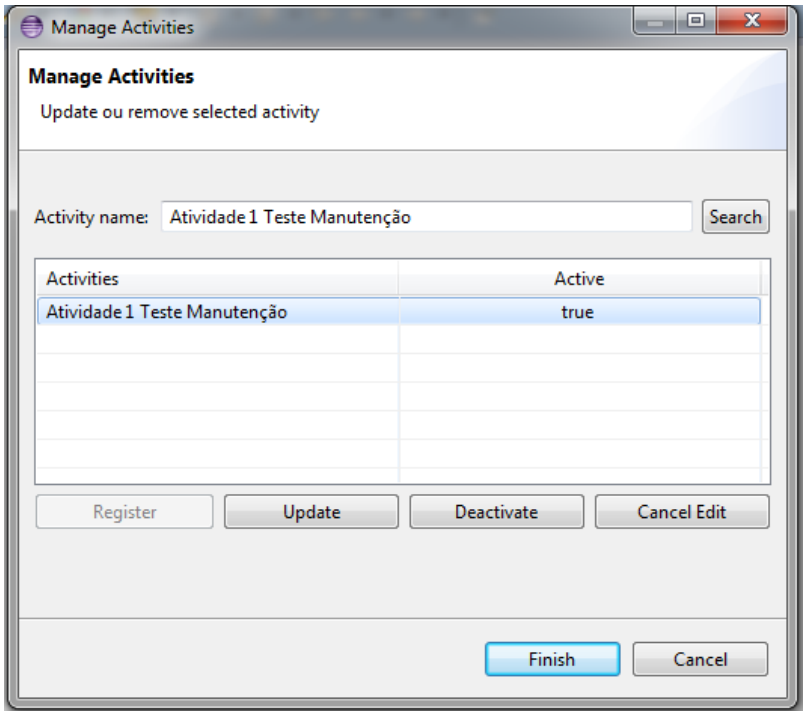

Figura 3.11: Gerenciamento das atividades de manutenção

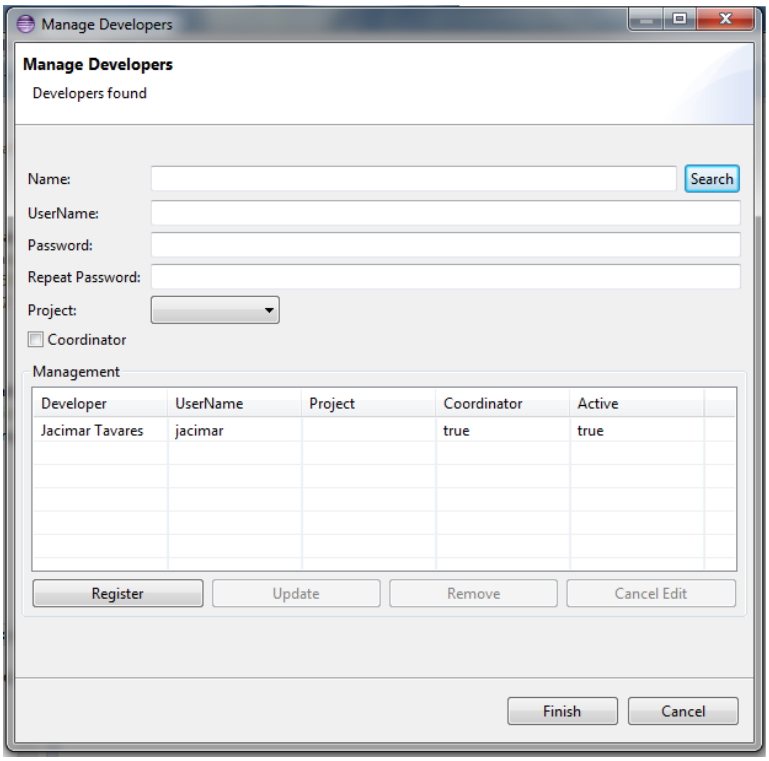

Figura 3.12: Gerenciamento de desenvolvedores

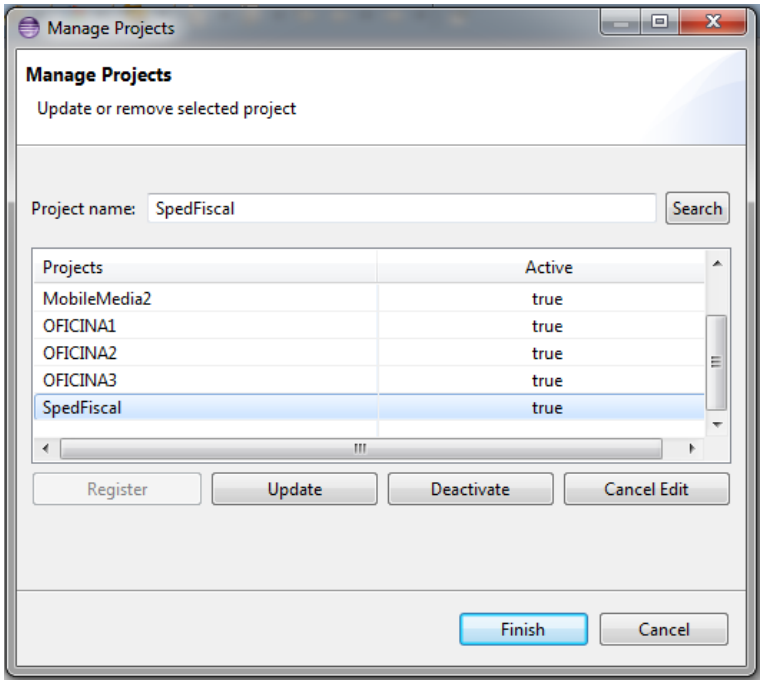

Figura 3.13: Gerenciamento de projetos

Para que o gerente da equipe de manutenção e os desenvolvedores possam atuar em uma atividade de manutenção, é necessário efetuar *login* na *GiveMe Infra*, como pode ser visto na Figura 3.14. Para isso, basta que seja informado *login*, senha e o projeto alvo da atividade de manutenção.

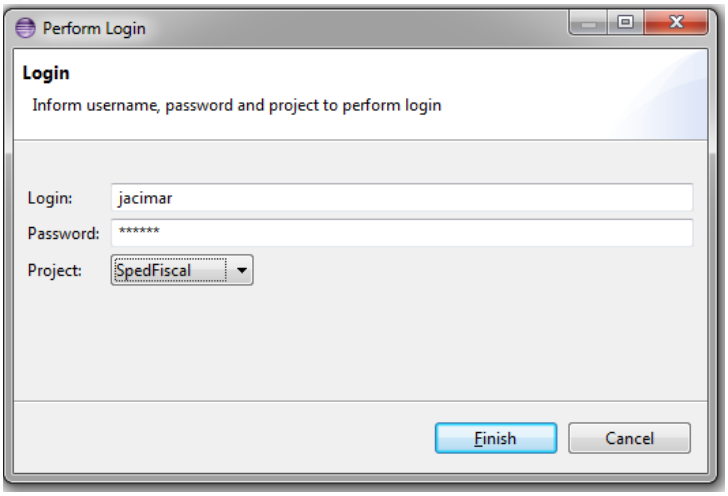

Figura 3.14: *Login* na *GiveMe Infra*

GiveMe Infra disponibiliza um conjunto de recursos que permitem ao desenvolvedor analisar quais os impactos de uma alteração desejada. A Figura 3.15 mostra a tela principal da *GiveMe Views.* Quando um desenvolvedor define que um dado método irá sofrer manutenção, basta selecioná-lo e executar o *GiveMe Views* (botão direito sobre o projeto a ser analisado e clique na opção *Visualizer with GiveMe* Views) que irá realizar o processamento estatístico usando o SAE.

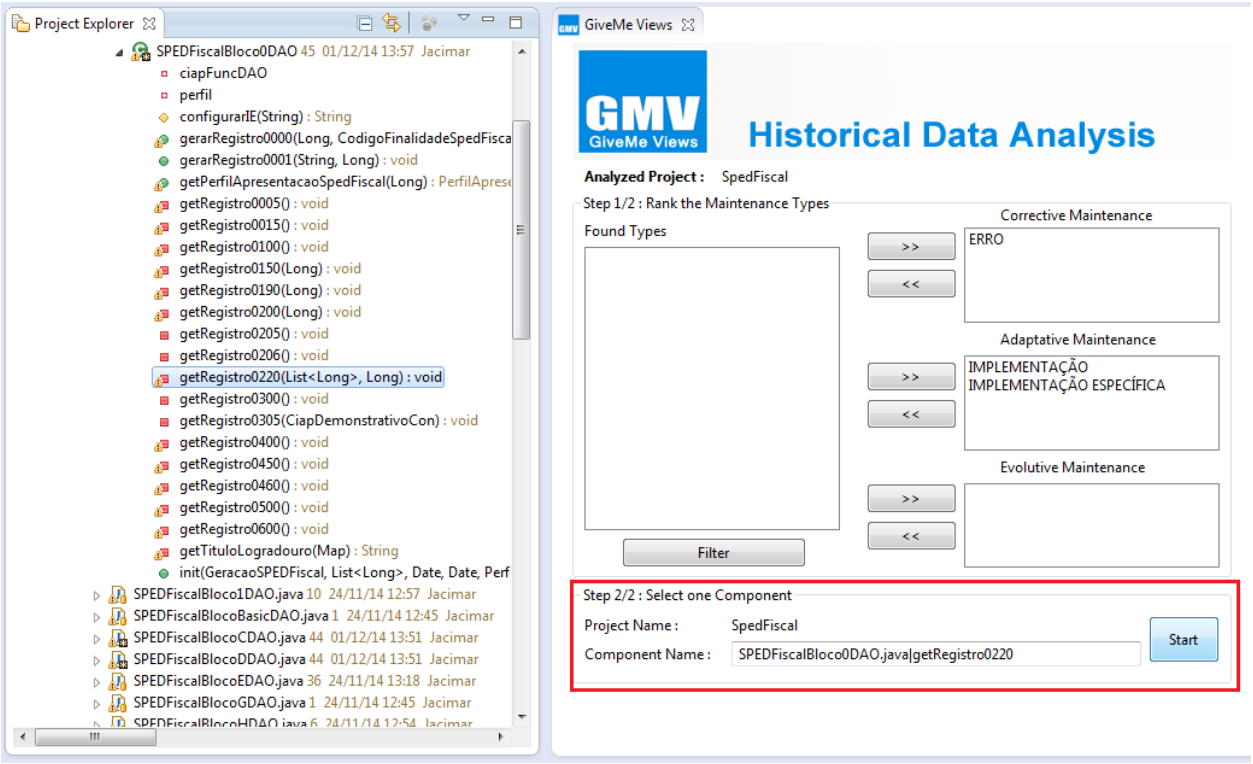

Figura 3.15: Tela principal GiveMe Views.

A visualização *Graph View*, no Contexto Histórico, permite que o desenvolvedor veja a relação histórica entre métodos. Cada pequeno círculo da Figura 3.16 (entidade) representa um método, sendo as arestas que os ligam a representação das relações existentes entre eles. Um método é ligado a outro se, historicamente falando, eles sofreram alteração juntos, na resolução de alguma solicitação de mudança anterior. O método selecionado tem suas propriedades exibidas na *Properties View*. O método selecionado é o *getRegistro200*, pertencente ao projeto *SpedFiscal* implementado na classe *SPEDFiscalBloco0DAO.java.*

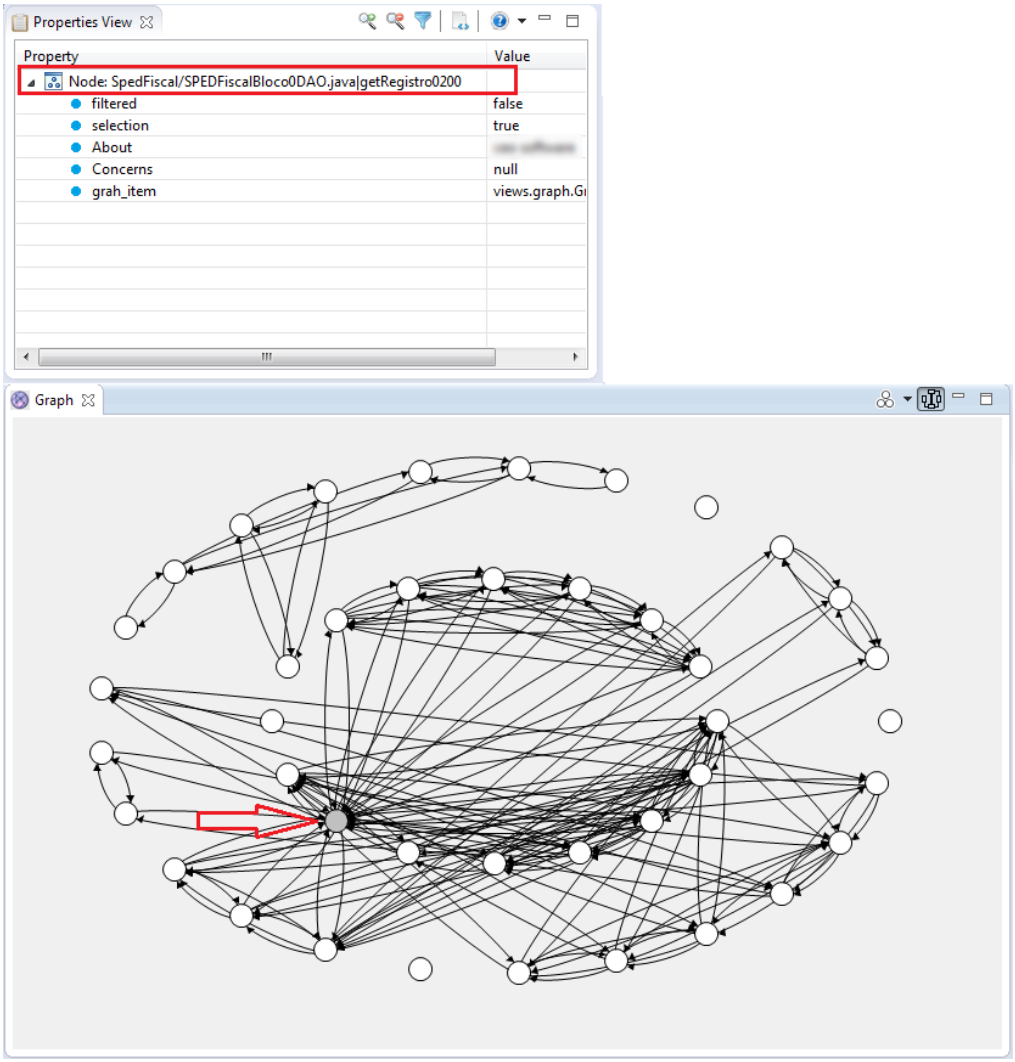

Figura 3.16: *Graph View* usada no Contexto Histórico

A *Tree View* tem a mesma finalidade da *Graph View*, mas com uma representação em forma de árvore. A Figura 3.17 mostra a *Tree View* criada após a seleção de um método para análise, na interface principal do *GiveMe Views*, bem com os métodos que se relacionam com ele.

Além de exibir os métodos que podem ser impactados quando o método selecionado for modificado, a *Tree View* ainda exibe um número que representa a porcentagem de chances de isso ocorrer considerando o histórico de alterações, calculado pelo SAE.

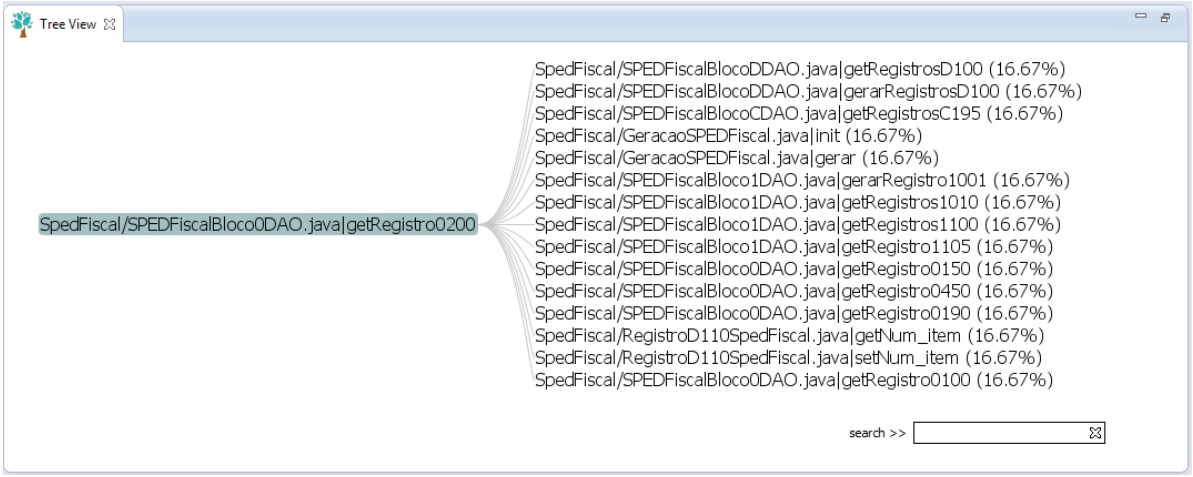

Figura 3.17: *Tree View*

A visão *Matrix View* (Figura 3.18), também permite visualizar a relação entre métodos, mas em forma de matriz, onde é possível ver listados todos os métodos de todas as classes que sofreram manutenções ao longo dos ciclos de manutenção do projeto *SpedFiscal*. Com isso é possível estabelecer a relação existente entre eles, bem como a porcentagem de chances de que um seja impactado quando outro for alterado, historicamente falando. O exemplo da Figura 3.18 mostra que as chances do método *getRegistrosD100* ser impactado com alterações no método *getRegistro0460*, historicamente falando, é de 33,33%.

