Uma Infraestrutura baseada em Múltiplas Visões Interativas para Apoiar Evolução de Software

Jacimar F. Tavares[1], José Maria N. David[1], Marco Antônio P. Araújo[1,2], Regina Braga[1], Fernanda Campos[1], Glauco de F. Carneiro[3]

[1]Programa de Pós Graduação em Ciência da Computação
Universidade Federal de Juiz de Fora (UFJF) – Juiz de Fora / MG

[2]Instituto Federal de Educação, Ciência e Tecnologia do Sudeste de Minas Gerais - Campus Juiz de Fora (IF Sudeste MG)

[3]Universidade Salvador (UNIFACS) – Salvador / BA

jacimar.tavares@ice.ufjf.br, {jose.david, marco.araujo, regina.braga, fernanda.campos}@ufjf.edu.br, glauco.carneiro@unifacs.br

Resumo. [Contexto] *Tecnologias e ambientes de desenvolvimento voltados para as áreas de manutenção e evolução colaborativas de software têm atraído a atenção de empresas e grupos de usuários no sentido de resolver seus problemas em projetos de software. Entretanto, em muitos casos é necessário utilizar diferentes ferramentas. Como resultado, o tempo e o esforço dispendidos são crescentes em função da diversidade de soluções. Esses problemas se tornam mais intensos quando se trata de equipes geograficamente distribuídas.* ***[Objetivo]*** *Neste sentido, foi desenvolvida a GiveMe Infra, uma infraestrutura para apoio à realização de atividades de manutenção e evolução de software por equipes co-localizadas ou geograficamente distribuídas. Além disso, essa infraestrutura considera os resultados obtidos das atividades de compreensão de software. Essas atividades são apoiadas por visualizações de software que permitem ao usuário obter diferentes perspectivas sobre as informações disponibilizadas.* ***[Método]*** *A infraestrutura apoia a identificação de métodos que podem ser afetados quando um outro método é alterado, auxiliando equipes de manutenção na modificação de sistemas. Para isso, foi realizado um estudo experimental num contexto real de manutenção em empresas parceiras, avaliando o apoio dado pela infraestrutura, através dos recursos oferecidos.* ***[Resultados]*** *Através da utilização dessa infraestrutura foi possível obter apoio na identificação de métodos a serem alterados quando uma determinada modificação é feita. Essa infraestrutura permitiu obter maiores informações sobre o processo de evolução de software, objetivando a realização de atividades de manutenção de maior qualidade.* ***[Conclusões]*** *Os resultados obtidos evidenciaram a importância da GiveMe Infra para apoiar as atividades de manutenção e evolução. Entretanto, avaliações adicionais são necessárias.*

Palavras-chave: compreensão de software, manutenção de software, visualização de software, evolução de software.

TAVARES, J. F.; DAVID, J. M. N.; ARAUJO, M. A. P.; BRAGA, R.; CAMPOS, F.; CARNEIRO, G. F.
Uma Infraestrutura baseada em Múltiplas Visões Interativas para Apoiar Evolução de Software
iSys - Revista Brasileira de Sistemas de Informação, Rio de Janeiro, vol. 8, No. 1, p. 65-101, 2015.

1. Introdução

A atividade de desenvolvimento de software frequentemente gera um conjunto de artefatos que necessitam ser gerenciados. Esses artefatos são muitas vezes armazenados em repositórios de dados e mantidos ao longo do ciclo de vida do software. Como resultado, podem fornecer uma gama de informações sobre sua evolução mas, para que isso seja possível, é necessário extrair dados e analisá-los. Em alguns casos, dados provenientes de diferentes repositórios devem ser combinados para se obter uma determinada informação [Poncin et al., 2011] [Ligu et al., 2013]. Desses repositórios, pode-se ainda extrair dados de um determinado período de tempo, o que pode ser uma tarefa complexa, dependendo de como os dados estejam armazenados em diferentes tipos de repositórios [Pedroso et al., 2013]. Os dados extraídos podem auxiliar as organizações de diferentes formas como, por exemplo, através de uma análise que objetiva entender porque os níveis de acoplamento modificam significativamente de uma versão do software para outra. Para tanto, muitas vezes é necessário que se utilize ferramentas específicas para cada tipo de repositório a ser manipulado, objetivando a extração de seus dados históricos.