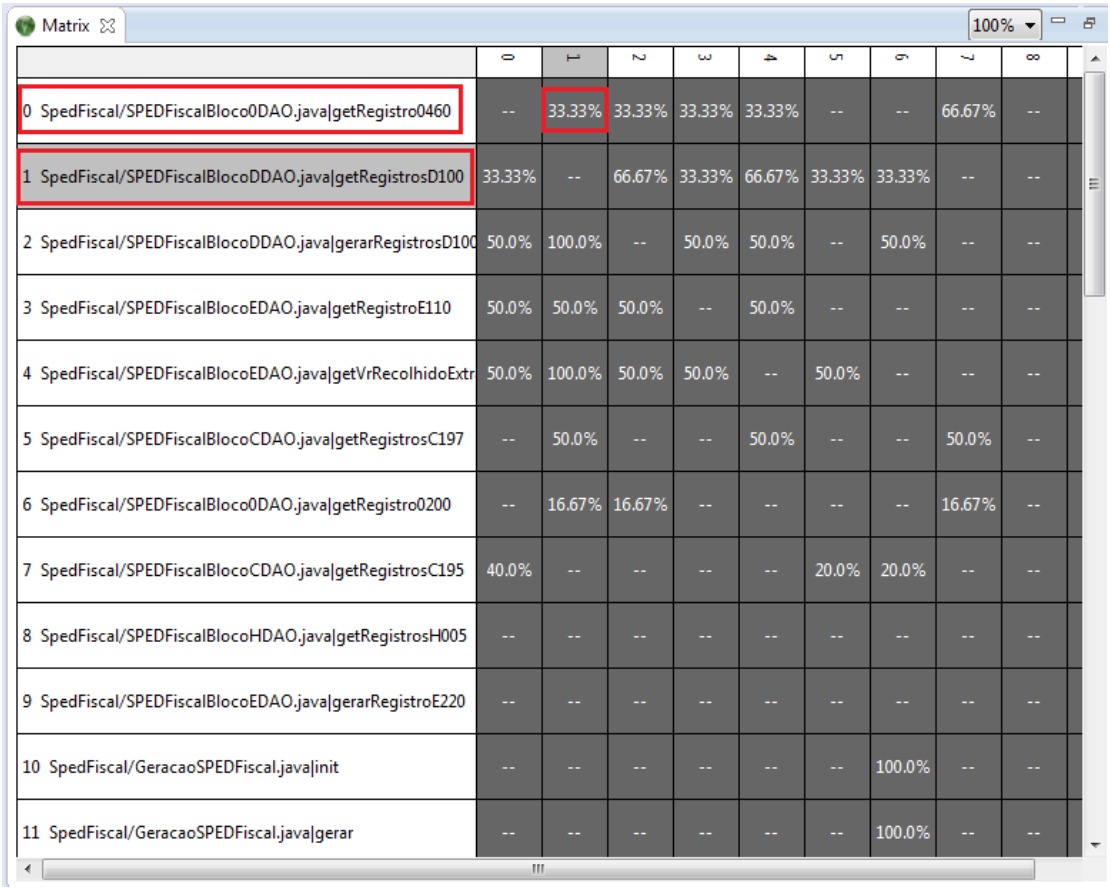

Figura 3.18: *Matrix View*

Caso um dado método que se deseja alterar possua relação com uma grande quantidade de métodos, a manutenção pode exigir alguns cuidados para que o impacto da alteração seja controlado. A *Grid View* permite que o desenvolvedor consulte a quantidade de alterações que um método sofreu em relação ao número de solicitações de mudanças que impactaram um dado projeto. A Figura 3.19 mostra que o projeto *SpedFiscal* foi alvo de 468 solicitações de mudanças, sendo que, delas, apenas 3 afetaram o método *getRegistrosD100*.

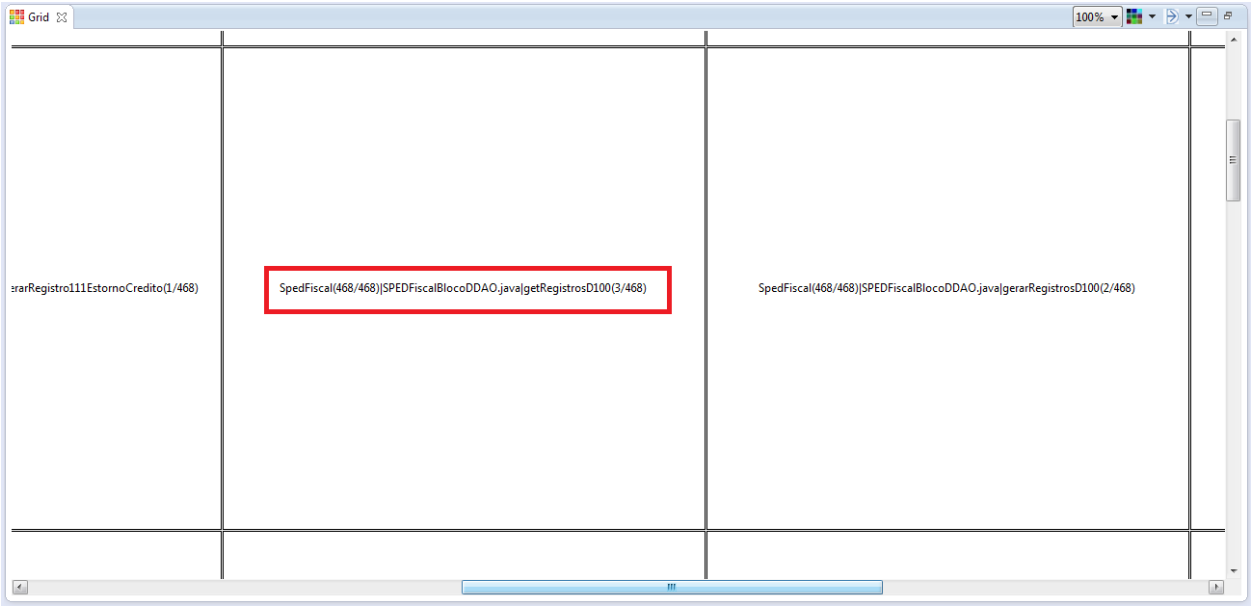

Figura 3.19: *Grid View*

Assim como a *Tree View*, a *Deep View* também exibe os métodos que podem ser impactados quando um dado método for alterado e a porcentagem estatística de um impactar o outro, historicamente falando, de forma ordenada pela maior porcentagem estatística. Entretanto, *Deep View* possui um diferencial: é capaz de analisar os impactos em diferentes níveis de profundidade. A Figura 3.20 mostra os métodos que podem ser impactados caso uma alteração seja feita no método *getRegistro0200*, pertencente a classe *SPEDFiscalBloco0DAO*. Numa atividade de manutenção, o desenvolvedor que modificasse o método *getRegistro0200* (método pai), teria que verificar, em todos os métodos que se relacionam com ele (filhos), se alguma alteração será necessária. Caso um dos filhos seja alterado, um novo ciclo se iniciará, sendo necessário verificar se os filhos do filho modificado foram impactados. Para analisar os métodos que podem ser impactados quando um método filho for alterado, basta clicar com o botão direito do *mouse* sobre um método que a *Deep View* automaticamente irá exibir os filhos.

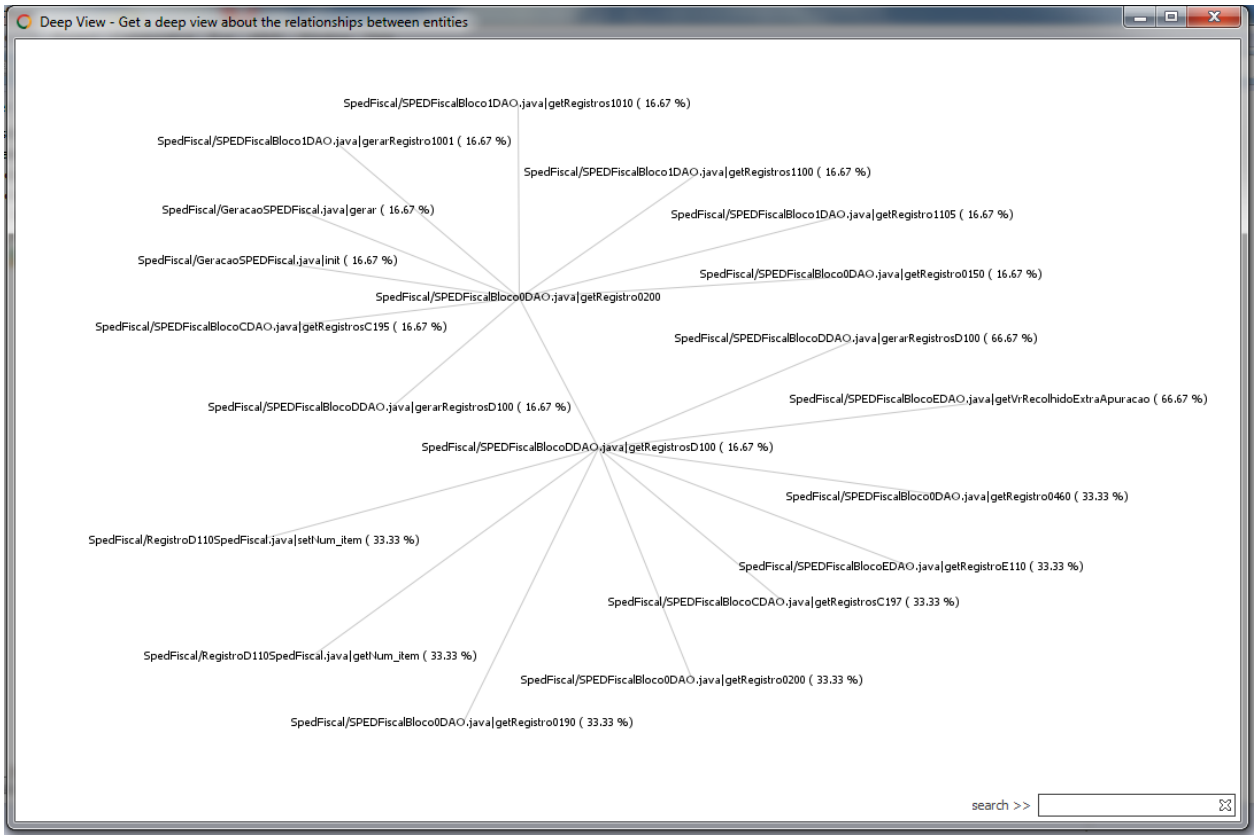

Figura 3.20: Deep View

À medida que um membro da equipe distribuída for se guiando pelas indicações fornecidas pelas visualizações apresentadas, pode ser necessário interagir com outros membros da equipe colaborativamente. *Graph View*, *Grid View*, *Matrix View*, *Tree View* e *Deep View* permitem que mensagens de colaboração sejam enviadas sobre uma entidade (que no contexto histórico refere-se a um método de uma classe). Para que uma mensagem seja enviada, basta clicar com o botão direito do *mouse* sobre a entidade e será aberta a janela para envio de mensagens, como pode ser visto na Figura 3.21, parte A. Para visualizar as mensagens de colaboração cadastradas para uma entidade basta um clique com o botão esquerdo do *mouse* (Figura 3.21, parte B).

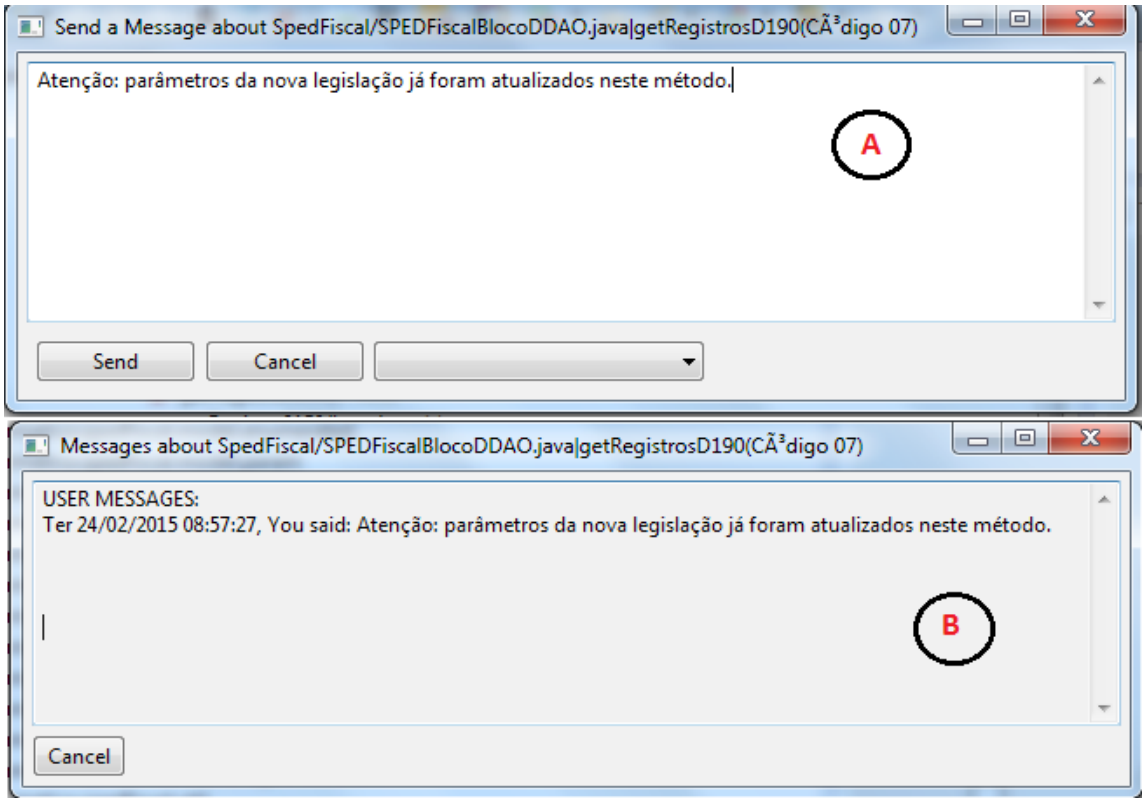

Figura 3.21: Envio e visualização de mensagens de colaboração

Toda a colaboração gerada através do envio de mensagens pode ser observada e gerenciada através da *Collaboration View* (Figura 3.22). Além das mensagens de colaboração inseridas nas visualizações, é possível ainda visualizar as mensagens de sistema, que são as geradas automaticamente quando ações são tomadas. Um exemplo é a mensagem gerada sempre que um desenvolvedor faz *login* na *GiveMe Infra*. Neste caso, um registro contendo data e hora é efetuado.

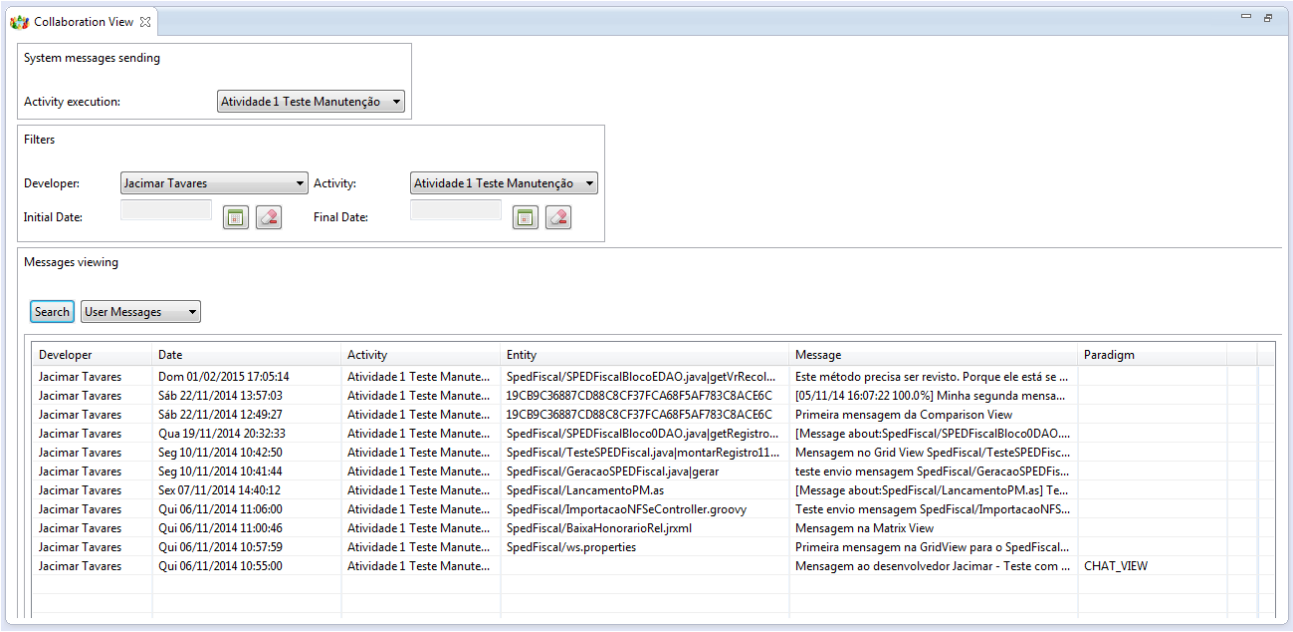

Figura 3.22: *Collaboration View*

Outro meio de comunicação disponível entre desenvolvedores é o envio de mensagens de conversação, como pode ser visto na *Message View*, Figura 3.23.

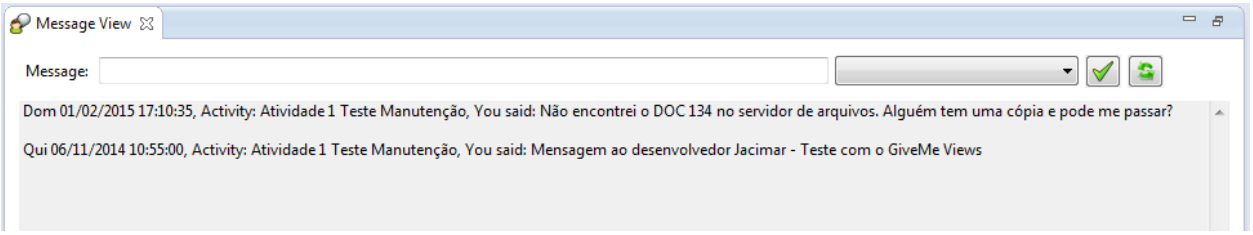

Figura 3.23: Message View

Durante o processamento dos dados históricos de um projeto, *GiveMe Infra* é capaz de calcular algumas métricas e exibi-las para o desenvolvedor, visando apoiar o processo de tomada de decisões sobre atividades de manutenção, como pode ser visto na Figura 3.24. Dentre as métricas listadas na Figura 3.24, é possível ver que as manutenções realizadas impactaram em apenas 41 métodos diferentes, de todos os métodos que o projeto *SpedFiscal* possui em suas classes. Isso mostra que as manutenções estão concentradas em um conjunto de métodos.

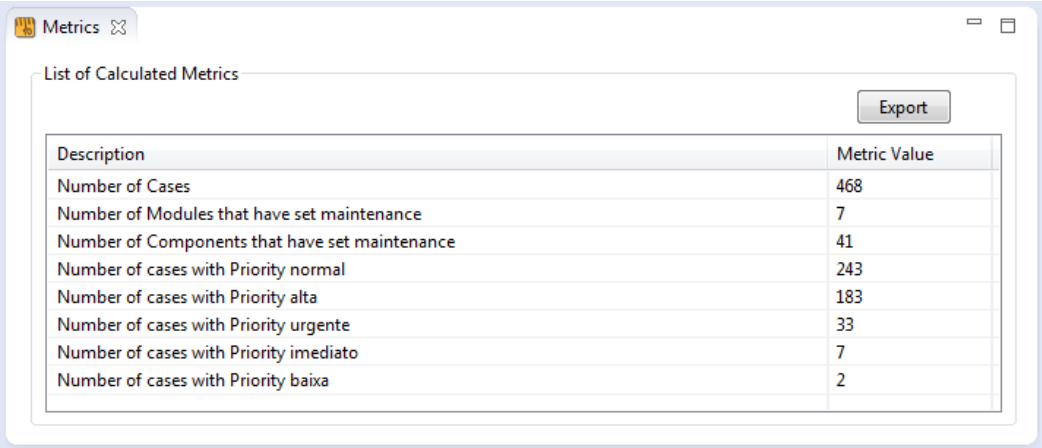

Figura 3.24: *Metric View*

GiveMe Infra implementa um conjunto de recursos que apoiam a tomada de decisões sobre alterações a serem realizadas no código fonte. Sempre que uma solicitação de mudança é encerrada quer dizer que uma atividade de manutenção foi também finalizada e as alterações de código foram enviadas para o repositório SVN ou GIT, em um ou vários *commits*. Para que as empresas parceiras possam acompanhar as alterações feitas no projeto (isto é, os métodos que foram alterados em um ou vários *commits*) foi desenvolvida a *Comparison View*, que permite comparar as indicações estatísticas calculadas pelo SAE com o que de fato foi alterado em uma solicitação de mudança. A Figura 3.25 mostra a *Comparison View* e os recursos que ela disponibiliza.

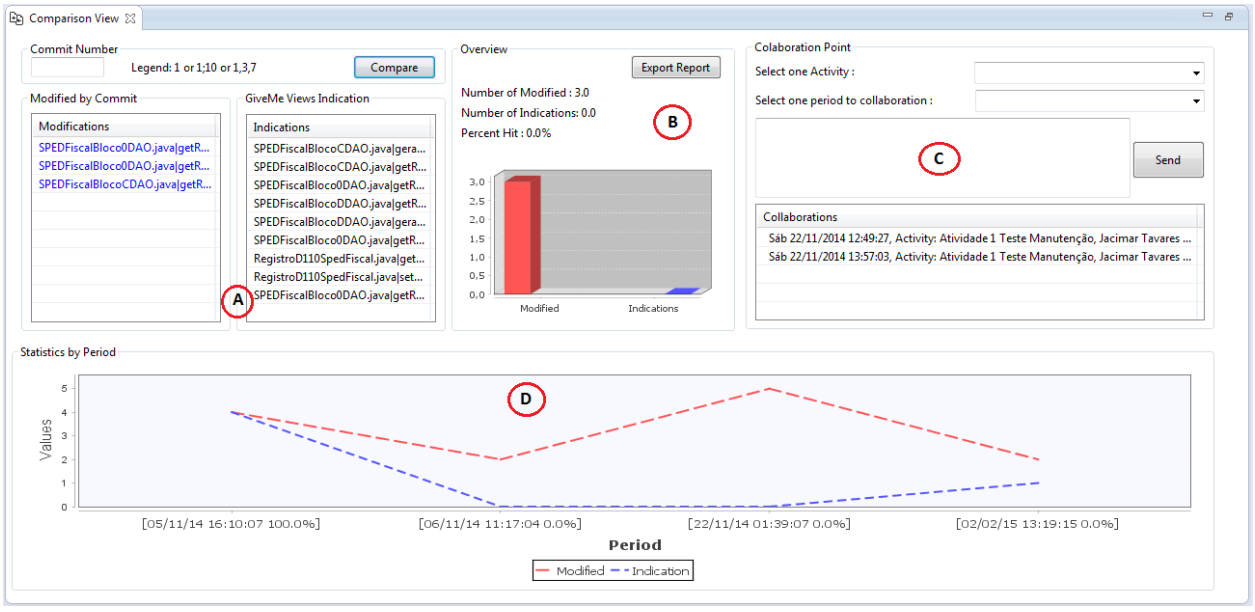

Figura 3.25: *Comparison View*

Na parte A da Figura 3.25 é possível visualizar os métodos que foram modificados para resolver uma solicitação de mudança e as indicações de métodos que deveriam ter sido considerados na manutenção, segundo indicações fornecidas pelo SAE. Os métodos modificados são obtidos informando um número ou *range* de *commits* que correspondem aos efetuados para a resolução da solicitação de mudança. Caso haja na lista das indicações algum dos métodos que também está na lista dos alterados, ele terá cor de fonte azul, tal como na lista dos modificados. O gráfico da parte B mostra a relação do número de indicados com o número dos que foram modificados. A parte D da figura mostra o histórico das últimas comparações realizadas, permitindo que o setor de qualidade acompanhe o contexto histórico das mudanças. A linha azul representa as indicações e a linha vermelha as modificações. Cada ponto no gráfico da parte D representa um período onde comparações foram geradas. Um período é formado por data, hora e a porcentagem de acerto obtido naquela comparação. Já a parte C é um ponto de colaboração, onde o setor de qualidade pode inserir mensagens de colaboração sobre algum período, refletindo alguma análise que possa ter feito daquela comparação.

A *Comparison View* permite comparar as modificações realizadas com as indicações estatísticas calculadas, mas para que isso seja possível, as mudanças efetuadas no código fonte já devem ter sido enviadas ao repositório de código fonte. Caso uma comparação mostre que algum método importante tenha ficado sem ser modificado, o setor de qualidade pode solicitar que o desenvolvedor efetue novas mudanças no código e depois atualize a solicitação de mudança com as informações de rastreabilidade.

Assim como *Comparison View*, *Changed View* atua permitindo a comparação entre o que foi indicado com o que de fato foi alterado, mas a diferença está no contexto de atuação. *Changed View* foi desenvolvida para guiar desenvolvedores no momento de efetuar a solicitação de mudança, comparando o que foi modificado na cópia de trabalho atual (*workcopy* do projeto) com as indicações da *GiveMe Infra*. A Figura 3.26 mostra a *Changed View*.

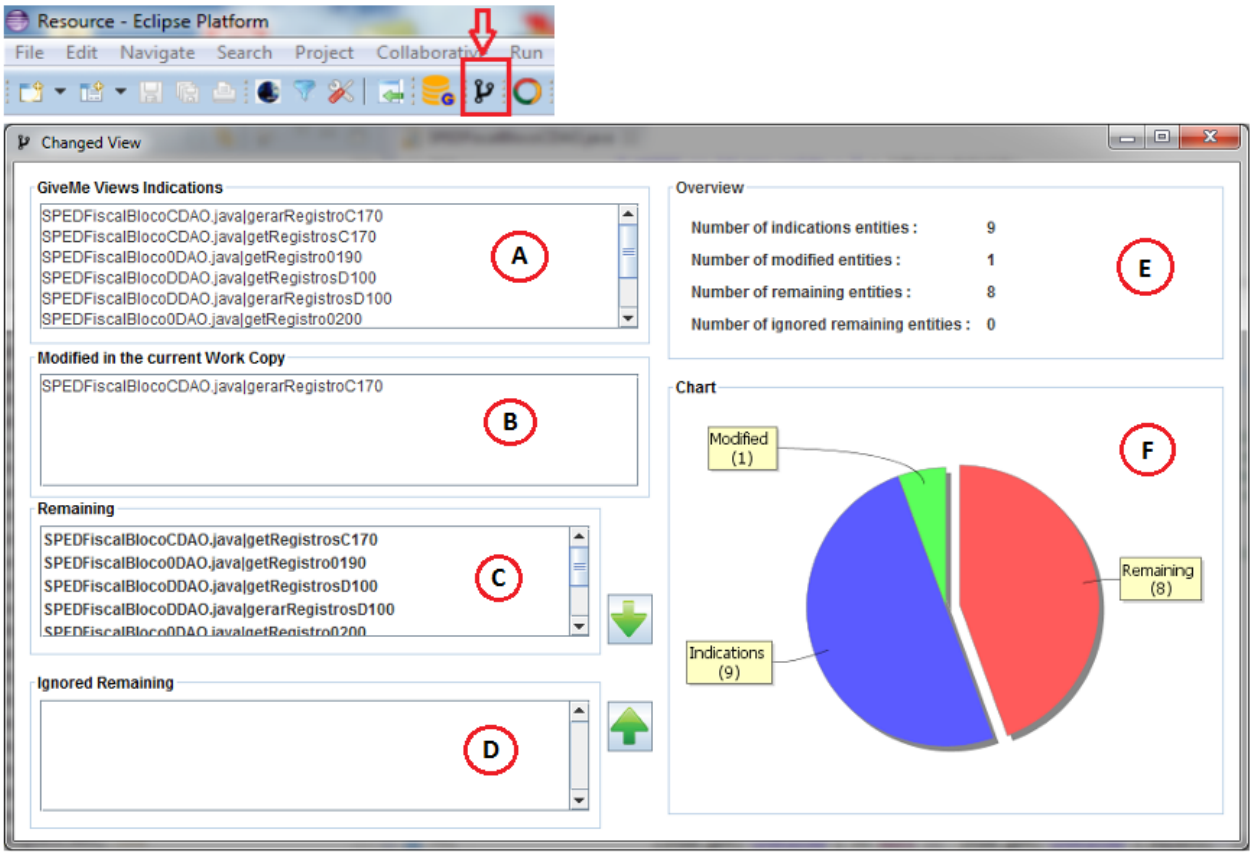

Figura 3.26: Changed View

A seta na Figura 3.26 mostra o botão de acesso rápido à *Changed View*. Ao acioná-lo, o desenvolvedor poderá acompanhar as indicações que a *GiveMe Infra* fornece para serem consideradas naquela solicitação de mudança (parte A). A lista na parte B mostra os métodos, dentre os indicados, que foram alterados no projeto atual (*workcopy*). A parte C mostra a lista dos restantes, ou seja, os que ainda não foram analisados pelo desenvolvedor, mediante as indicações feitas. Caso o desenvolvedor queira desconsiderar algum método indicado, basta movê-lo para a lista de ignorados (parte D). Isto é útil em casos onde o desenvolvedor, baseado na experiência, sabe que o método indicado não será impactado nesta solicitação de mudança. As partes E e F mostram um resumo das informações em formato textual e gráfico, respectivamente.

A vantagem da *Changed View* em relação a *Comparison View* é que ela atua antes das alterações no código fonte serem enviadas ao repositório do projeto. A medida que o desenvolvedor for efetuando modificações no código para resolver uma solicitação de mudança, *Changed View* poderá ser acionada para consulta.

Como dito na seção que apresentou as ferramentas que compõem a *GiveMe Infra*, o *plugin Subclipse* foi integrado à infraestrutura para permitir que as alterações realizadas no código sejam enviadas ao repositório. Foi implementado no *Subclipse* um recurso para permitir que, quando um desenvolvedor for realizar *commit* das modificações, uma verificação seja feita para analisar se existe alguma indicação de modificação restante que ainda não foi considerada pelo desenvolvedor (ou seja, algum método restante que não foi nem modificado, nem adicionado à lista de ignorados). Caso exista, o desenvolvedor é notificado através de uma mensagem que o questiona se deseja efetuar o *commit* mesmo assim ou visualizar as indicações de modificação restantes. Caso queira visualizar, a *Changed View* é aberta exibindo tais informações. Do contrário, será direcionado para a realização do *commit*.

Durante a realização de uma atividade de manutenção, o desenvolvedor pode se deparar com questões de tempo que o impeça de verificar todos os métodos indicados pela *GiveMe Infra* como possíveis alvos de manutenção. Neste contexto, o que se pode fazer é considerar os de maiores chances estatísticas, historicamente falando, e começar por eles. Além das porcentagens estatísticas, tem-se outro recurso que pode ser considerado. Está disponível na *Modules and Components View* (Figura 3.27).

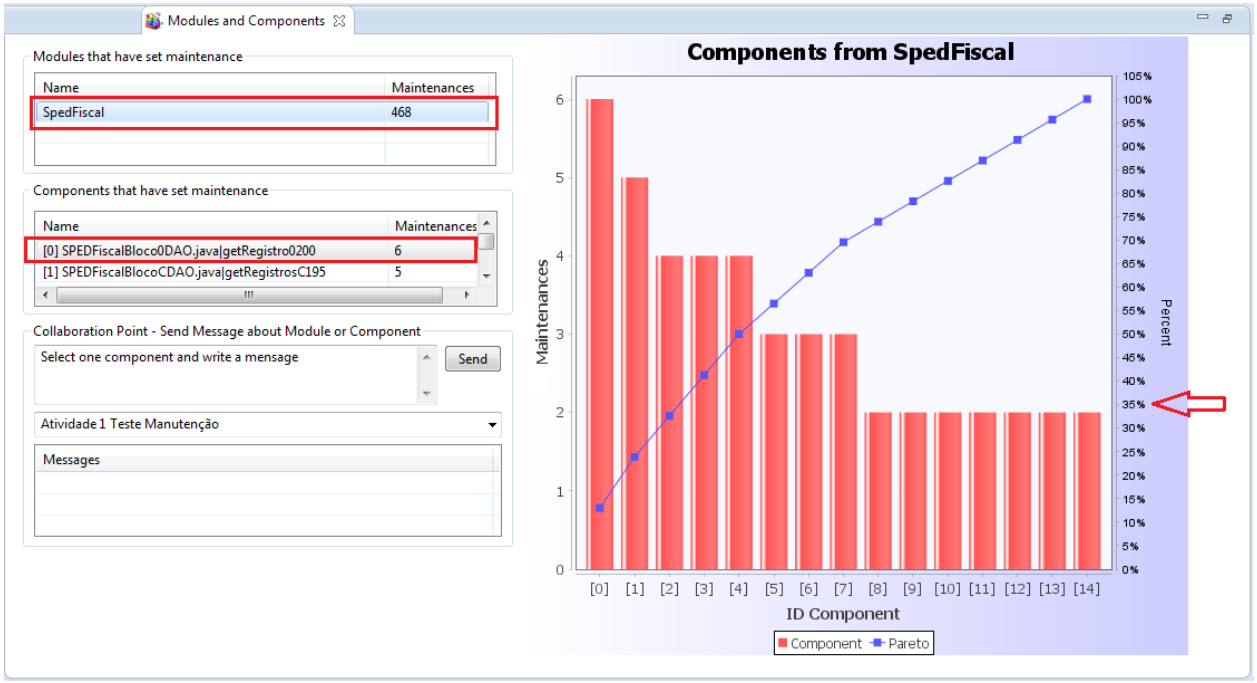

Figura 3.27: Modules and Components View

A Figura 3.27 mostra o exemplo da análise do projeto *SpedFiscal*. A primeira informação que se tem é que ele foi alvo de 468 manutenções, sendo que delas, 6 afetaram o componente *getRegistro0200*.

O gráfico exibido à direita mostra o número de manutenções por componente e uma curva de Pareto (HOFFMANN, 1973), que indica que 35% das manutenções ocorreram nos métodos representados pelos *ids* 0, 1 e 2, tornando-os boas opções de manutenção inicial.

As duas últimas visualizações a serem apresentadas nesta seção correspondem a recursos de exportação de relatórios e configurações de filtros. *Output View* (Figura 3.28) permite que sejam gerados relatórios para exportar desde as análises geradas pelo SAE até as métricas calculadas para um projeto, e as análises históricas geradas na *Comparison View*. Já *Filter View* permite que sejam informados parâmetros para configurar as visualizações *Comparison View, Changed View, Deep View,* e *Tree View*. O primeiro parâmetro permite configurar as indicações estatísticas mostradas na *Comparison View* e *Changed View* para considerar apenas as acima de 20% de chances de ocorrência de impacto, permitindo que o desenvolvedor calibre as indicações e veja somente as com relevância acima de uma dada porcentagem de sua escolha. Os demais parâmetros da *Filter View* correspondem ao número de indicações que cada uma das visualizações deverá exibir. Em caso de parâmetro zero, a visualização irá exibir todas as indicações estatísticas, independente de quantas tenham sido calculadas.

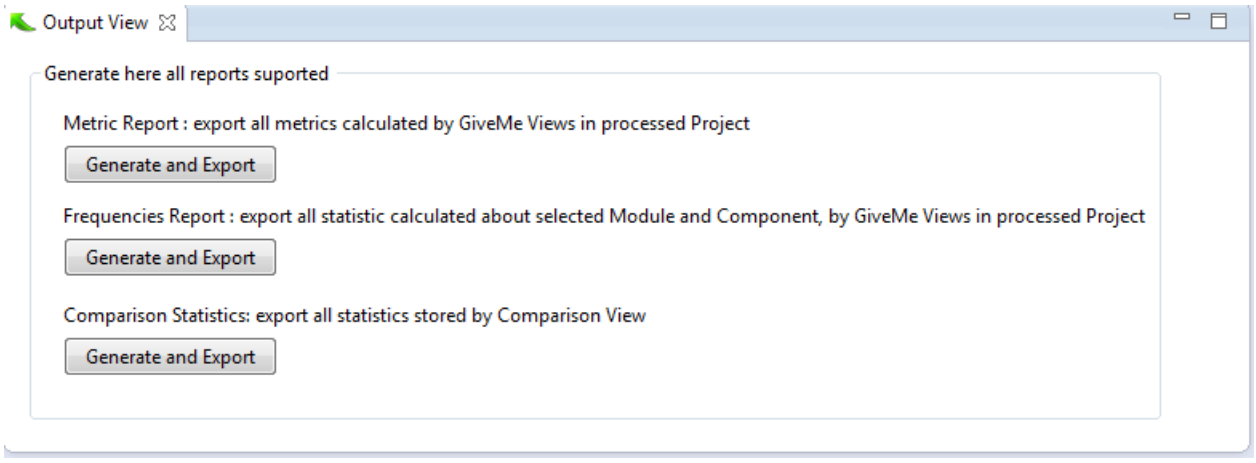

Figura 3.28: Output View

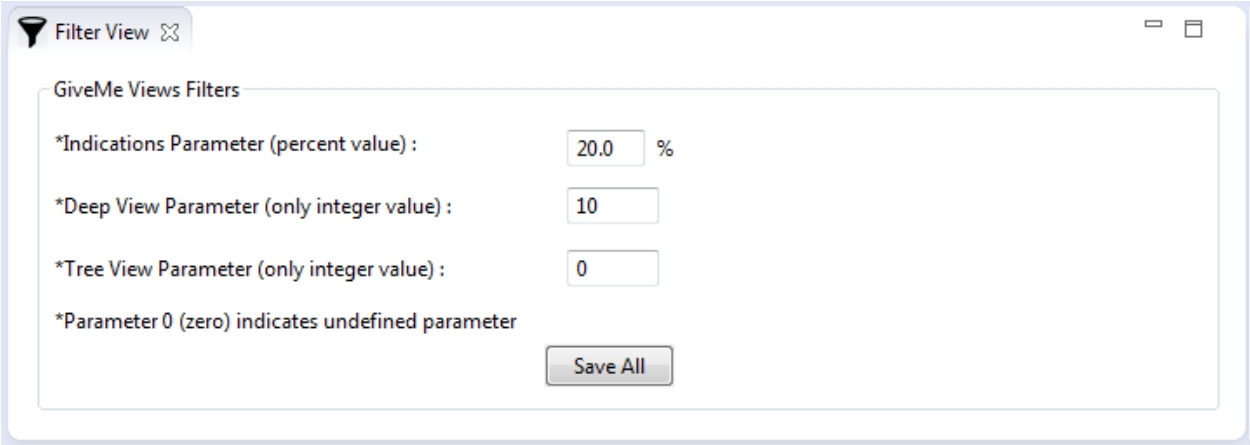

Figura 3.29: Filter View

3.3 PROCESSO DE DESENVOLVIMENTO DA SOLUÇÃO

A Etapa (iii) da metodologia, definida no capítulo de Introdução deste trabalho, deu origem a um processo formado por uma sequência de atividades. Através delas é possível visualizar todas as atividades realizadas desde o planejamento até o desenvolvimento e validação da solução apresentada neste trabalho, a *GiveMe Infra*. O processo aqui descrito possui duas sub-etapas, (i) Fundamentação da Proposta e (ii) Materialização da Proposta.