Nesse contexto, inicialmente foi proposta uma solução que integra diferentes ferramentas existentes, capazes de extrair dados históricos sobre evolução de software. Tal solução se constitui em um *framework* conceitual[1], que através da estruturação de conceitos, denota um modelo que visa guiar atividades de extração de dados históricos. Seu objetivo é descrever, conceitualmente, uma sequência de passos rumo à extração de dados históricos e guiar os usuários na tarefa de identificar qual o caminho mais indicado, dado seu objetivo. Essa solução, denominada de *GiveMe Metrics* [Tavares et al., 2014], trata-se de um *framework* conceitual cujo objetivo é guiar usuários na tarefa de extrair dados históricos.

Adicionalmente, identificou-se a necessidade de melhorar o processo de manutenção, intensificando o controle sobre a evolução de software. Neste sentido, buscou-se tratar os aspectos de colaboração entre equipes geograficamente distribuídas que necessitam de recursos visuais para facilitar as atividades de manutenção. Para tanto, foi desenvolvida a ferramenta *GiveMe Views* [Tavares et al., 2015]. Essa ferramenta apoia as atividades de manutenção através de elementos de visualização colaborativa de software.

Posteriormente, essa solução evoluiu para uma infraestrutura[2], denominada *GiveMe Infra*, que combina ferramentas de terceiros com ferramentas desenvolvidas para obter indicações de módulos e componentes a serem alterados. *GiveMe Infra* é uma infraestrutura baseada em múltiplas visões interativas para apoio de 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. AIMVs fornecem meios para que análises sobre dados sejam feitas a partir da utilização de visualizações, e que tais ambientes forneçam meios para que haja a

1 No contexto desta pesquisa, um *framework* conceitual é um conjunto de conceitos relacionados, que é construído com o objetivo de resolver um problema de um domínio específico, no caso deste trabalho, a evolução de software.

2 Infraestrutura se caracteriza como um conjunto de modelos e arquiteturas que formam uma base para a especificação e implementação de aplicações em num domínio específico, através da reutilização desse conjunto. No contexto desta pesquisa, a infraestrutura utilizada permite implementações de aplicações para apoiar a evolução de software e a integração dessas aplicações.

coordenação entre as visualizações. Além disso, combinar diferentes visualizações pode permitir a análise de dados em várias perspectivas [Silva et al., 2012].

As ferramentas, que compõem a solução apresentada, foram obtidas a partir de um processo de mapeamento sistemático[3], no qual foram definidos critérios para que uma ferramenta pudesse ser selecionada para compor o *framework*. *GiveMe Metrics* apoia a extração de dados em três diferentes tipos de repositórios de dados históricos, tais como: (i) de código fonte, (ii) de defeitos e (iii) de processos de desenvolvimento de software. Com o objetivo de obter indícios sobre a viabilidade de uso do *framework*, uma avaliação foi realizada [Tavares et al., 2014], a partir da qual foram extraídos dados sobre defeitos em projetos do SINAPAD[4] (Sistema Nacional de Processamento de Alto Desempenho), que realiza pesquisas científicas na área da computação. Também, Tavares et al. [2014] apresentam uma análise dos dados extraídos, objetivando investigar algumas questões como, por exemplo, quais os projetos mantidos que são os maiores alvos de manutenções corretivas. Nesse contexto, considerando a avaliação realizada no SINAPAD, o projeto *sinapad-framework* teve a maior porcentagem de defeitos ao longo dos anos e sofreu uma quantidade substancial de manutenções corretivas, por se tratar de um projeto base para outros. Essa dependência entre projetos faz com que defeitos no *sinapad-framework* pudessem interferir no funcionamento de outros softwares. Ações corretivas foram especificadas no sentido de minimizar esses efeitos [Tavares et al., 2014]. Ao final, foi possível obter indícios sobre a viabilidade de utilização do *GiveMe Metrics* na extração de dados sobre defeitos.

Entretanto, o *GiveMe Metrics* é apenas um *framework* conceitual e necessita evoluir. Os objetivos deste artigo são apresentar (i) a evolução do *framework,* em relação à primeira versão descrita em [Tavares et al., 2014], (ii) sua integração com soluções que visam apoiar as atividades de manutenção e evolução, tal como *GiveMe Views*, apresentada em [Tavares et al., 2015], (iii) a solução proposta que se constitui na infraestrutura para apoiar as atividades de manutenção e evolução de software, e (iv) a avaliação das soluções integradas (*GiveMe Metrics e GiveMe Views*), que compõem a infraestrutura *GiveMe Infra*, através de um estudo experimental realizado. Na avaliação, deseja-se verificar se *GiveMe Infra* é, de fato, capaz de apoiar atividades de manutenção e evolução ao se basear na análise estatística gerada com os dados extraídos com *GiveMe Metrics.* Sendo assim, buscou-se verificar até que ponto os dados históricos extraídos podem apoiar a compreensão e a evolução de projetos de software.