A sub-etapa Fundamentação da Proposta (Figura 3.30) engloba o período do desenvolvimento da *GiveMe Infra* que corresponde a fundamentação das teorias estatísticas, atualmente implementadas na ferramenta no *GiveMe Views*.

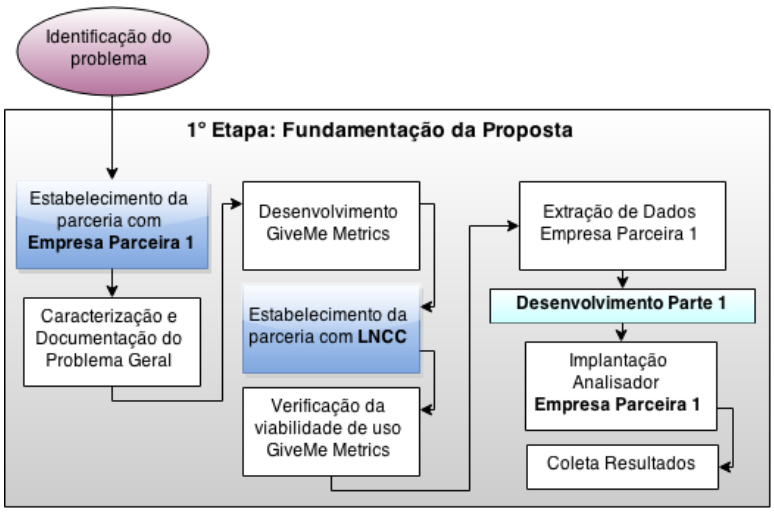

Figura 3.30: Fundamentação da Proposta

Após identificação e caracterização do problema foi desenvolvido o *GiveMe Metrics*, com o intuito de permitir a extração dos dados históricos que seriam analisados estatisticamente. Em seguida, uma parceria foi estabelecida com o Sistema Nacional de Processamento de Alto Desempenho (SINAPAD[2]) atualmente vinculado ao Laboratório Nacional de Computação Científica (LNCC[3]) que forneceu acesso a repositórios de dados que foram manipulados via *GiveMe Metrics*, objetivando a extração de dados históricos para serem usados numa prova de conceitos. Depois que a viabilidade de uso do *GiveMe Metrics* foi verificada, conduziu-se a extração de dados da Empresa Parceira 1. Os dados extraídos foram analisados e o modelo estatístico (SAE) foi criado e validado. A atividade Desenvolvimento Parte 1 (Figura 3.30) pode ser vista de forma expandida na Figura 3.31. Com esta atividade, foi possível verificar a viabilidade de uso SAE junto a Empresa Parceira 1, como pode ser visto na Figura 3.30 (Implantação *Analisador* na Empresa Parceira 1). Após cinco meses da implantação da ferramenta piloto na Empresa Parceira 1, dados de resultados qualitativos foram coletados.

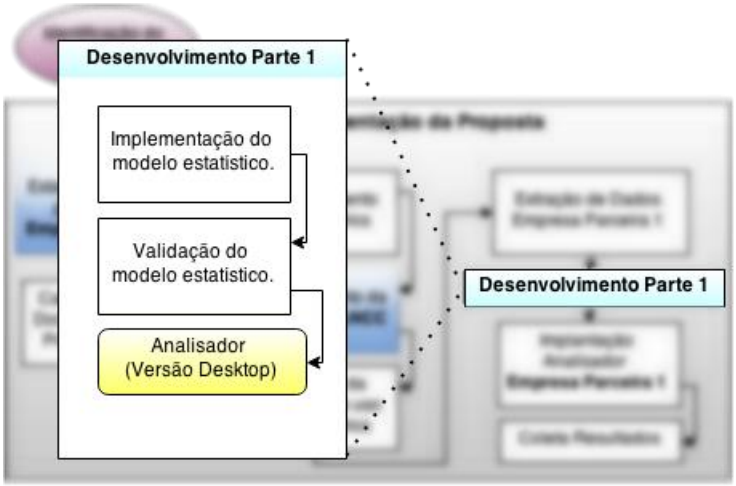

Figura 3.31: Atividade Desenvolvimento Parte 1 expandida

A sub-etapa Materialização da Proposta inicia após a fundamentação da proposta (Figura 3.32).

[2] https://www.lncc.br/sinapad/

[3] http://www.lncc.br

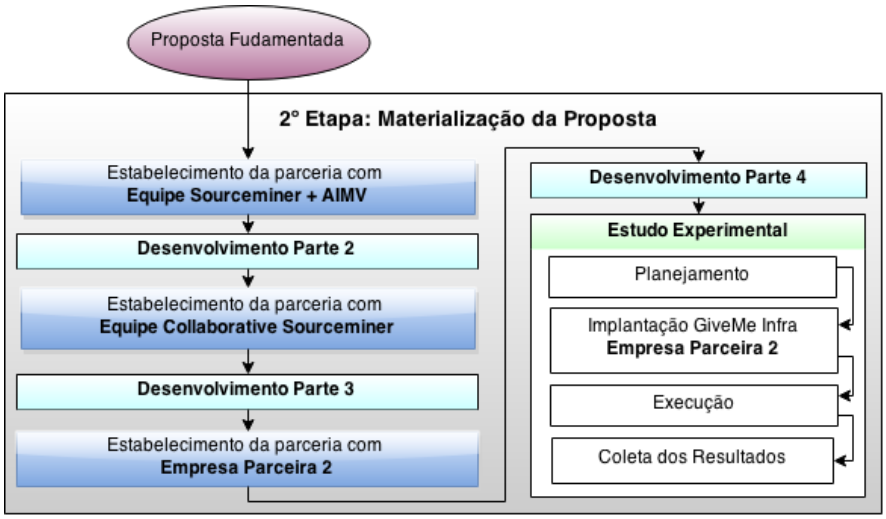

Figura 3.32: Materialização da Proposta

Com a proposta fundamentada, isto é, com o SAE implementado na ferramenta piloto *Analisador*, foi estabelecida a parceria com a Equipe Sourceminer + AIMV por um motivo em especial: possuíam um conjunto de visualizações que ajudariam diretamente na resolução do problema geral deste trabalho, evitando que fosse necessário desenvolver funcionalidades idênticas para a *GiveMe Infra.* Além disso, essa parceria proporcionaria ainda que *GiveMe Infra* fornecesse informações para equipes de manutenção nos dois contextos: Contexto Atual e Contexto Histórico, como apresentados na seção 3.1. A Figura 3.33 exibe o diagrama de dependências gerado com base nas relações de dependência existentes entre os *plugins* fornecidos pela Equipe Sourceminer + AIMV. Nela é possível ver que todas as visualizações e o *Sourceminer* dependem do *plugin* AIMV. Isto se dá pelo fato de que o AIMV é o responsável por manter em memória todos os dados que são manipulados pelas visualizações e gerados pelo *Sourceminer.*

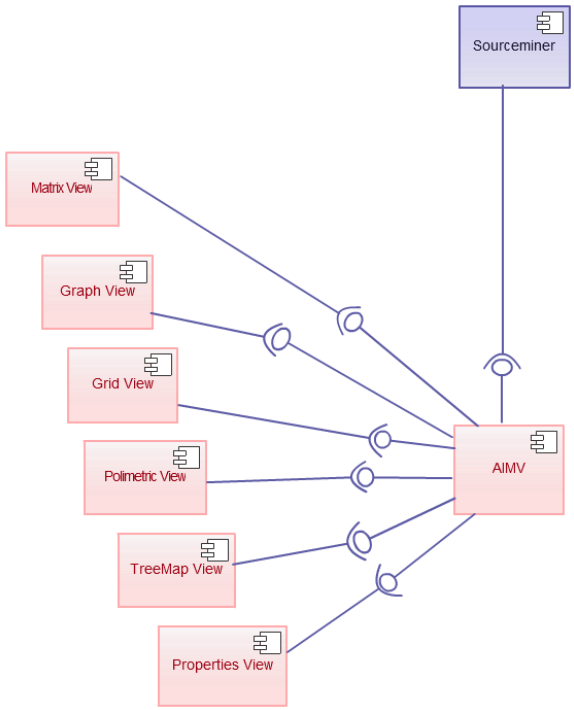

Figura 3.33: Diagrama que mostra a dependência entre os plugins fornecidos pela Equipe Sourceminer + AIMV

Em seguida deu-se início ao Desenvolvimento Parte 2 (Figura 3.32), que pode ser visto de forma expandida na Figura 3.34.

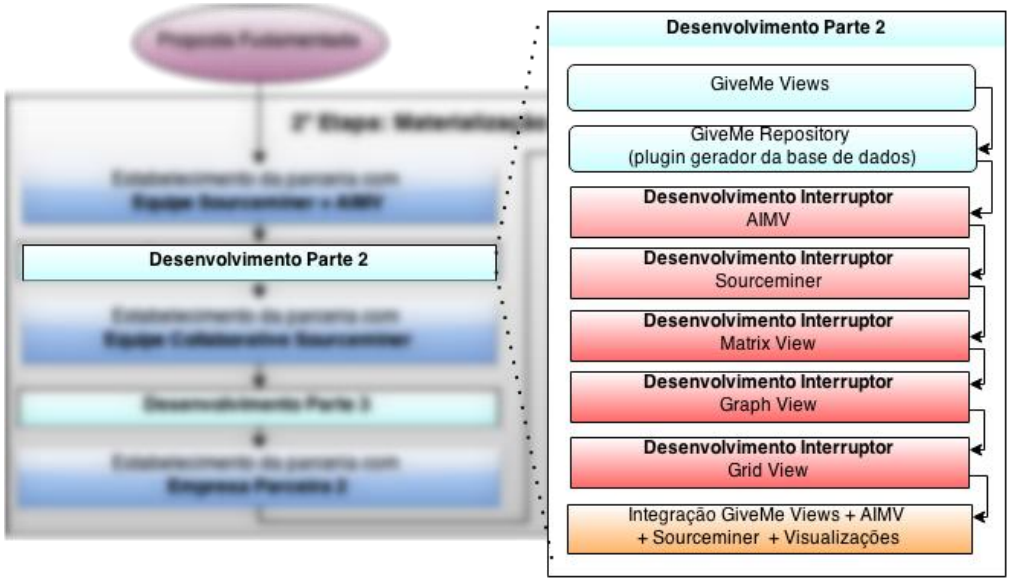

Figura 3.34: Atividade Desenvolvimento Parte 2 expandida

No Desenvolvimento Parte 2 foi implementada a ferramenta *GiveMe Views*. Com isso, surgiu a necessidade da implementação do *plugin GiveMe Repository*, para gerar o repositório responsável por armazenar os dados históricos.

As próximas atividades consistiram na preparação dos *plugins* AIMV, *Sourceminer* e na preparação das visualizações pertencentes ao *plugin* AIMV para que pudessem ser integradas ao *GiveMe Views.* Interruptores foram desenvolvidos para que tais *plugins* pudessem iniciar de forma diferente, dependo do tipo de demanda. A Figura 3.35 ilustra um interruptor, implementado na *Matrix View* cujo objetivo é fazer com que ela carregue dados de duas diferentes fontes de dados. Como a solicitação que chega a *Matrix View* é para exibir dados de código fonte, o interruptor irá ativar o acesso a fonte de dados provenientes de código fonte e, consequentemente, desativar o acesso à fonte de dados sobre a evolução do software, automaticamente.

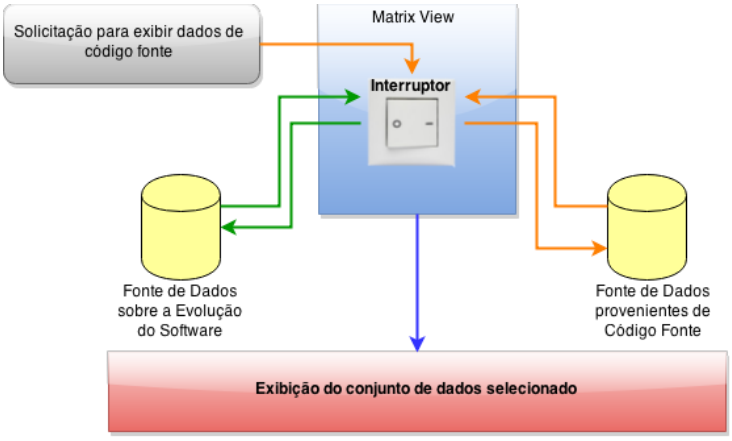

Figura 3.35: Ilustração de um Interruptor

A atividade Desenvolvimento Parte 2 (Figura 3.34) termina com a integração entre as ferramentas *GiveMe Views*, AIMV, *Sourceminer* e visualizações dando início a integração das ferramentas que compõem a *GiveMe Infra*. A Figura 3.36 mostra a evolução do diagrama de dependências da Figura 3.33 após a integração realizada no Desenvolvimento Parte 2.

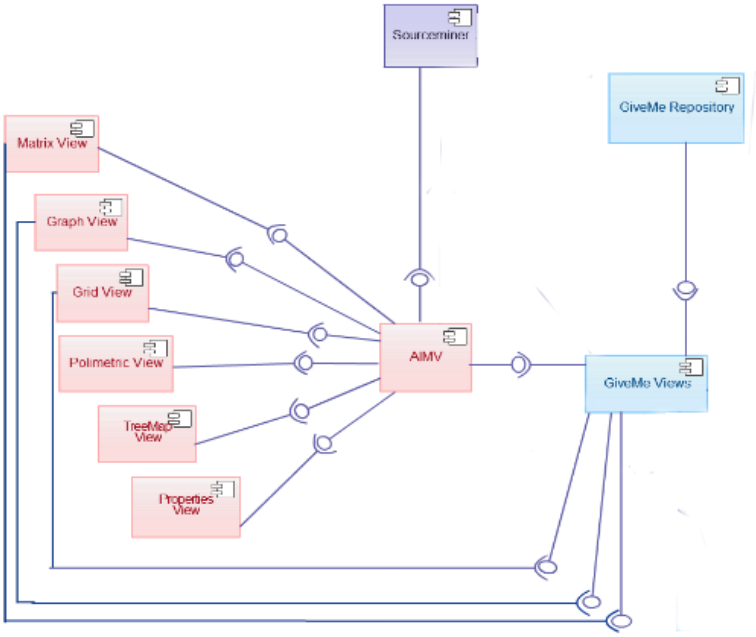

Figura 3.36: Primeira evolução do diagrama de dependências

Neste ponto do desenvolvimento da solução, tem-se uma infraestrutura capaz de apoiar atividades de manutenção e evolução de software, mas ainda sem contar com recursos de colaboração que suporte equipes geograficamente distribuídas. Nesse sentido foi iniciada uma parceria com a Equipe Collaborative Sourceminer (Figura 3.32), com o objetivo de disponibilizar o *plugin Collaborative Sourceminer*, que possui os recursos de colaboração. Isto permitiu o início do Desenvolvimento Parte 3 (Figura 3.32), que pode ser visto de forma expandida na Figura 3.37.

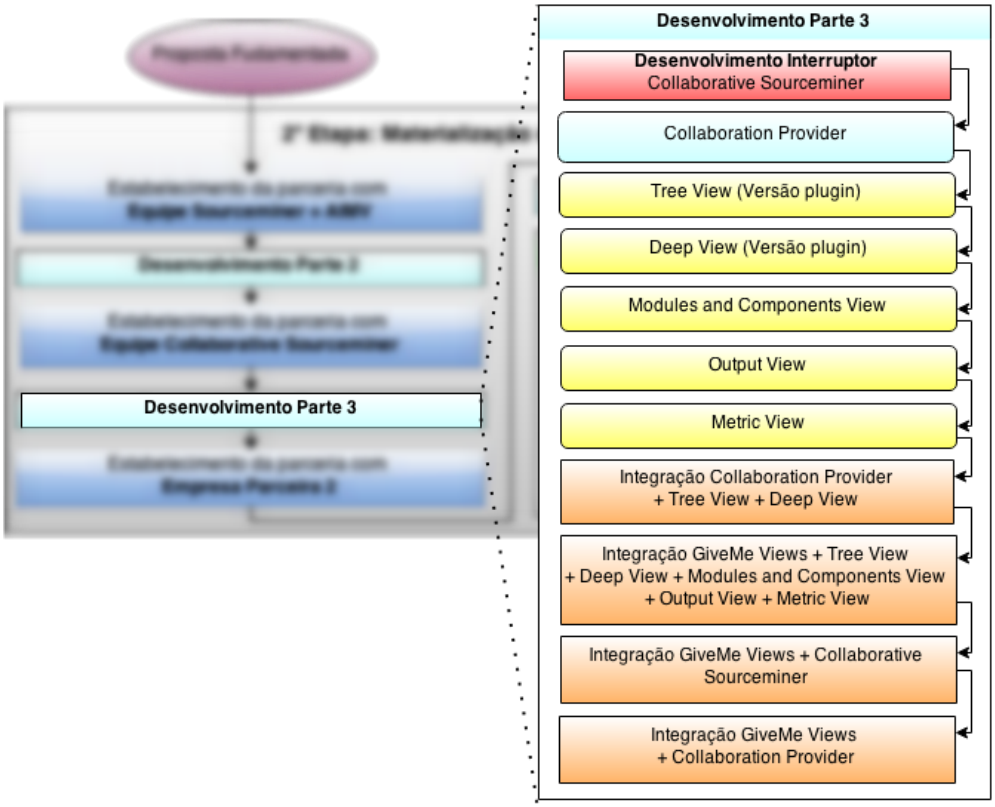

Figura 3.37: Atividade Desenvolvimento Parte 3 expandida

A primeira atividade do Desenvolvimento Parte 3 se deu com a definição de um interruptor para o *plugin Collaborative Sourceminer*, para que o mesmo passasse a trabalhar com dados de duas fontes de dados diferentes, assim com todos os outros *plugins* que receberam interruptores. Já o desenvolvimento do *Collaboration Provider* permitiu a criação de uma interface simples para acesso a recursos de colaboração.

A partir desse momento, deu-se início ao desenvolvimento das visualizações específicas para a *GiveMe Infra*, como *Tree View*, *Deep View*, *Modules and Components View*, *Output View* e *Metric View*. Posteriormente as atividades de integração foram realizadas. A Figura 3.38 mostra a segunda evolução do diagrama de dependências da Figura 3.33, já contemplando as atividades do Desenvolvimento Parte 3 (Figura 3.37).

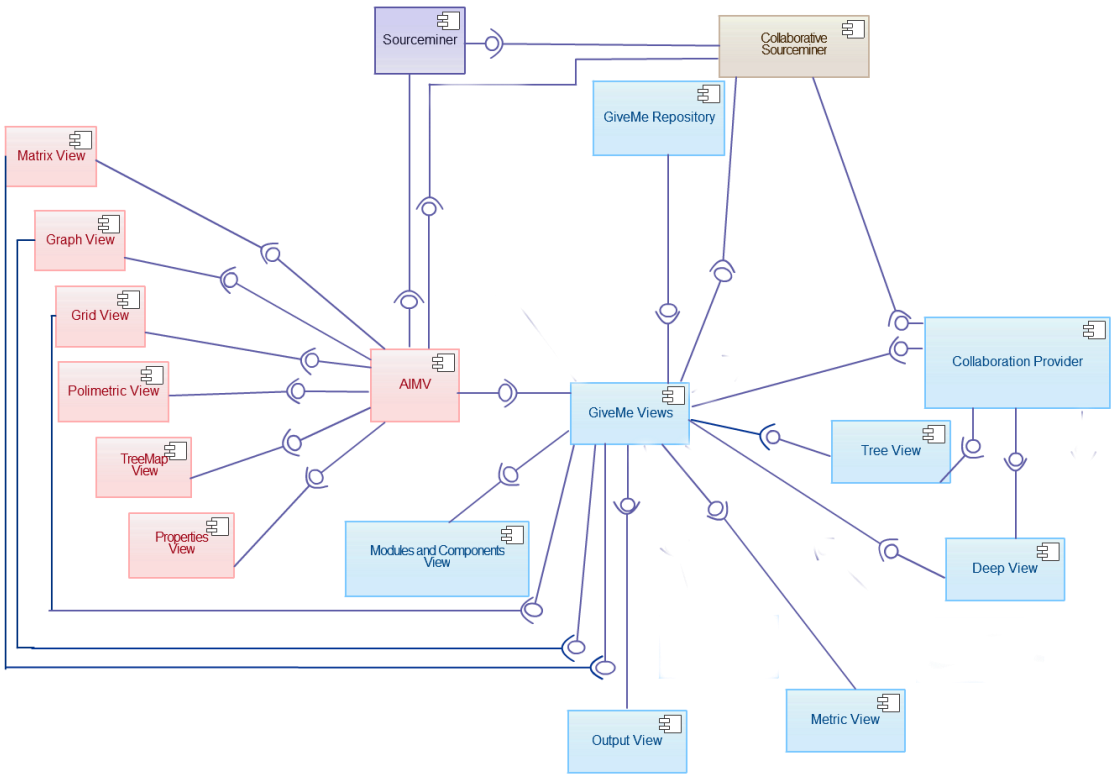

Figura 3.38: Segunda evolução do diagrama de dependências

Concluída a atividade Desenvolvimento Parte 3 (Figura 3.32) deu-se início a uma nova parceria com outra empresa de desenvolvimento de software para gestão comercial, que também por questões de confidencialidade será chamada aqui de Empresa Parceira 2. Através dessa parceria conseguiu-se o acesso a um repositório de código fonte GIT e um repositório de solicitações de mudanças gerenciado pela ferramenta *Mantis*. Esta parceria permitiu testar a *GiveMe Infra* sobre repositórios diferentes dos disponibilizados pela Empresa Parceira 1.

Após o estabelecimento da parceria com a Empresa Parceira 2, foi iniciada a última etapa do desenvolvimento da solução apresentada neste trabalho. A Figura 3.39 mostra a atividade Desenvolvimento Parte 4 de forma expandida.

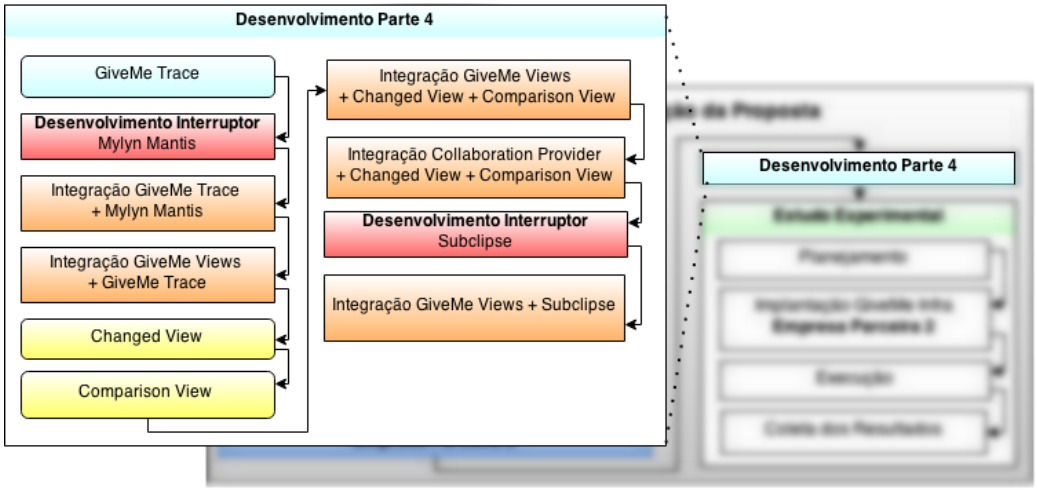

Figura 3.39: Atividade Desenvolvimento Parte 4 expandida

A primeira atividade foi o desenvolvimento do *GiveMe Trace*, que é resultado de um trabalho de graduação iniciado e finalizado durante o período de desenvolvimento deste trabalho. *GiveMe Trace* foi planejada para atuar em uma limitação que a ferramenta *GiveMe Views* possuía, como descrito na seção 3.1. Com o desenvolvimento do *GiveMe Trace*, *GiveMe Views* passou a fornecer indicações em um nível de granularidade mais baixo, tornando-se capaz de indicar métodos de uma classe que poderiam ser impactados quando um dado método fosse ser alterado.

Com o desenvolvimento do *GiveMe Trace* tornou-se viável o desenvolvimento das visualizações *Changed View* e *Comparison View* (Figura 3.39), que posteriormente foram integradas com o *GiveMe Views* e com o *Collaboration Provider* (para acessar os recursos de colaboração). Finalmente, foi desenvolvido um interruptor para o *plugin Subclipse* para permitir que, antes de uma alteração ser enviada a um repositório de controle de versão, o desenvolvedor possa ser notificado, caso necessário, de que há alterações a nível de código que foram indicadas pela *GiveMe Infra*, mas que não foram efetuadas por ele.

Após a finalização do desenvolvimento da solução proposta para este trabalho foi conduzido o estudo experimental (Figura 3.32), que pode ser visto em detalhes no capítulo Avaliação da Solução. Já o diagrama de dependências apresentado na Figura 3.38 sofreu uma nova evolução após a atividade Desenvolvimento Parte 4, ficando tal como pode ser visto na Figura 3.40.

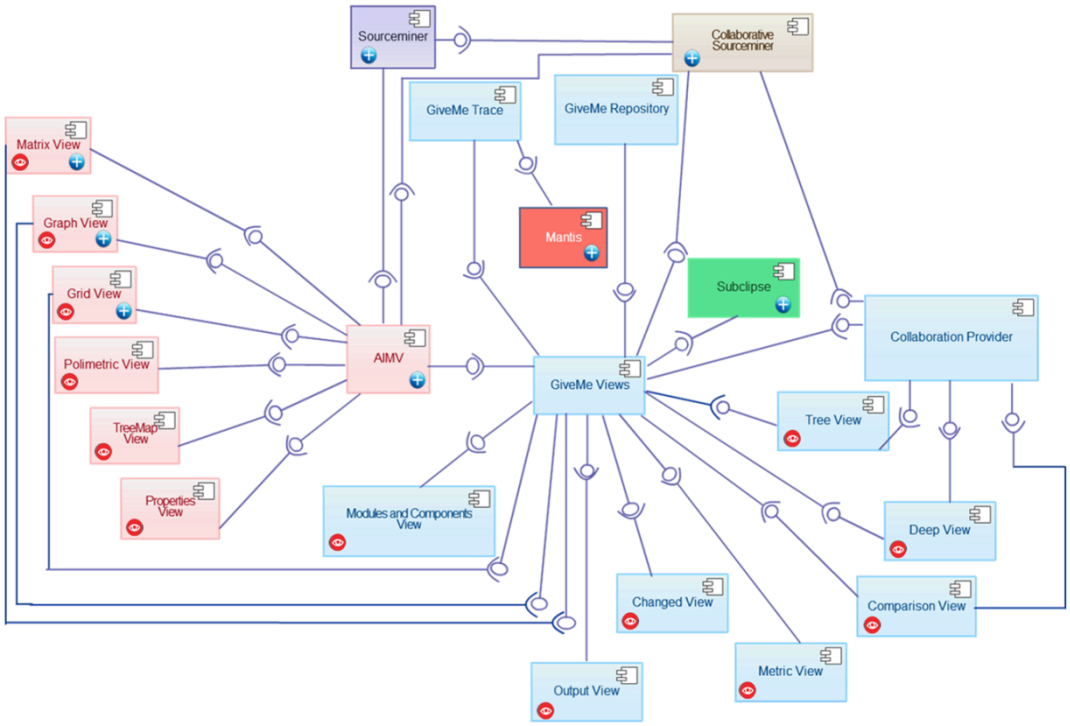

Figura 3.40: Terceira e última evolução do diagrama de dependências

O grupo das ferramentas em azul (Figura 3.40) engloba somente os *plugins* que foram desenvolvidos neste trabalho. Os demais grupos foram desenvolvidos por terceiros. *Plugins* que possuem um ícone vermelho e redondo são visualizações. Já os que contêm um ícone com sinal de mais (+) são todos os que tiveram que ser modificados (inserção de interruptores, por exemplo) para se serem integrados a infraestrutura. Ao todo *GiveMe Infra* possui 22 *plugins* integrados com o objetivo de resolver o problema geral deste trabalho.

3.4 CONSIDERAÇÕES FINAIS DO CAPÍTULO

Este capítulo visou apresentar a solução desenvolvida com base no problema encontrado na Empresa Parceira 1. Vale destacar aqui que a *GiveMe Infra* é uma tecnologia flexível que pode ser usada em outras empresas desde que a mesma possua dados históricos de solicitações de mudanças e repositórios SVN ou GIT. Atualmente a *GiveMe Infra* suporta análise de dados históricos extraídos do *Service Center* (Empresa Parceira 1) e do *Mantis*, mas outros *drivers* para suportar análise de dados históricos de

outras bases estão sendo desenvolvidos, como para a base do *Redmine* (REDMINE, 2013) e do *Bugzilla* (BUGZILLA, 2013), dentre outras.

Foi desenvolvido um site para este projeto[4] onde serão disponibilizados todos os tutoriais e manuais das ferramentas que compõem a *GiveMe Infra*, bem como *links* para os sites dos projetos que são parceiros deste. A ideia foi apresentar somente as ferramentas e os principais recursos aqui e as informações técnicas serem disponibilizadas no site do projeto. O próximo capítulo visa apresentar a avaliação da solução apresentada neste capítulo. Ela foi avaliada junto a um projeto real de uma empresa de desenvolvimento de software.

[4] http://www.givemeinfra.com.br.

4. AVALIAÇÃO DA SOLUÇÃO

Este capítulo apresenta o estudo experimental executado durante o desenvolvimento deste trabalho. Foi realizado com o apoio da Empresa Parceira 2 e corresponde às etapas (iv) a (viii) da metodologia definida no capítulo de introdução deste trabalho.

Este estudo experimental tem por objetivo colocar *GiveMe Infra* a prova. O objetivo é verificar sua viabilidade de uso em um contexto real de manutenção, através da análise de uma atividade de manutenção e evolução, desenvolvida no ambiente de uma empresa de desenvolvimento de software, a Empresa Parceira 2.

Em resumo, será analisada, com o apoio da *GiveMe Infra*, uma atividade de manutenção e evolução realizada por profissionais de TI, onde modificações no código de um projeto real foram realizadas. A ideia é verificar se a infraestrutura é capaz de indicar as mesmas modificações conduzidas pelos profissionais de TI da empresa parceira.

4.1 PLANEJAMENTO DO ESTUDO EXPERIMENTAL

A atividade de manutenção e evolução realizada pelos profissionais de TI da Empresa Parceira 2, foi sobre o *SpedFiscal*. Trata-se de um software que atua na extração de dados e montagem de arquivos que contêm informações de interesse da Receita Federal do Brasil e de demais órgãos envolvidos na prestação de contas relativas a impostos sobre operações realizadas pelo contribuinte, como operações de compra e venda na indústria e comércio.

O conjunto de arquivos gerados pelo software *SpedFiscal* seguem algumas normas e padrões, pois serão assinados digitalmente e enviados, via Internet, para outro sistema, chamado *Sped* (Sistema Público de Escrituração Digital) (SPED, 2015) da Receita Federal do Brasil. Esse, por sua vez, realizará a validação dos arquivos recebidos e as demais operações que envolvem a prestação de contas.

Os padrões, aos quais os arquivos de prestação de contas gerados pelo *SpedFiscal* estão sujeitos, são fornecidos pela Receita Federal através de manuais que ditam os formatos em que as informações devem ser geradas, bem como devem ser organizadas no arquivo. Cada nova demanda constitui o que é chamado de novo Bloco, portanto, a disponibilização de novo manual para implementação de um novo Bloco representará a necessidade de implementar novas funcionalidades, ou adaptar as existentes, do projeto *SpedFiscal* da Empresa Parceira 2.

A atividade de manutenção e evolução realizada pela equipe de profissionais da Empresa Parceira 2 surgiu após a Receita Federal disponibilizar os padrões para implementação do Bloco K (bloco de prestação de contas sobre operações de entrada e saída de mercadorias no comércio). O manual que dita as regras do Bloco K está disponível para *download* no site da receita (SPED MANUAL, 2015). Sua implementação foi prevista para a versão 47 do *SpedFiscal*.

4.1.1 Contexto do Estudo Experimental

Trata-se de um estudo controlado, que visa caracterizar a efetividade da *GiveMe Infra* no apoio a atividades de manutenção e evolução de software. Para tal, deseja-se estabelecer um comparativo entre (i) as decisões tomadas pela equipe de TI da Empresa Parceira 2 no desenvolvimento do Bloco K (implementado na versão 47) do *SpedFiscal* e (ii) as indicações de alterações a serem realizadas no código fonte, fornecidas pela *GiveMe Infra*, para o mesmo cenário: desenvolvimento do Bloco K. Espera-se verificar se a infraestrutura é capaz de indicar as mesmas alterações que foram realizadas no projeto *SpedFiscal* para a implementação do Bloco K, na sua versão 47, ao analisar os dados históricos de todo o ciclo de vida do software, que compreende as versões 1 a 46 (versões que antecedem o desenvolvimento do Bloco K).

4.1.2 Objetivos e questões de pesquisa

Seguindo o modelo GQM (*Goal – Question − Metric*) foram definidos os objetivos deste estudo experimental e em seguida as questões de pesquisa juntamente com a hipótese. Posteriormente, métricas foram definidas para cada uma das questões de pesquisa, tal como pode ser visto na Figura 4.41.

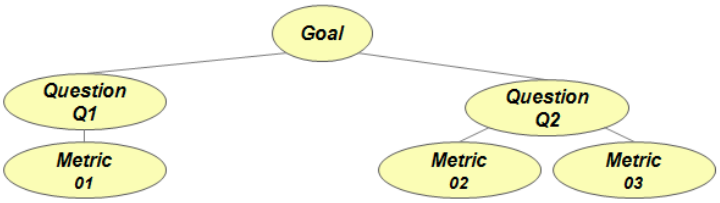

Figura 4.41: Hierarquia do modelo GQM.

Goal (Objetivo): Caracterizar a efetividade do apoio dado pela *GiveMe Infra* em atividades de manutenção e evolução de software ao comparar com atividades de manutenção e evolução realizadas em um contexto real de evolução de software. Seguindo o modelo GQM, tem-se: "Analisar o uso da *GiveMe Infra* com a finalidade de caracterizar sua efetividade com respeito à capacidade de apoiar atividades de manutenção e evolução de software em comparação com atividades de manutenção e evolução realizadas em ambiente real de evolução de software no contexto de atividades de manutenção e evolução de software".

Questions (Questões):

- Q1: o uso da *GiveMe Infra* provê meios de identificar quais módulos/componentes podem ser impactados, mediante a necessidade de se alterar um módulo/componente específico de um dado projeto? Objetivo: verificar se a *GiveMe Infra* é capaz de apoiar a implementação de novas funcionalidades, baseando-se em indicações de análises realizadas sobre dados históricos de versões anteriores do software;
- Q2: até que ponto *GiveMe Infra* apoia os desenvolvedores nas atividades de manutenção e evolução de software futuras? Objetivo da questão de pesquisa: verificar como se dá o apoio a atividades de manutenção e evolução de software usando a *GiveMe Infra*.

  Com base nos objetivos, foi definida a seguinte hipótese:

- H nula: a utilização da *GiveMe Infra* não é capaz de apoiar a tomada de decisões no contexto da implementação realizada pela equipe de TI da Empresa Parceira 2;

- H alternativa: a utilização da *GiveMe Infra* é capaz de apoiar a tomada de decisões no contexto da implementação realizada pela equipe de TI da Empresa Parceira 2.

Metrics (Métricas): foram definidas com base nas questões de pesquisa Q1 e Q2, sendo a Métrica 01 referente à questão Q1, e as Métricas 02 e 03 referentes à questão Q2:

- Métrica 01: porcentagem de acerto obtida através do somatório do número de impactos causados em métodos com as alterações no código fonte realizadas pela Empresa Parceira 2 em comparação com o número correto de indicações impactos fornecidos pela *GiveMe Infra;*
- Métrica 02: número de indicações extras, refere-se ao conjunto de indicações de impacto em métodos que pode ser fornecido pela *GiveMe Infra* mas que não refletem alterações realizadas pela Empresa Parceira 2;
- Métrica 03: número indicações não fornecidas pela *GiveMe Infra* mas que resultaram em alterações no software analisado.

Para a realização do cálculo das métricas e visando responder as questões de pesquisa, estão previstas a utilização de teorias estatísticas.

Teorias Estatísticas do Estudo Experimental: as seguintes teorias estatísticas serão utilizadas na execução do estudo experimental: análise de correlação de dados (LEVIN, et al., 2012). Permite estabelecer a correlação existente entre diferentes conjuntos de dados. Neste estudo experimental será útil para definir a correlação existente entre o aumento do número de indicações de alterações fornecidas pela *GiveMe Infra* com o aumento do número de indicações corretas fornecidas, se comparado as alterações realizadas pela equipe de TI da Empresa Parceira 2.

- distribuição de frequência simples e conjunta (MEYER, 1983): permitirá descobrir quais métodos sofreram impacto nas alterações realizadas no código fonte por parte da equipe de TI da Empresa Parceira 2.

Com relação às hipóteses, objetivando verificar qual será rejeitada, serão consideradas as variáveis dependentes e independentes.

Variáveis Independentes: as variáveis independentes são as versões do *SpedFiscal* a serem analisadas, que possuem dois tratamentos: (i) Versões 1 a 46 do *SpedFiscal* (correspondem ao conjunto de dados históricos que serão usados pela *GiveMe Infra* para fornecer as indicações) e (ii) Versão 47 do *SpedFiscal* (versão a ser

analisada). O delineamento deste estudo experimental é de um fator e dois tratamentos, sendo o fator as versões do software alvo de manutenção na implementação do Bloco K.