A metodologia seguida neste trabalho envolveu as seguintes etapas: (i) evolução do *framework GiveMe Metrics*. Nessa etapa foi necessário criar uma nova versão (em relação a primeira versão apresentada em [Tavares et al., 2014]) dado que não era possível extrair dados históricos de repositórios customizados; (ii) integração com ferramenta de apoio à compreensão e evolução de software. Etapa que permitiu que os dados históricos extraídos com o *framework* pudessem ser usados para apoiar atividades de compreensão e evolução de projetos de software na indústria; (iii) a especificação, projeto e implementação da infraestrutura proposta; e (iv) estudo experimental, que visou verificar a forma pela qual a solução, integrada ao *GiveMe*

3 Disponível na íntegra em: http://www.givemeinfra.com.br/files/givememetricsmapping.pdf.

4 https://www.lncc.br/sinapad/.

Metrics, apoia atividades de compreensão e evolução de software, ao fazer uso dos dados históricos extraídos.

Este artigo está organizado da seguinte forma: a seção 2 apresenta trabalhos relacionados, enquanto a seção 3 apresenta o *framework GiveMe Metrics*. A seção 4 apresenta a integração do *GiveMe Metrics* com a ferramenta *GiveMe Views*, através da infraestrutura *GiveMe Infra*. A seção 5 ilustra a importância da integração de um Ambiente Interativo de Múltiplas Visões (AIMV) voltado para a compreensão de software com ferramentas que apoiam a evolução de software. A seção 6 apresenta um estudo experimental realizado com a infraestrutura *GiveMe Infra*. Por fim, a seção 7 apresenta as considerações finais e os trabalhos futuros.

2. Trabalhos Relacionados

Alguns trabalhos na literatura técnica de manutenção e evolução de software abordam o mesmo problema deste trabalho, que diz respeito à 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. Essa solução se baseia em informações provenientes de dados históricos, bem como utilizam elementos visuais que apoiam as atividades de compreensão de software. Esta seção apresenta os trabalhos relacionados que foram obtidos através da condução de uma Revisão Sistemática de Literatura [Kitchenham et al., 2007] que está disponível em [GiveMe Infra, 2014].

Syeed et al. (2013) mencionam que dados históricos podem indicar o processo de evolução no qual um dado software foi submetido, e sobre o conhecimento tácito que foi utilizado durante a evolução. Além disso, podem fornecer um histórico sobre a comunicação e a colaboração realizadas pelos desenvolvedores. Nesse contexto, os autores definem um *framework* para automatizar a análise e a visualização da evolução do software, através de dados históricos de projetos *open source*. Estão interessados em entender como se dá a evolução de software observando, por exemplo, mudanças no número de linhas de código fonte, número de *commits* realizados, comentários inseridos, complexidade ciclomática, dentre outros. Adicionalmente, têm como objetivo analisar, por exemplo, como se deu a comunicação entre membros da equipe de desenvolvimento nas atividades de evolução de software. Entretanto, o *framework* apresentado não é capaz de apoiar, por exemplo, atividades de manutenção colaborativa, fornecendo informações que apoiem a tomada de decisões quando manutenções no nível de código fonte são necessárias.

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 análise de dados históricos objetivando apoiar a tomada de decisões sobre manutenções mediante as indicações de pontos a serem alterados. A solução tem como foco mostrar um panorama sobre a evolução do software. Isso, de fato, pode contribuir para a tomada de decisões rumo a novas manutenções, mas não de forma direta como é proposto pelas

ferramentas que compõem a *GiveMe Infra*, a qual 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 oferecer visualizações colaborativas para apoiar o trabalho de equipes distribuídas nas tarefas de compreensão sobre a evolução do software. Essas tarefas podem ser de manutenção, mas não é o foco exclusivo e nem principal de SourceVis [Anslow, 2010]. Entre as informações que podem ser visualizadas estão métricas de código fonte como número de acoplamentos, linhas de código e 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 fonte, que fornece métricas e visualizações sobre a relação entre entidades de código como classes e métodos, também apresentará uma visão sobre dados históricos.

Em [Dambros e Lanza, 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 [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 fonte 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 na linguagem 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 a capacidade de obter automaticamente dados históricos de repositórios de código fonte como SVN e a capacidade de apoiar equipes distribuídas. Apoia a colaboração entre membros das equipes, 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* [2013] e *Mantis Bug Tracker* [2013] ou repositórios customizados, desenvolvidos por diferentes empresas; e (ii) não oferece indicações diretas e estatísticas sobre pontos que podem ser afetados, dada uma solicitação de mudança.

Em [Telea e Voinea, 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), através do qual são definidos meios com os quais é possível navegar, de forma visual, entre artefatos de código. Esse conceito é utilizado na técnica apresentada em forma de visões. Dentre elas estão *Syntactic View* e *Evolution View*. *Syntactic View* permite que o usuário visualize, em miniatura, um conjunto selecionado de arquivos de código fonte disponíveis em um projeto. Por exemplo, pode-se selecionar 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 fonte, ao longo das versões do software.

A principal diferença entre a técnica apresentada em [Telea e Voinea, 2005] e a solução proposta para este trabalho está na forma como a evolução de um software pode ser acompanhada. A solução proposta para este trabalho deve processar as informações e disponibilizá-las em visualizações específicas. Já essa técnica 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 apoiar a análise de arquivos de código fonte provenientes de repositórios. Tem como objetivo fornecer ao usuário do *framework* de 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. Nesse trabalho é apresentado um mecanismo que permite que os arquivos de código fonte 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 exibe 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 fonte com suas classes. Um pacote seria representado por um 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. Essa métrica pode oferecer algum indício de pontos que podem ser afetados quando uma 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 e Auber, 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 seu ciclo de vida. Com base nisso, 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 em um projeto de código fonte 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, isso quer dizer que *Code Flows* é capaz de indicar que um bloco de código que estava em uma classe em uma versão de um software, agora, em uma verão seguinte encontra-se em outra classe ou em outro ponto da classe original. 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 que pertencia a uma classe (representada como sendo um nó pai na árvore e o método como sendo um nó filho) em uma outra versão do software analisado passou a pertencer a uma outra classe (portanto agora o método é nó filho da nova classe). Assim, como as outras soluções previamente apresentadas, *Code Flows* atua fornecendo meios para que se possa analisar a evolução de software focando exclusivamente para o contexto do código fonte. Não permite 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, a descoberta da rastreabilidade entre entidades de código fonte orientado a objetos, como métodos de uma classe. Nesse sentido, busca-se apoiar a análise do grau de dependência entre as entidades objetivando aprender sobre os acoplamentos existentes. A técnica desenvolvida é acompanhada de visualizações que permitem explorar graficamente a rastreabilidade descoberta. Entretanto, a técnica não contempla atividades de evolução em equipes geograficamente distribuídas, o que é importante na resolução do problema apresentado neste trabalho. Além disso, não integra atividades de compreensão àquelas relacionadas com a evolução de software.

Kevic et al. [2013] apresentam uma técnica que permite a análise de defeitos cadastrados em um sistema de *bug tracking* objetivando encontrar similaridades entre eles. 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, irão analisar outros defeitos que já tenham sido resolvidos anteriormente e selecionar aqueles que julgarem semelhantes ao defeito a ser resolvido. Em seguida, são apresentadas as mudanças ocorridas, no nível de código fonte, indicando que o defeito a ser resolvido poderá afetar aqueles mesmos trechos de código fonte, dada a similaridade entre eles.

A técnica apresentada em [Kevic et al., 2013] atua considerando dados históricos de defeitos e dados históricos de alterações no nível de código fonte. Possui ainda a capacidade de permitir que equipes trabalhem de forma colaborativa em atividades de manutenção e evolução de software. Entretanto, a técnica apresentada necessita de um conjunto de desenvolvedores atuando de forma colaborativa e utilizando conhecimento tácito para descobrir quais defeitos são semelhantes àquele a ser resolvido. Essa tarefa é feita analisando apenas os dados cadastrais dos defeitos, como tipo, data de abertura e, principalmente, descrição textual do defeito encontrado.