Tratamento: Será utilizada a seguinte combinação de tratamentos neste estudo experimental: (i) versões anteriores ao desenvolvimento do Bloco K (compreende as versões 1 a 46) e (ii) última versão do SpedFiscal (versão 47), que corresponde a implementação do Bloco K.

Variáveis Dependentes: a completude da atividade de manutenção realizada pela Empresa Parceira 2, que é uma medida da eficácia do apoio prestado pela *GiveMe Infra*.

4.1.3 Ferramenta Base

A ferramenta base para este estudo experimental é a *GiveMe Infra*. O objetivo é utilizar seus recursos para indicar alterações a nível de código fonte, no projeto *SpedFiscal*, com o intuito de estabelecer um comparativo com o que foi alterado pela equipe de TI da Empresa Parceira 2 na implementação do Bloco K.

Objeto: O objeto deste estudo será o software *SpedFiscal* e seus dados históricos. Trata-se de um software ERP contábil para empresas, desenvolvido e licenciado pela Empresa Parceira 2.

4.1.4 Sujeitos e equipes

Neste estudo experimental, que é de verificação de manutenções realizadas anteriormente em um contexto real, não serão selecionados sujeitos e equipes na execução, além do pesquisador responsável por este estudo. É importante ressaltar que atividades de verificações de manutenções em outros contextos podem implicar na necessidade da escolha de sujeitos, diferentemente deste.

Atividades: as atividades deste estudo experimental se iniciam após o encerramento da implementação do Bloco K, por parte da equipe dos profissionais da área de TI. O diagrama de atividades da UML (Figura 4.42) mostra o fluxo das atividades:

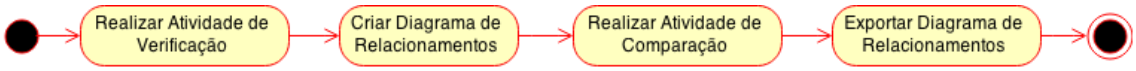

Figura 4.42: Diagrama das atividades

- realizar atividade de verificação: análise das manutenções executadas no código fonte do SpedFiscal pela equipe dos profissionais da área de TI e catalogação dos métodos afetados. Isso será possível analisando os históricos das alterações conduzidas, utilizando para tal as indicações dos diffs (SVN BOOK, 2015) fornecidos pela ferramenta Tortoise SVN (TORTOISE, 2014). O uso desses recursos é necessário para que o resultado da verificação seja imparcial, não influenciado por nenhuma das ferramentas que compõem a GiveMe Infra;
- criar diagrama de relacionamentos explorados pela equipe dos profissionais da área de TI na implementação do Bloco K. Trata-se de uma representação em forma de grafos constituída por vértices (representando métodos alterados) e arestas (representando os relacionamentos entre eles). Com isso espera-se criar grafos que mostrem quais métodos foram alterados na implementação do Bloco K e quais métodos foram impactados com as alterações; os relacionamentos serão calculados utilizando as mesmas teorias usadas na concepção do motor SAE, que são as teorias estatísticas de distribuição de frequência discutidas em (MEYER, 1983);
- realizar atividade de comparação: análise, via *GiveMe Infra*, de cada um dos métodos alterados pela equipe dos profissionais da área de TI, na implementação do Bloco K. O objetivo dessa atividade é gerar um diagrama de relacionamentos com as indicações da *GiveMe Infra.* Essa atividade tem por objetivo criar grafos que mostrem as indicações de alterações previstas para a versão 47 do *SpedFiscal*, considerando verificações realizadas em cada um dos métodos de fato alterados na versão 47;
- explorar diagrama de relacionamentos que mostra as indicações fornecidas pela *GiveMe Infra* com base na análise dos métodos que foram de fato alterados na implementação do Bloco K. Nesse caso, o diagrama será fornecido pela visualização *TreeView* pertencente a *GiveMe Infra.*

4.1.5 Operacionalização

O primeiro passo deste estudo experimental consistiu em analisar a atividade de manutenção e evolução realizada pela Empresa Parceira 2 objetivando a efetivação da implementação do Bloco K no projeto *SpedFiscal*. Para tal, foi necessário acesso ao repositório de solicitações de mudanças da empresa. Posteriormente, foi conduzida a atividade de verificação. Isso foi possível com a utilização da ferramenta *Tortoise SVN* e os arquivos *diff* fornecidos por ela. O uso desses recursos foi necessário para que o resultado da verificação seja imparcial, não influenciado por nenhuma das ferramentas que compõem a *GiveMe Infra.*

Após a catalogação dos métodos alterados na implementação do Bloco K na versão 47, foi criada uma matriz de relacionamentos com base nas mesmas teorias usadas na concepção do motor SAE, que são as teorias estatísticas de distribuição de frequência discutidas em (MEYER, 1983). Foi possível estabelecer as relações de impactos entre os métodos alterados para a implementação do Bloco K. De posse dessas informações foi possível responder perguntas como: quantos métodos foram impactados quando um método foi alterado na implementação do Bloco K? Nesse sentido, o próximo passo consistiu na comparação com as indicações fornecidas pela *GiveMe Infra* para os mesmos métodos alterados pela equipe de TI da Empresa Parceira 2 (atividade de comparação). Cada uma das comparações seguiram o modelo de execução definido a seguir, que mostra os passos necessários para a realização da atividade de comparação:

- Modelo de execução: quando a equipe de TI da Empresa Parceira 2 conduziu alterações em um determinado método foram conduzidas modificações também em outros métodos. Verifica-se na *GiveMe Infra* os impactos de uma alteração no mesmo determinado método e analisa-se quais outros podem ser impactados. Estabelece-se um comparativo entre o que foi modificado na prática, na versão 47 do *SpedFiscal*, e o que foi indicado pela *GiveMe Infra*.

Os resultados das atividades de verificação e comparação foram coletados e utilizados para responder as questões de pesquisa estabelecidas, para o cálculo das métricas definidas e também para a avaliação da hipótese.

4.1.6 Coleta de dados

A coleta de dados no estudo experimental se dará através da utilização de planilhas eletrônicas que serão utilizadas para registro dos dados obtidos na atividade de verificação. Todos os métodos alterados pela equipe de TI da Empresa Parceira 2 na implementação do Bloco K serão registrados em uma planilha eletrônica onde ficarão disponíveis para análise.

Além disso, também não são necessárias coletas de informações como tempo de duração das verificações.

Os dados de corretude, que são dados usados na validação de informações a serem analisadas, serão fornecidos pelas ferramentas *Tortoise* SVN e pela própria *GiveMe Infra*, e serão utilizados na validação dos dados do estudo experimental. Como exemplo, tem-se a lista de métodos alterados na implementação do Bloco K. Os dados que validam essa lista são fornecidos pelos arquivos *diff* da ferramenta *Tortoise* SVN.

Como o planejamento do estudo experimental definido, bem como as hipóteses e as questões de pesquisa, foi realizada a execução do estudo experimental seguindo o protocolo. Os resultados são descritos na subseção a seguir.

4.2 EXECUÇÃO DO ESTUDO EXPERIMENTAL

O primeiro passo foi acessar o repositório de solicitações de mudanças da Empresa Parceira 2 com o intuito de analisar a atividade de manutenção executada na implementação do Bloco K (que originou a versão 47 do *SpedFiscal*) e obter conhecimento sobre ela. A Figura 4.43 mostra uma captura de tela exemplificando-a.

Principal | Registro de Mudanças | Planejamento | Histórico de Versão

Caso # Ir para

Tempo Gasto: Dias 04 Horas 23 Minutos 09 Segundos 59

Ver Detalhes do Caso [Ir para as Anotações] [Histórico do Caso] [Imprimir]

Núm	Categoria	Visibilidade	Data de Envio	Última Atualização
0010821	Fiscal	público	2014-08-04 15:06	2015-01-08 10:55

Relator	[redacted]			
Prioridade	normal	**Gravidade**	recurso	**Freqüência**
Estado	resolvido	**Resolução**	corrigido	
Plataforma		**SO**		**Versão SO**
Versão do Produto	2.4.17-1 (Costa Rica)			

[redacted]

Resumo	0010821: Implementação - Exportação Sped Fiscal - Inclusão do Registro K
Descrição	Módulo Fiscal – Na rotina – Arquivos Digitais – Exportação – Sped Fiscal, implementar no sistema a geração do registro bloco K. Este bloco se destina a prestar informações mensais da produção e respectivo consumo de insumos, bem como do estoque escriturado, relativos aos estabelecimentos industriais ou a eles equiparados pela legislação federal e pelos atacadistas, podendo, a critério do Fisco, ser exigido de estabelecimento de contribuintes de outros setores (conforme § 4ºdo art. 63 do Convênio s/número, de 1970) que entrará em vigor na validação do ano base 2015.
Passos para Reproduzir	1> Arquivos Digitais 2> Exportação 3> Sped Fiscal 4> Data Inicial: 01/06/2014 – Data Final: 30/06/2014
Marcadores	Nenhum marcador aplicado.
Alteracao Interface	Sim
Complexidade	Grande
email-enviado	Resolvido

Figura 4.43: Captura de tela da atividade de manutenção que originou a versão 47 do SpedFiscal, referente ao Bloco K

As modificações no código fonte do SpedFiscal para a implementação do Bloco K foram analisadas utilizando a ferramenta cliente SVN e os arquivos *diff* resultantes das modificações realizadas, visando criar uma lista de todos os métodos que foram impactados. A Figura 4.44 mostra um exemplo de identificação de um dos métodos impactados. Nela é possível ver o *diff* das modificações na classe SPEDFiscalBlocoDAO, para a versão 47 do SpedFiscal. Vale resaltar que, em um estudo experimental onde o número de linhas alteradas seja suficientemente grande, analisar manualmente o arquivo *diff* pode não ser o mais indicado. Nesse caso, deve-se buscar por outra solução que permita identificar automaticamente os métodos alterados.

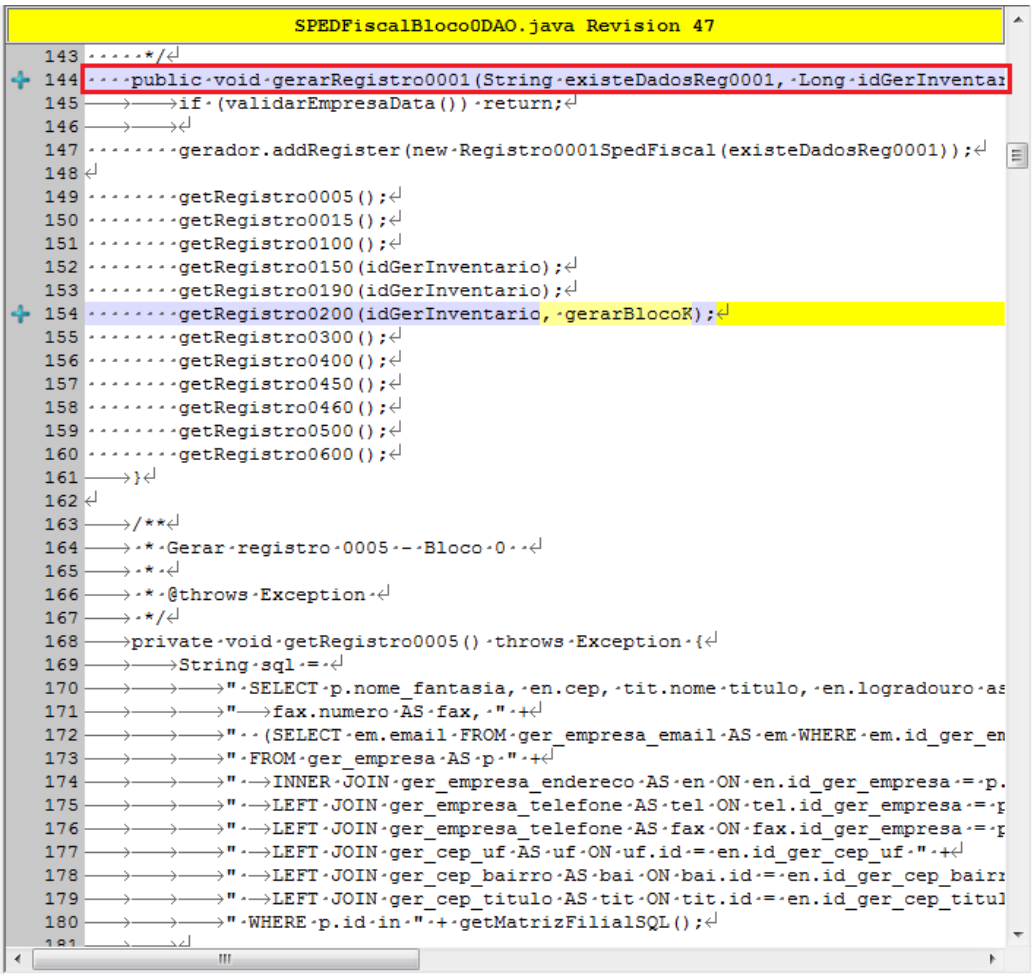

Figura 4.44: Exemplo de identificação de um dos métodos impactados usando o *diff*

Após a identificação de todos os métodos impactados foram estabelecidas as relações entre os métodos que foram impactados em conjunto, seguindo o princípio da distribuição de frequência conjunta proposta por (MEYER, 1983). Em resumo, num total foram identificados 5 métodos alterados durante a implementação do Bloco K. A Figura 4.45 exibe os cinco métodos alterados e os métodos que eles impactaram, dada a distribuição de frequência calculada.

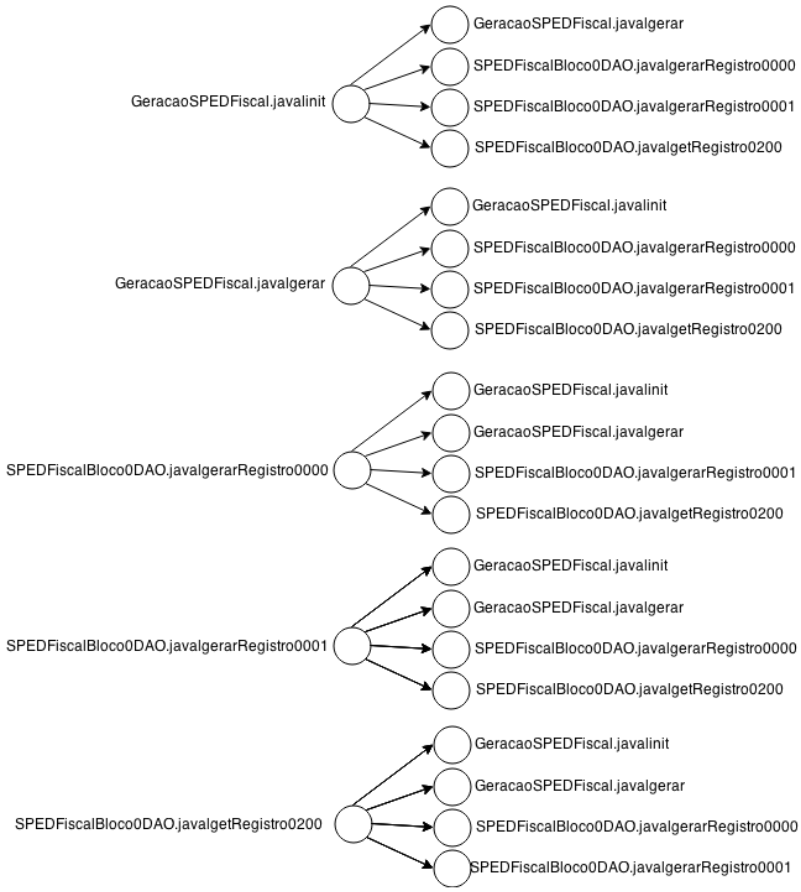

Figura 4.45: Relacionamentos entre métodos.

Neste ponto do estudo experimental tem-se a análise de todos os métodos que foram impactados durante o processo de implementação do Bloco K, bem como os que eles possivelmente impactaram, dados os cálculos estatísticos. O próximo passo consiste na execução da Atividade de Comparação utilizando a *GiveMe Infra*, que mostrou que, quando a equipe de TI da Empresa Parceira 2 conduziu alterações no método GeracaoSPEDFiscal.java|init (Figura 4.45) foram conduzidas modificações, por exemplo, nos métodos SPEDFiscalBloco0DAO.java|gerarRegistro0001 e SPEDFiscalBloco0DAO.java|getRegistro0200 (Figura 4.45). A verificação na *GiveMe Infra* do impacto de uma alteração no método GeracaoSPEDFiscal.java|init (considerando dados históricos das versões anteriores, 1 a 46) mostrará quais dos métodos de fato impactados foram indicados pela *GiveMe Infra* e quais dentre os indicados não foram alterados na implementação do Bloco K. A Figura 4.46 foi gerada com as exportações dos diagramas de relacionamentos obtidos após as verificações de impactos realizadas com a *GiveMe Infra*. Os métodos nela contidos são os mesmos cujos impactos foram apresentados na Figura 4.45, e os métodos marcados por um

quadro são os que correspondem a modificações realizadas de fato pela equipe da Empresa Parceira 2, na versão 47.

Figura 4.46: Verificação de impacto com a *GiveMe Infra*

Ao analisar a Figura 4.46 percebe-se que não foi possível verificar na *GiveMe Infra* o impacto que alterações nos métodos SPEDFiscalBloco0DAO.java|gerarRegistro0001 e SPEDFiscalBloco0DAO.java| gerarRegistro0000 podem ocasionar. Isso ocorreu porque a *GiveMe Infra* não foi capaz de indicar possíveis pontos que podem ser impactados quando este for alterado. O motivo é que o conjunto dos dados históricos, das versões 1 a 46, não apresentaram nenhuma manutenção nos métodos em questão ao longo do ciclo de vida do projeto *SpedFiscal*. Uma investigação no repositório de versionamento de código fonte do projeto mostrou que a primeira manutenção realizada nesse método foi após o início da atividade de manutenção que levou a implementação do Bloco K, ou seja, na versão 47, justificando a inexistência de indicações. Esse fato caracteriza uma limitação e ao mesmo tempo uma característica da *GiveMe Infra*: caso o conjunto de dados históricos a ser analisado não contemple alterações em dado método, ao longo de todo o ciclo de vida do software, não será possível estimar os impactos de uma alteração nele.

Entretanto, esse problema pode ser minimizado a medida que, ao longo tempo, novos registros de manutenções sejam armazenados.

Outro dado a ser considerado na Figura 4.46 refere-se a porcentagem calculada para os métodos marcados em vermelho. Apresentaram a maior porcentagem dentre todos os demais que podem ser impactados (100% e 16,67%). Ao mesmo tempo em que isso é um indicativo de que as chances deles serem impactados (como de fato foram na implementação do Bloco K) são as maiores dentre todos, a mesma figura mostra que outros métodos têm chances iguais estatisticamente de serem impactados, mas que na prática não culminaram em alterações. Isso pode ser um indício de que alterações deixaram de ser realizadas ou verificadas pela equipe de TI da Empresa Parceira 2.