A solução proposta nesse trabalho irá calcular as indicações automaticamente e necessitará de apenas um desenvolvedor para dar início na execução do processo. Além disso, apresenta os possíveis pontos no código fonte que podem ser afetados apenas

quando a atividade de manutenção a ser realizada corresponde a uma solicitação de mudança do tipo corretiva (para correção de defeitos). Não contempla outros tipos de solicitações de mudança como corretivas, adaptativas e evolutivas. Mais ainda, 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 afetados.

A partir da análise das ferramentas existentes na literatura técnica, e do conjunto inicial de requisitos definidos para a solução proposta, a Tabela 1 apresenta uma síntese das características previamente identificadas. Para esse conjunto, foram definidos os seguintes requisitos para a infraestrutura proposta como solução:

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 afetados quando um dado método for alterado.

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

Tecnologias	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, 2010] e [Anslow et al., 2013]	PARCIAL	PARCIAL	PARCIAL	PARCIAL	X
[Dambros e Lanza, 2010]	X	TOTAL	PARCIAL	PARCIAL	X
[Telea e Voinea, 2005]	X	X	PARCIAL	PARCIAL	X
[Mao, 2011]	X	X	PARCIAL	PARCIAL	X
[Telea e Auber, 2008]	X	X	PARCIAL	PARCIAL	X
[Voigt et al., 2009]	PARCIAL	X	PARCIAL	PARCIAL	PARCIAL
[Kevic et al., 2013]	PARCIAL	PARCIAL	PARCIAL	PARCIAL	PARCIAL

Legenda: TOTAL - apoio completo; PARCIAL - apoio parcial; X - sem apoio.

3. Um *Framework* Conceitual para Extração de Dados Históricos sobre Evolução de Software

GiveMe Metrics é um *framework* conceitual cujo objetivo é guiar usuários na tarefa de extrair dados históricos sobre a evolução de software. É constituído por ferramentas que foram obtidas através de mapeamento sistemático de literatura realizado.

3.1 Mapeamento Sistemático

O mapeamento sistemático é um mecanismo que permite ao pesquisador fornecer uma visão geral sobre a área de pesquisa que está sendo investigada, levando em conta fatores como a quantidade de pesquisas publicadas sobre a área pesquisada e os tipos de pesquisa disponíveis [Steinmacher et al., 2012]. Já um protocolo de mapeamento é o método que o pesquisador define na formalização de buscas sobre a área de pesquisa alvo de investigação [Kitchenham, 2004]. Neste artigo, optou-se pela realização de um mapeamento sistemático para auxiliar na busca por ferramentas, para compor o *framework* GiveMe Metrics, dada a formalidade e o rigor que se pode obter ao realizar uma busca.

O protocolo desenvolvido foi constituído dos seguintes tópicos: objetivo, critérios para avaliação das ferramentas, questões de pesquisa, *string* de busca e bases de dados de publicações.

Objetivo: identificar as ferramentas capazes de reportar dados sobre evolução de software, provenientes de três categorias de repositórios: Categoria 1 - Repositórios de Código Fonte; Categoria 2 - Repositórios de Defeitos de Software; Categoria 3 - Repositórios de Dados sobre o Processo de Desenvolvimento de Software.

Critérios para a avaliação e aceitação das ferramentas: foram definidos critérios para essa tarefa, estabelecidos com base na experiência de pesquisadores da área, tendo sido apresentadas as características dos repositórios (de código fonte, de defeitos e de processos de desenvolvimento de software), contendo características utilizadas por outras instituições parceiras[5], além do SINAPAD. Assim, os critérios foram definidos com base naqueles identificados na literatura e revisados pelo grupo de pesquisadores.

Cada categoria possui um conjunto específico de características, conforme apresentado nas Tabelas 2 a 4.

Tabela 2. Características das Ferramentas da Categoria 1 - Repositórios de Código Fonte

Característica	Descrição
C1	capacidade de se conectar a repositórios de código fonte SVN ou GitHub.
C2	capacidade de analisar código fonte Java ou C#
C3	capacidade de extrair automaticamente um conjunto de métricas por cada *release* ou *commit*, ou por projeto de um software no repositório. Uma ferramenta será considerada válida se extrair pelo menos uma das métricas de cada uma das características de software definidas em [Araújo et al., 2012], que são Tamanho[6], Periodicidade[7] e Complexidade[8]
C4	capacidade de exportar os resultados nos seguintes formatos: .xls, .csv, .txt, .xml, ou gravá-los em um banco de dados

5 O nome das instituições parceiras foi omitido por questões de sigilo.

6 Tamanho: caracterizado pela quantidade de artefatos produzidos em cada etapa do ciclo de vida do software e medido por uma das métricas a seguir: número de pontos de função, números de pontos de caso de uso, número de requisitos, número de classes, número de métodos por classe, número de classes de domínio, número de classes de suporte, número de subsistemas, número de linhas de código fonte.

7 Periodicidade: representa o intervalo de tempo decorrido entre cada versão produzida de um artefato e medida pelo intervalo de tempo entre versões de um produto (diz respeito à versão do software a partir da qual as métricas são extraídas).

8 Complexidade: identificada através de elementos que podem medir a complexidade estrutural de um artefato e medida por uma das métricas a seguir: número de casos de uso, número de diagramas de classes, número de diagramas de sequência, número de diagramas de estados, número de diagramas de empacotamento, número de diagramas de atividades, profundidade de herança por classe, número de filhos por classe, acoplamento entre objetos, resposta de uma classe, falta de coesão entre métodos, complexidade ciclomática por método.

Tabela 3. Características das Ferramentas da Categoria 2 - Repositórios de Defeitos de Software

Característica	Descrição
C1	capacidade de se conectar a repositórios de registro de defeitos, mesmo sob autenticação
C2	capacidade de retornar automaticamente um conjunto de dados históricos por *release, commit* ou por projeto de um software no repositório. Devido à diversidade de tipos de dados que podem ser extraídos sobre um defeito, não será limitado o tipo de defeito registrado. Entretanto, ferramentas que extraiam defeitos de código fonte devem considerar apenas softwares desenvolvidos nas linguagens Java ou C#
C3	capacidade de exportar os resultados obtidos nos seguintes formatos: .xls, .csv, .txt, .xml, .html, ou gravá-los em um banco de dados

Tabela 4. Características das Ferramentas da Categoria 3 - Repositórios de Dados sobre o Processo de Desenvolvimento de Software

Característica	Descrição
C1	capacidade de se conectar a repositórios de registro de dados de processos de desenvolvimento de software, mesmo sob autenticação
C2	capacidade de retornar um conjunto de dados históricos por *release, commit* ou projeto de um software no repositório. Uma ferramenta será considerada válida se retornar qualquer tipo de informação sobre o processo de desenvolvimento de software, incluindo produtividade da equipe ou tempo para realização de tarefas. Uma ressalva está no fato de que, caso a ferramenta considere projetos de código fonte para extrair informações, somente serão aceitas ferramentas que manipulem código fonte nas linguagens Java ou C#
C3	capacidade de exportar os resultados obtidos nos seguintes formatos: .xls, .csv, .txt, .xml, .html, ou gravá-los em um banco de dados

Questões de Pesquisa: mediante o objetivo apresentado e os critérios para avaliação das ferramentas, foram definidas as seguintes questões de pesquisa, com o objetivo de guiar a execução do mapeamento sistemático:

- (Q1) Quais são as ferramentas disponíveis em cada uma das categorias de repositórios?
- (Q2) Em quais critérios pré-definidos para cada uma das categorias de repositórios as ferramentas se encaixam?

Strings de busca: para cada uma das categorias de repositórios, foram definidas *strings* de busca, conforme a Tabela 5.

Tabela 5. *Strings* de busca para cada categoria

Categoria	*String* de busca
1	((ferramenta OR tool OR plugin) AND (extração OR extract OR mining OR mineração) AND (métricas OR métrica OR metric OR metrics) AND (repositório OR repositórios OR repository OR repositories) AND ((código AND fonte) OR (source AND code)))
2	((ferramenta OR tool OR plugin) AND (extração OR extract OR mining OR mineração) AND (métricas OR métrica OR metric OR metrics) AND (repositório OR repositórios OR repository OR repositories) AND (defeitos OR defeito OR defect OR defects OR bug OR bugs OR "bug tracker"))
3	((ferramenta OR tool OR plugin) AND (extração OR extract OR mining OR mineração) AND (métricas OR métrica OR metric OR metrics) AND (repositório OR repositórios OR repository OR repositories) AND (software AND development AND process))

Bases de dados de publicações: para executar as *strings* de busca, foram consideradas cinco bases de dados eletrônicas de publicações com acesso livre na Universidade Federal de Juiz de Fora (UFJF). As bases usadas foram *Science@Direct*, *EI Compendex*, *IEEE Digital Library*, *ISI Web of Science* e *Scopus*. O protocolo do mapeamento sistemático definido nesta seção permite uma busca formal pelas ferramentas, diferentemente de uma abordagem de pesquisa *ad-hoc* que não possui uma metodologia rigorosa para obter resultados através de investigação [Kitchenham, 2004].