A comparação entre os métodos que foram apresentados na Figura 4.45 (contexto real) com os que foram apresentados na Figura 4.46 (Atividade de Comparação) mostra que o número de métodos indicados com a *GiveMe Infra* é maior que o número de métodos alterados na versão 47 do *SpedFiscal* nos casos dos métodos GeracaoSPEDFiscal.java|init, GeracaoSPEDFiscal.java|gerar e SPEDFiscalBloco0DAO.java|getRegistro0200, sendo o último o que sofreu maior variação (2 indicações de métodos realmente impactados em um total de 22 indicados na verificação). A Tabela 2 apresenta um resumo de todas as informações geradas na Atividade de Comparação. Nela é possível observar que o número de impactos causados em métodos mediante a implementação do Bloco K é menor que o número de indicações de impacto, mediante a comparação realizada pela *GiveMe Infra*. Mesmo assim, não é possível afirmar que as indicações a mais representam métodos que deixaram de ser mantidos ou que representam indicações desnecessárias. Para tal, seria fundamental que a equipe que efetuou a implementação do Bloco K revisasse a implementação considerando agora as indicações a mais fornecidas pela *GiveMe Infra*. Na Tabela 2 é utilizado o conceito de métodos impactados em referência aos métodos que aparecem como afetados mediante a alterações em outros métodos. Como exemplo, tem-se o método GeracaoSPEDFiscal.java|gerar (Figura 4.45) que foi impactado por alterações no método GeracaoSPEDFiscal.java|init.

Tabela 2: Resumo final dos dados da verificação realizada

	Número de métodos impactados na implementação do Bloco K	∑ [número de métodos impactados]
Dados Reais	5	20
Dados da Atividade de Verificação com a *GiveMe Infra*	5	34

A Tabela 2 mostra que o somatório de todos os impactos de manutenção realizados na implementação do Bloco K é 20. Em termos práticos, isso quer dizer que, a cada método modificado, foram ou não conduzidas modificações nos métodos que possuem algum tipo de relacionamento com ele. O somatório de todas as modificações conduzidas é 20 (Atividade de Verificação). Já o somatório de todas as indicações - calculadas pela *GiveMe Infra* (Atividade de Comparação) totalizam 34, o que na prática significa que o número de métodos que deveriam ser considerados na manutenção é aproximadamente 59% maior do que o total de métodos impactados na prática. Há também a possibilidade da equipe que efetuou a implementação do Bloco K ter analisado todos os pontos indicados pela *GiveMe Infra* mesmo sem terem tido suporte da infraestrutura, mas julgaram que não eram necessárias mais alterações do que as que foram realizadas. Caso estivessem utilizando a *GiveMe Infra* na implementação do Bloco K, os métodos verificados, mas sem necessidade real de modificação, poderiam ser marcados como ignorados. Isso é possível com o uso da visualização *Changed View*. A Figura 4.47 mostra um exemplo baseado nas verificações realizadas para este estudo experimental.

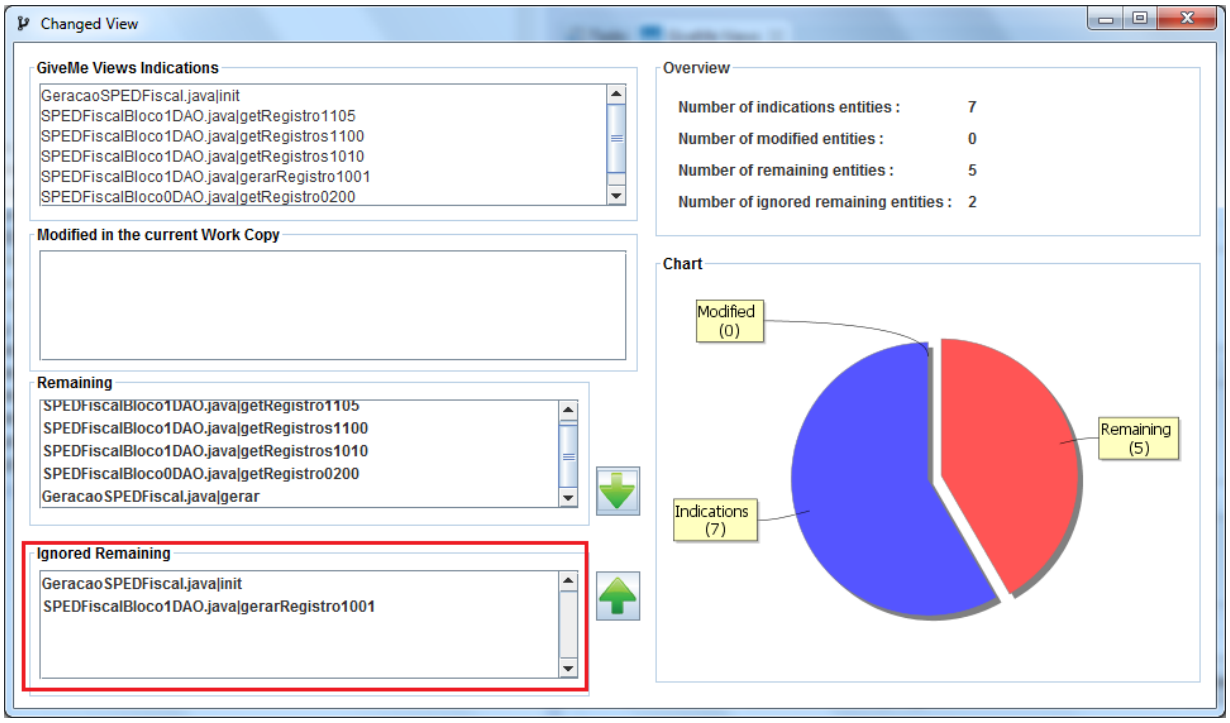

Figura 4.47: Ignorando métodos que não serão de fato impactados

Analisando os dados obtidos com a execução da Atividade de Comparação é possível concluir que a *GiveMe Infra* só não é capaz de indicar os métodos que de fato vão ser alterados em duas situações:

- quando o conjunto de dados históricos é pequeno, ou seja, não reflete um conjunto considerável de manutenções passadas de modo a proporcionar relevância estatística;
- quando no conjunto de dados históricos analisado não há nenhum indício histórico de ligação entre o método alterado e o que será impactado. Nesse caso a única solução é tentar utilizar outro conjunto de dados históricos, caso haja (ou aguardar novos dados e compor uma base histórica mais completa). Outro fator que deve ser considerado é que o método a ser analisado pode estar implementado em um projeto novo, não havendo, portanto dados históricos armazenados.

O próximo passo deste estudo experimental consiste nos cálculos das métricas previstas para as questões de pesquisa definidas no planejamento. Através dos cálculos será possível embasar as respostas das respectivas questões de pesquisa utilizando critérios matemáticos. A Tabela 3 mostra o resultado das métricas calculadas. Nela também é utilizado o conceito de métodos impactados em referência aos métodos que

aparecem como afetados mediante as alterações em outros métodos. Como exemplo, tem-se o método GeracaoSPEDFiscal.java|gerar (Figura 4.45) que foi impactado por alterações no método GeracaoSPEDFiscal.java|init.

Tabela 3: Métricas calculadas

	[∑ (número de métodos impactados)] - número de indicações extras	Métrica 1	Métrica 2	Métrica 3
Dados da Atividade de comparação, com *GiveMe Infra*	6/20	30%	28	14/20

Analisando os dados calculados para a Métrica 1 (porcentagem de acerto obtida através do somatório do número de impactos causados em métodos com as alterações no código fonte realizadas pela Empresa Parceira 2 em comparação com o número correto de indicações impactos fornecidos pela *GiveMe Infra*) é possível perceber que, dos 20 impactos causados em métodos na implementação do Bloco K, a *GiveMe Infra* foi capaz de indicar 6, o que corresponde a 30% do total. Na prática isto quer dizer que, se a equipe que implementou o Bloco K usasse a *GiveMe Infra* (analisando o conjunto de dados históricos das versões 1 a 46) para prever as possíveis alterações necessárias na implementação do Bloco K (que gerou a versão 47), teria êxito em 30% das modificações de fato necessárias. Entretanto, se a infraestrutura tivesse sido utilizada, os outros métodos que foram indicados, mas não foram alterados de fato, poderiam resultar em novas manutenções, aumentando a quantidade de métodos alterados na implementação do Bloco K.

A Métrica 2 (número de indicações extras, refere-se ao conjunto de indicações de impacto em métodos que pode ser fornecido pela *GiveMe Infra* mas que não refletem alterações realizadas pela Empresa Parceira 2) apresentada na Tabela 3 mostra que, o número de indicações extras fornecidas pela *GiveMe Infra* é de 28, ou seja, 28 indicações de alterações foram feitas pela *GiveMe Infra*, mas que não foram efetuadas pela equipe de TI da Empresa Parceira 2, por dois motivos: (ii) ou por não terem de fato considerado esses pontos ou (ii) por terem analisado esses impactos, mas descoberto

que na prática não eram necessários; o que em ambos os casos é um problema, pois o primeiro motivo pode levar a inserção de defeitos no software, que se não forem detectados nos testes ou pelo setor de qualidade, poderão chegar até o usuário final. Já o segundo motivo mostra que a *GiveMe Infra* não foi executada no seu melhor caso, que é de indicar somente os pontos que de fato caracterizam alterações necessárias.

Todas as análises realizadas até o presente momento visam dar sustentação às respostas das questões de pesquisa deste estudo experimental. A questão de pesquisa Q1 gira em torno do apoio da *GiveMe Infra* na tarefa de identificar quais pontos podem ser impactados, mediante a necessidade de se alterar um módulo/componente específico. Os resultados obtidos com o cálculo das métricas mostra que *GiveMe* Infra foi capaz de indicar 30% das verificações realizadas, o que quer dizer que, caso a equipe de TI da Empresa Parceira 2 tivesse utilizado a infraestrutura para auxiliar na tomada de decisões sobre o que deveria ser alterado na implementação do Bloco K, o apoio seria de 30%. É importante destacar que o esse valor não pode ser assumido como sendo medida de apoio para outros projetos e até mesmo para implementações de outras funcionalidades dentro do próprio *SpedFiscal.*

Para a questão de pesquisa Q2 foi investigado até que ponto *GiveMe Infra é* capaz de apoiar desenvolvedores nas atividades de manutenção e evolução de software, com o intuito de verificar se há diferenças entre as atividades de manutenção e evolução que visaram implementar o Bloco K e as verificações feitas na *GiveMe Infra*. Com o resultado da Métrica 2 não é possível afirmar que a *GiveMe Infra* apoiaria menos que a equipe de TI da Empresa Parceira 2 na implementação do Bloco K, dado que o fato de terem sido fornecidas 28 indicações extras de alterações pode ser um indício de que a manutenção realizada não foi eficiente na tarefa de alterar todos os métodos de fato necessários. Outro fator que impede afirmar que a *GiveMe Infra* apoiaria menos que a equipe de TI, mesmo indicando 30% dos impactos efetuados, está ligado ao conjunto de dados históricos utilizados: o apoio dado pode ser maior ou menor caso eles reflitam ou não de fato os relacionamentos existentes entre os componentes.

Ainda sobre a questão de pesquisa Q2, foi realizado o cálculo do Coeficiente da Correlação de *Pearson*, tal como descrito em (LEVIN, et al., 2012), para descobrir se à medida em que cresce o número de indicações fornecidas pela *GiveMe Infra* cresce também o número de indicações corretas sobre o que de fato foi efetuado na implementação do Bloco K? A Fórmula para o Cálculo é :

$$r = \frac{\Sigma(A)(B)}{\sqrt{\Sigma(A^2)\Sigma(B^2)}}$$

$$= SP/\sqrt{SQ(NIE)SQ(NIC)}$$

Coeficientes de correlação indicam intensidade e direção da correlação dos dados, expressos da seguinte forma:

- até -1,00: correlação negativa perfeita;
- até -0,60: correlação negativa forte;
- até -0,30: correlação negativa moderada;
- até -010: correlação negativa fraca;
- até 0,00: nenhuma correlação;
- até 010: correlação positiva fraca;
- até 0,30: correlação positiva moderada;
- até 0,60: correlação positiva forte;
- até 1,00: correlação positiva perfeita.

Em resumo, valores negativos indicam correlação negativa e valores positivos, indicam correlação positiva. Quanto mais próximo de -1,00 ou 1,00, maior a intensidade da correlação entre os dados.

Neste sentido, foi calculado o coeficiente da correlação de *Pearson* entre o número de indicações totais por método (NIT) e número de indicações corretas sobre o que de fato foi alterado (NIC). Considera-se a Tabela 4, que contém a lista de métodos alterados na implementação do Bloco K. Os dados do NIT e NIC foram obtidos com base na contagem do número de métodos impactados (Figura 4.46)

Tabela 4: Lista dos métodos alterados

métodos alterados	NIT	NIC
GeracaoSPEDFiscal.java|init	6	2
GeracaoSPEDFiscal.java|gerar	6	2
SPEDFiscalBloco0DAO.java|gerarRegistro0000	0	0
SPEDFiscalBloco0DAO.java|gerarRegistro0001	0	0
SPEDFiscalBloco0DAO.java|getRegistro0200	22	2

A Tabela 5 é complementar a Tabela 4, pois apresenta os cálculos iniciais do coeficiente da correlação, onde A, que é o valor calculado do desvio entre duas variáveis, é obtido pela subtração do NIT' do NIT. O NIT', por sua vez, é obtido através do cálculo da média do somatório dos valores de NIT. Já o valor de B é calculado pelo desvio entre duas variáveis, e obtido pela subtração do NIC' do NIC. O NIC' é obtido através do cálculo da média do somatório dos valores de NIC. A coluna 6 da Tabela 5 apresenta a multiplicação entre A e B, que representa o valor calculado dos desvios entre elas. Por último, o valor de SP é obtido através do somatório de todos os valores calculados na sexta coluna da Tabela 5.

Tabela 5: Cálculos do coeficiente da correlação

Métodos alterados	NIT	NIC	A = NIT - NIT'	B = NIC - NIC'	A * B
GeracaoSPEDFiscal.java|init	6	2	-0,8	0,8	-0,64
GeracaoSPEDFiscal.java|gerar	6	2	-0,8	0,8	-0,64
SPEDFiscalBloco0DAO.java|gerarRegistro0000	0	0	-6,8	-1,2	8,16
SPEDFiscalBloco0DAO.java|gerarRegistro0001	0	0	-6,8	-1,2	8,16
SPEDFiscalBloco0DAO.java|getRegistro0200	22	2	15,2	0,8	12,16
	∑ (NIE = 34) Média = NIE' = 6,8	∑ (NIC = 6) Média = NIC' = 1,2			SP = ∑ (A*B) = 27,2

Dado que o valor de SP é um valor positivo, isso indica que há uma associação dita positiva, entre NIT e NIC. Resta agora descobrir a intensidade da correlação entre eles. Considera-se então a Tabela 6.

Tabela 6: Cálculo da intensidade da correlação entre NIE e NIC

A^2	B^2
0,64	0,64
0,64	0,64
46,24	1,44
46,24	1,44
231,04	0,64
SQ(NIE) = $\sum(A^2) = 324,8$	SQ(NIC) = $\sum(B^2) = 4,8$

Nesse ponto têm-se todas as variáveis necessárias para a efetivação do cálculo da Correlação de *Pearson* (r):

$$r = 27{,}2/\sqrt{(324{,}8)(4{,}8)}$$
$$r = 27{,}2/39{,}48$$
$$r = +0{,}68$$

A Correlação de *Pearson* calculada mostra que o número de indicações totais por método (NIT) se relaciona positivamente de forma perfeita com número de indicações corretas sobre o que de fato foi alterado (NIC). Na prática isto significa que, para o caso da implementação do Bloco K, sempre que o número de indicações totais por método aumentar, aumentarão também as chances de se indicar mais métodos a serem alterados de fato.

Além das análises conduzidas, das métricas calculadas e das questões de pesquisa investigadas, foram definidas hipóteses para serem aceitas ou rejeitadas neste estudo experimental. A Hipótese nula considera que a utilização da *GiveMe Infra* não é capaz de apoiar a tomada de decisões no contexto da implementação realizada pela equipe de TI da Empresa Parceira 2. Já a Hipótese alternativa considera que a utilização da *GiveMe Infra* é capaz de apoiar a tomada de decisões no contexto da implementação realizada pela equipe de TI da Empresa Parceira 2. Neste sentido, não há indícios que permitam rejeitar a hipótese alternativa, dado que as análises realizadas visaram caracterizar o apoio dado pela *GiveMe Infra* e as limitações desse apoio.

4.3 AMEAÇAS A VALIDADE

As ameaças a validade deste estudo experimental são relacionadas a qualidade dos dados históricos usados nas análises realizadas, considerando a corretude e a completude das informações provenientes dos repositórios de dados e provenientes da manutenção realizada para a implementação do Bloco K.

O nível de corretude indica até que ponto uma manutenção afetou somente módulos e componentes que necessariamente deveriam ser afetados. Caso outros tenham sido alterados como, por exemplo, em parte de outra manutenção, a corretude da informação de rastreabilidade pode ser comprometida. Já a completude dos dados, seguindo o mesmo exemplo, está relacionada ao quão completa é a informação de rastreabilidade gerada em uma manutenção efetuada. Caso módulos e componentes

tenham sido desconsiderados erradamente na manutenção, as informações de rastreabilidade geradas podem estar incompletas.

Para critério de comparação, foram analisados os impactos nos métodos na implementação do Bloco K e os impactos nos métodos indicados pela *GiveMe Infra.* Neste sentido, caso a equipe de TI que implementou o Bloco K não tenha alterado todos os métodos de fato necessários (completude) e somente os de fato necessários (corretude), a comparação com as indicações poderá mostrar mais impactos extras (isto é, que não resultaram em impactos reais na implementação do Bloco K) do que o normal. Isso na prática quer dizer que o nível de corretude e completude das informações analisadas não seria considerado satisfatório.

4.4 CONSIDERAÇÕES FINAIS SOBRE O CAPÍTULO

Este capítulo apresentou um estudo experimental que objetivou verificar o apoio dado pela *GiveMe Infra* ao verificar as alterações realizadas na resolução de uma atividade de manutenção realizada em um contexto real: o projeto *SpedFiscal* da Empresa Parceira 2. Para isso, foram analisadas atividades de manutenção realizadas pela equipe de TI da Empresa Parceira 2 com o intuito de descobrir quais impactos foram ocasionados no projeto de código ao implementar o Bloco K. Posteriormente os impactos foram verificados na *GiveMe Infra* com o objetivo de analisar se era capaz de indicar os mesmos pontos alterados no cenário real. *GiveMe Infra* apoio em 30% das verificações, para o caso de implementação. Outra informação que foi gerada no estudo experimental diz respeito à quantidade de indicações estatísticas que a *GiveMe Infra* foi capaz de processar. Nas verificações realizadas, foram fornecidas indicações que vão além do conjunto das modificações de fato conduzidas pela equipe de TI da Empresa Parceira.