Os resultados obtidos mostram que foi possível recuperar ferramentas proprietárias e livres, sendo que na Categoria 1 foram encontradas 12 ferramentas, sendo que dessas, apenas 2 contemplaram todos os critérios definidos no protocolo (*Analizo* e *Kalibro Desktop*). Na Categoria 2 foram encontradas 22 ferramentas, onde apenas 7 obedeceram aos critérios definidos (*BuCo Reporter*, *Bugzilla*, *Bug Track*, *Redmine*, *Mantis Bug Tracker*, *Hudson* e *Jenkins*). Já na Categoria 3, foram encontradas 5 ferramentas, onde apenas 3 foram consideradas (*Issue Player*, *DIG* e *Disco*). A lista completa de ferramentas recuperadas, bem como demais resultados obtidos, estão disponíveis na página do *GiveMe Metrics*[9].

3.2 Cenários de Uso do *GiveMe Metrics*

A utilização do *framework* pode ser dividida em três diferentes cenários, permitindo ao usuário escolher entre três diferentes tipos de repositórios de dados para analisar: repositório de código fonte, repositório de defeitos ou repositório de dados sobre o processo de desenvolvimento de software, conforme apresentado na Figura 1.

[9] http://www.givemeinfra.com.br/givememetrics.html

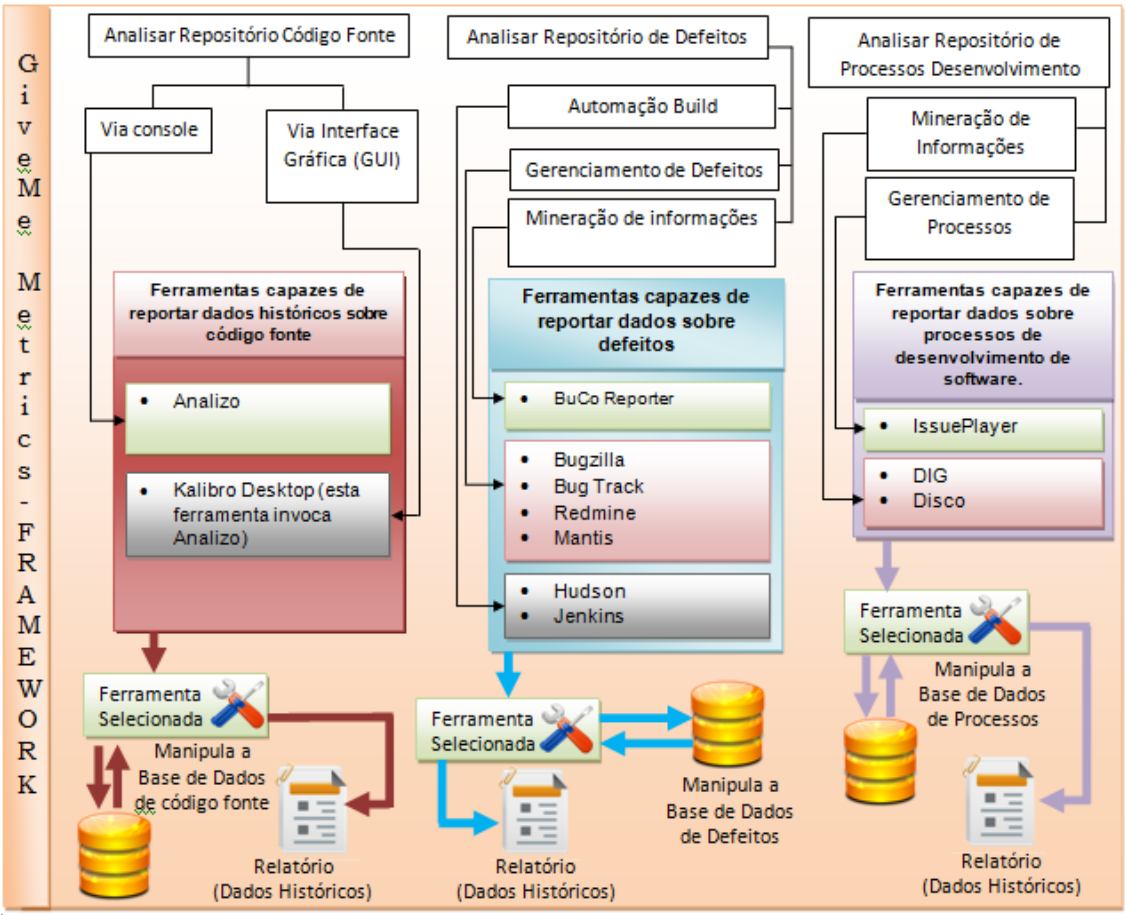

Figura 1. *GiveMe Metrics Framework*

Cenário 1: Analisar Repositório de Código Fonte

O usuário poderá escolher entre uma ferramenta manipulável via console (através de linha de comando) ou uma ferramenta manipulável via interface gráfica do usuário (GUI). Ambas as ferramentas permitem a extração de métricas de código fonte na linguagem Java e são baseadas em software livre. Entre as métricas definidas no critério C1 da Categoria 1 do protocolo de mapeamento, *Analizo* e *Kalibro Desktop* são capazes de extrair as seguintes métricas em cada uma das características de software:

- Tamanho: número de classes, número de métodos por classe, número de subsistemas, número de linhas de código fonte;
- Periodicidade: intervalo de tempo por versão de um projeto;
- Complexidade: profundidade de herança por classe, número de filhos por classe, acoplamento entre objetos, falta de coesão entre métodos e complexidade ciclomática.

Após a seleção da ferramenta, pode-se manipular a base de dados de código fonte, que significa executar a ferramenta sobre as versões de um projeto que estão sendo versionadas no repositório, e obter determinadas métricas sobre cada uma das versões. Ao final, obtém-se um conjunto de dados sobre o software, nesse caso, métricas de código fonte.

Cenário 2: Análise de Repositórios de Defeitos