É sabido que a qualidade das indicações calculadas pelo *Statistical Analysis Engine*, o SAE, depende da qualidade dos dados históricos que se tem para um método (MEYER, 1983). Quanto mais manutenções um software sofrer ao longo do tempo, maiores podem ser as chances de se ter um conjunto de dados históricos considerado bom para a realização do cálculo estatístico com o motor SAE. A verificação realizada com a *GiveMe Infra* visou mostrar até que ponto a qualidade dos dados está influenciando na qualidade das indicações fornecidas pela *GiveMe Infra.*

5. CONCLUSÕES

Este trabalho apresentou *GiveMe Infra*, uma infraestrutura baseada em múltiplas visões interativas para apoiar atividades de manutenção e evolução distribuídas. Seu desenvolvimento foi baseado em um problema inicialmente encontrado na Empresa Parceira 1, o qual se relacionava à falta de uma solução (técnica, metodologia, ferramenta, *framework,* ou modelo) que apoiasse as atividades de manutenção e evolução de software, no contexto de equipes geograficamente distribuídas ou co-localizadas. As atividades de manutenção eram realizadas sob a supervisão de uma gerência, que tomava decisões com base nas tarefas realizadas pelas equipes, porém sem um suporte adequado para as atividades de manutenção. Foi verificado que esse mesmo problema acontecia com a Empresa Parceira 2, que também através de parceria, disponibilizou projetos de código e repositórios de dados históricos, como repositórios de código fonte e solicitação de mudanças. As parcerias para fornecimento de dados históricos se estenderam à Empresa Parceira 1 e o SINAPAD.

As hipóteses definidas no início deste trabalho referem-se à capacidade da *GiveMe Infra*, integrada a um Ambiente Interativo baseado em Múltiplas Visões, em auxiliar equipes distribuídas, ou colocalizadas, na tarefa de manter e acompanhar a evolução de projetos de software. O estudo experimental realizado mostrou como se deu auxílio a atividades de manutenção e evolução em um contexto real de manutenção. Nesta avaliação o foco estava no suporte oferecido pelas atividades de compreensão de software ao processo de manutenção e evolução de software. Já a implantação da infraestrutura na Empresa Parceira 1 mostrou indícios de que ela é capaz de auxiliar equipes distribuídas ou colocalizadas em atividades de manutenção e evolução de software. No entanto, novos estudos experimentais devem ser conduzidos objetivando caracterizar o apoio dado ao contexto distribuído.

Através da solução apresentada foi possível visualizar os diferentes tipos de funcionalidades disponíveis para equipes de manutenção e evolução de software. As funcionalidades voltadas para a colaboração permitem que equipes geograficamente distribuídas possam interagir de modo que toda a informação gerada através da interação seja armazenada em um ponto único (um serviço web de envio e recebimento de informações de colaboração).

As contribuições deste trabalho estão na integração com diferentes ferramentas que apoiam a compreensão de software em dois diferentes contextos: (i) o Contexto Atual, que se refere à análise do projeto de código de um software disponível no espaço de trabalho do ambiente de desenvolvimento *Eclipse,* e (ii) Contexto Histórico, que se refere a análise de informações obtidas ao longo do ciclo de manutenção de um software. Sendo assim, equipes de manutenção e evolução de software tem à disposição um conjunto de visualizações integradas à infraestrutura e ao ambiente *Eclipse*. Essas visualizações são capazes de proporcionar diferentes perspectivas sobre um mesmo conjunto de dados. Caso a equipe necessite compreender alguma característica estrutural do software em manutenção, por exemplo, basta iniciar os recursos de análise de código fonte que diferentes visualizações serão habilitadas. Isso é possível graças à integração entre Ambientes Interativos baseados em Múltiplas visões, como o AIMV e o *Sourceminer*. Em outro contexto, caso a equipe necessite analisar como se dá a evolução do software ao longo dos ciclos de manutenção, basta utilizar os recursos de análise de dados históricos, os quais habilitarão visualizações, tais como: *Changed View, Deep View*, *Comparison View*, entre outras.

Além da contribuição dada com a integração entre ferramentas e visualizações, que apoiam a compreensão de software, cabe citar as contribuições dadas à área de colaboração na manutenção e evolução de software. *GiveMe Infra* integra recursos que apoiam a colaboração entre membros de uma equipe distribuída ao permitir que mensagens de colaboração sejam inseridas diretamente nas visualizações integradas à infraestrutura. Neste sentido, os recursos disponíveis na infraestrutura são passíveis de serem utilizados em um ambiente distribuído.

Outra contribuição deste trabalho está na possibilidade de integração, na *GiveMe Infra,* de outras ferramentas e visualizações, que atuem de forma distribuída ou colocalizada, em atividades de manutenção e evolução de software. Como resultado, outros projetos integrados à infraestrutura, podem se beneficiar desses recursos ao mesmo tempo em que apoiam as atividades de compreensão em relação ao software em manutenção.

Além disso, é possível desenvolver *drivers* de leitura de dados históricos para bases de dados de outras empresas, além daquelas que foram utilizadas neste projeto. Com isso, é possível oferecer mais opções de diferentes tipos de repositórios de dados históricos a serem analisados.

As dificuldades encontradas neste trabalho são relacionadas às integrações

realizadas, principalmente entre ferramentas de terceiros, como *Collaborative Sourceminer*, *Sourceminer*, AIMV e Subclipse. Inicialmente, foram necessárias adaptações que viabilizassem a comunicação entre todas as ferramentas da infraestrutura, em uma arquitetura definida. Como a proposta deste trabalho é permitir que atividades de manutenção e evolução de software sejam realizadas em dois contextos (Contexto Atual e Contexto Histórico), foi necessário desenvolver mecanismos que permitissem a troca de contexto em tempo de execução, para as ferramentas e recursos pertencentes à infraestrutura. Outra dificuldade foi encontrar dados históricos e repositórios de dados históricos com certo nível de corretude e completude, como mencionado nas ameaças a validade do estudo experimental. A corretude e a completude das informações faz com que a ferramenta que realiza as análises estatísticas melhorem cada vez mais a precisão das indicações de possíveis pontos a serem alterados em uma manutenção de software.

Como trabalhos futuros espera-se realizar parcerias com outras empresas de desenvolvimento para que diferentes tipos repositórios possam ser analisados. Como resultado, espera-se aumentar o conjunto de métricas calculadas disponíveis na *GiveMe Infra*. Além disso, espera-se desenvolver novos *drivers* de leitura de dados históricos para supportar todos os repositórios passíveis de serem manipulados com o *GiveMe Metrics*.

Espera-se, a partir de um conjunto maior de métricas, conduzir outros estudos experimentais com objetivo, por exemplo, de verificar a viabilidade de uso da *GiveMe Infra* em atividades de manutenção realizadas em ambiente distribuído. Sendo assim, será possível verificar o apoio dado pelos recursos de colaboração disponíveis. Mais ainda, espera-se com isso explorar aspectos relacionados ao desenvolvimento distribuído de software como, por exemplo: reputação, confiança, gerência de risco, rastreabilidade na manutenção e evolução, entre outros.

Em outra vertente dentro do mesmo objetivo, que é apoiar a evolução colaborativa de software, espera-se desenvolver um mecanismo baseado em ontologias para que análises semânticas possam ser realizadas nos tipos de manutenção atualmente suportados pela *GiveMe Infra*. No cenário atual, está prevista uma etapa através da qual o usuário da infraestrutura é o responsável por associar os diferentes termos encontrados no conjunto de dados históricos de solicitações de mudanças com os tipos de manutenção suportados (manutenção corretiva, adaptativa e evolutiva), para efetuar o

cálculo de algumas métricas. Com a implementação desse mecanismo, espera-se que essa associação seja feita de forma automática ou semiautomática.

O desenvolvimento de uma nova ferramenta que apoie o planejamento e a gestão das atividades de manutenção e evolução (atualmente suportadas pela *GiveMe Infra*) é outro trabalho futuro. Atualmente, *GiveMe Infra* apenas permite o cadastramento e a realização de atividades de manutenção, mas não apoia o planejamento de tempo e custo de tais atividades. Futuramente espera-se integrar à infraestrutura recursos que apoiem o planejamento como parte do processo de execução de atividades de manutenção e evolução de software.

Agradecimentos

A CAPES - Coordenação de Aperfeiçoamento de Pessoal de Nível Superior pelo financiamento do projeto, as empresas parceiras e ao SINAPAD pela disponibilização de repositórios de dados históricos, ao Departamento de Estatística da UFJF, aos grupos de pesquisa em compreensão e colaboração que foram parceiros neste trabalho e aos colaboradores Emmanuel Coimbra, Marcos Alexandre Miguel e Cláudio Lélis.

REFERÊNCIAS

ANSLOW, C.; MARSHALL, S.; NOBLE, J.; BIDDLE, R., **SourceVis: Collaborative software visualization for co-located environments**. *Software Visualization (VISSOFT), 2013 First IEEE Working Conference on* , vol., no., pp.1,10, 27-28 Sept. 2013.

ANSLOW, CRAIG. **Co-located collaborative software visualization**. *Human Aspects of Software Engineering*. ACM, 2010.

BASILI, VICTOR R. CALDIERA, GIANLUIGI H. ROMBACH, DIETER. **The Goal Question Metric Approach**. Chapter in Encyclopedia of Software Engineering, Wiley, 1994.

BLY, S. A. **A use of drawing surfaces in different collaborative settings**. in Proceedings of the ACM conference on Computer-supported cooperative work (CSCW'88). New York, NY, USA: ACM, 1988, pp. 250–256, 1988.

BRAGA, R. M. M., COSTA, M. N., WERNER C. M. L., MATTOSO, M. L. Q., 2000, A Multi-Agent System for Domain Information Discovery and Filtering, In: XIV Simpósio Brasileiro de Engenharia de Software (SBES), João Pessoa, Oct. 2000.

BUGZILLA. Disponível em: http://www.bugzilla.org/features/. Acesso 25 de novembro de 2013.

CARNEIRO, G. F. **Sourceminer: um ambiente integrado para Visualização multi-perspectiva de software**. 2011. 230 f. Tese (doutorado) – Universidade Federal da Bahia, Instituto de Matemática, Doutorado Multi-institucional em Ciência da Computação. 2011.

CONCEIÇÃO, C.F.R. **Analisando o Uso de Elementos de Percepção Para Atividades de Compreensão de Software em um Ambiente de Desenvolvimento Distribuído**. Dissertação de Mestrado, UNIFACS, Salvador. 2012

D'AMBROS, M.; LANZA, M.A **Flexible Framework to Support Collaborative Software Evolution Analysis**. *Software Maintenance and Reengineering, 2008. CSMR 2008. 12th European Conference on* , vol., no., pp.3,12, 1-4 April 2008

DAMBROS, MARCO. LANZA, MICHELE. **Distributed and Collaborative Software Evolution Analysis with Churrasco**. *Sci. Comput. Program.* 75, 4. April, 276-287. 2010.

FUKS, H., RAPOSO, A.B. E GEROSA, M.A., 2003. **Do Modelo de Colaboração 3C à Engenharia de Groupware**. Simpósio Brasileiro de Sistemas Multimídia e Web – Webmidia 2003, Trilha especial de Trabalho Cooperativo Assistido por Computador. Salvador-BA. 03 a 06 de Novembro, 2003.

GIT. Disponível em: http://git-scm.com/. Acesso em 20 de novembro de 2014.

GIVEME INFRA. Disponível em: https://www.givemeinfra.com.br/documents. Acesso 24 dez. 2014.

GIVEME METRICS. Disponível em: https://www.givemeinfra.com.br/givememetrics. Acesso: 24 dez. 2014.

GIVEME TRACE. Disponível em: https://www.givemeinfra.com.br/givemetrace. Acesso: 24 dez. 2014.

GIVEME VIEWS. Disponível em: https://www.givemeinfra.com.br/givemeviews. Acesso 24 dez. 2014.

HOFFMANN, R. Considerações Sobre a Evolução Recente da Distribuição da Renda no Brasil. *RAE-Revista de Administração de Empresas*, v. 13, n. 4, out-dez, 1973.

IEEE93 IEEE STD. 1219: **Standard for Software Maintenance**. Los Alamitos CA., USA. IEEE Computer Society Press, 1993.

ISENBERG, P., ELMQVIST, N. SCHOLTZ, J. CERNEA, D., MA, K., HAGEN, H. **Collaborative visualization: definition, challenges, and research agenda**. *Information Visualization* 10, 4, 310-326, 2011.

JOHNSON, R. E., 1997, "Frameworks = (Components + Patterns)". Communications of the ACM, v. 40, n. 10, pp. 39-42, Oct. 1997.

KEVIC, K.; MULLER, S.C.; FRITZ, T.; GALL, H.C. C**ollaborative bug triaging using textual similarities and change set analysis**. *Cooperative and Human Aspects of Software Engineering (CHASE), 2013 6th International Workshop on* , vol., no., pp.17,24, 25-25 May. 2013.

KITCHENHAM, B.A., CHARTERS, S. **Guidelines for performing systematic literature reviews in software engineering**. Tech. Rep. EBSE-2007-01. KeeleUniversity. 2007.

LEHMAN, M. M. **Laws of Software Evolution Revisited**. In *Proceedings of the 5th European Workshop on Software Process Technology* (EWSPT '96), Carlo Montangero (Ed.). Springer-Verlag, London, UK, UK, 108-124, 1996.

LÉLIS, C. A. S, ARAÚJO, M. P, DAVID, J. M. N. ***GiveMe Trace*: Uma ferramenta para apoio a rastreabilidade de software**. 2014. Trabalho de Conclusão de Curso (Graduação em Sistemas de Informação) - Universidade Federal de Juiz de Fora, 2014.

LEVIN, J. FOX, J, FORDE, A. DAVID, R. Estatística para Ciências Humanas. (2012). 11ª ed. São Paulo: Pearson Education do Brasil.

LORGE, D. PARNAS. **Software Aging**. pp 279-287, ICSE 1994.

LUNGU, MIRCEA. MICHELE LANZA. OSCAR NIERSTRASZ. **Evolutionary and collaborative software architecture recovery with Softwarenaut**. *Sci. Comput. Program.* 79, 204-223. 2014.

MANTIS: Disponivel em http://www.mantisbt.org/. Acesso 25 de novembro de 2013.

MAO, CHENGYING. **Structure visualization and analysis for software dependence network**. *Granular Computing (GrC), 2011 IEEE International Conference on* , vol., no., pp.439,444, 8-10 Nov. 2011.

MEYER, PAUL L. **Probabilidade – Aplicações à Estatística**. 2ª ed. LTC – Livros Técnicos e Científicos Editora LTDA. 1983.

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.

MYLYN-MANTIS. Disponível em http://marketplace.eclipse.org/content/mylyn-mantis-connector. Acesso em 26 de novembro de 2014.

PRESSMAM, R. S. **Engenharia de Software**. 6a. ed. McGraw-Hill, 2006

REDMINE. Disponível em: http://www.redmine.org/. Acesso 25 de novembro de 2013.

SANTANA, F. OLIVA, GUSTAVO. SOUSA, GLEIDSON R.B, GEROSA, MARCO AURÉLIO. **Xflow: An extensible tool for empirical analisys of software systems evolution**. In Proceeding of the VIII Experimental Software Engeneering Latin Americam Workshop, ESELAW'11, 2011.

SHRESTHA, A.; YING ZHU; MILLER, B. V**isualizing time and geography of open source software with storygraph**. *Software Visualization (VISSOFT), 2013 First IEEE Working Conference on* , vol., no., pp.1,4, 27-28 Sept. 2013.

SILVA, A. N., **Um tollkit de referência para o desenvolvimento de ambientes interativos baseados em múltiplas visões**. Dissertação de Mestrado, UNIFACS, Salvador. 2013.

SILVA, A. N. CARNEIRO, G. F. ZANIN, R. B. DAL POZ, A. P. MARTINS, E. F. O. **Propondo uma Arquitetura para Ambientes Interativos baseados em Multiplas Visões**. II Workshop Brasileiro de Visualização de Software, Natal, RN (*1 – 8)*, 2012.

SINAPAD: https://www.lncc.br/sinapad/. Acesso em 25 novembro de 2013.

SNEED, H.M., **Software evolution. A road map**. *Software Maintenance, 2001. Proceedings. IEEE International Conference on* , vol., no., pp.7, 2001

SOMMERVILLE, IAN. **Engenharia de software**, 8ª Ed. São Paulo: Pearson Addison-Wesley, 2007.

SPED MANUAL: http://www1.receita.fazenda.gov.br/sistemas/sped-fiscal/download/minuta_guia_pratico_efd_icms_ipi.pdf. Acesso em 03 janeiro de 2015.

SPED: http://www1.receita.fazenda.gov.br/sistemas/sped-fiscal/. Acesso em 03 de janeiro de 2015.

STOREY, M.; BENNETT, C.; BULL, R.I.; GERMAN, D.M. **Remixing visualization to support collaboration in software maintenance**. *Frontiers of Software Maintenance, 2008. FoSM 2008.* , vol., no., pp.139,148, Sept. 28 2008-Oct. 4, 2008.

SUBCLIPSE. Disponível em: http://subclipse.tigris.org/. Acesso 28 de nov. 2014.

SVN BOOK. Disponível em: http://svnbook.red-bean.com/en/1.7/svn.ref.svn.c.diff.html. Acesso 12 de jan. 2015.

SVN. Disponível em: https://subversion.apache.org/. Acesso em 22 de novembro de 2014.

SYEED, M. M. MAHBUBUL, IMED HAMMOUDA, CSABA BERKO. **Exploring Socio-Technical Dependencies in Open Source Software Projects: Towards an Automated Data-driven Approach**. In *Proceedings of International Conference on Making Sense of Converging Media* (AcademicMindTrek '13), Artur Lugmayr, Heljä Franssila, Janne Paavilainen, and Hannu Kärkkäinen (Eds.). ACM, New York, NY, USA, , Pages 273 , 8 pages. 2013.

TAVARES, J., DAVID, J. M. N., ARAÚJO, M. A.P., BRAGA, R., CAMPOS, F. C. A. **GiveMe Metrics – Um framework conceitual para extração de dados históricos sobre a evolução do software**. X Simpósio Brasileiro de Sistemas de Informação (SBSI), Londrina/PR. 2014.

TAVARES, J., DAVID, J. M. N., ARAÚJO, M. A.P., BRAGA, R., CAMPOS, F. C. CARNEIRO, G. F. **GiveMe Views: uma ferramenta de suporte à evolução de software baseada na análise de dados históricos**. Artigo submetido ao XI Simpósio Brasileiro de Sistemas de Informação (SBSI) 2015.

TELEA, A.; VOINEA, L. **Interactive Visual Mechanisms for Exploring Source Code Evolution**. *Visualizing Software for Understanding and Analysis. VISSOFT 2005. 3rd IEEE International Workshop on* , vol., no., pp.1,6, 0-0 0. 2005

TELEA, ALEXANDRU. AUBER, DAVID. **Code flows: visualizing structural evolution of source code**. In *Proceedings of the 10th Joint Eurographics / IEEE - VGTC conference on Visualization* (EuroVis'08). Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 831-838. 2008.

TORTOISE: http://tortoisesvn.net/. Acesso em 10 de novembro de 2014.

VOIGT, S.; BOHNET, J.; DOLLNER, J. **Object aware execution trace exploration**. *Software Maintenance, 2009. ICSM 2009. IEEE International Conference on* , vol., no., pp.201,210, 20-26 Sept. 2009.

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.