O mapeamento sistemático realizado permitiu a descoberta de dois formatos diferentes de registro de defeitos, tais como: (i) registros textuais (ou *Tickets*), manualmente cadastrados sobre um defeito encontrado no software, ou (ii) um defeito automaticamente detectado (com uso de ferramentas de verificação de código fonte) ao analisar o código fonte de um software, como problemas no uso dos recursos da linguagem de programação, problemas na API de desenvolvimento e exceções que não foram tratadas. A extração de dados sobre defeitos pode ser feita usando três diferentes grupos de ferramentas, como pode ser visto na Figura 1: (i) ferramentas de automação de *build*, (ii) ferramentas de gerenciamento de defeitos e (iii) ferramentas de mineração de informações em repositórios do tipo *Bugzilla*. Caso os defeitos cadastrados sejam normalmente acessados pelas ferramentas *Hudson* ou *Jenkins* (ambas baseadas em software livre), as mesmas poderão ser usadas para extrair dados sobre os defeitos. Se a organização utilizar ferramentas como *Bugzilla* (software livre e *open source*)*, Bug Track* (software livre)*, Redmine* (software livre) ou *Mantis Bug Tracker* (software livre e *open source*) para gerenciamento de defeitos encontrados em um projeto de software, poderão ser usadas também para extrair dados sobre defeitos. Já o terceiro grupo de ferramentas diz respeito à mineração de informações em repositórios como *Bugzilla*. Essa atividade é possível mediante o uso da ferramenta *BuCo Reporter* (software livre). Após a escolha da ferramenta, a base de dados de defeitos é analisada e dados extraídos dos defeitos são obtidos.

Com as ferramentas *Hudson* e *Jenkins* é possível extrair, através de suas funcionalidades de exportação, dados provenientes de bases de dados de defeitos como o *Bugzilla, Redmine* e *Mantis Bug Tracker*. Isso é possível através da instalação de *plugins* compatíveis. O tipo de dado que pode ser extraído com essas ferramentas depende do tipo de *plugin* e da base de dados de defeitos integrada ao *Hudson* ou *Jenkins*. Em [Hudson, 2015] e [Jenkins, 2015] é possível encontrar informações sobre esses *plugins*.

A partir das ferramentas de gerenciamento de defeitos *Bugzilla, Bug Track, Redmine* ou *Mantis Bug Tracker* é possível extrair dados como defeitos em aberto, defeitos resolvidos, autor dos defeitos, descrição do defeito, dentre outros.

Com a ferramenta *BuCo Reporter* é possível minerar informações como número de defeitos não resolvidos, número de defeitos resolvidos, tempo médio de correção de um defeito, idade média de um defeito não resolvido, número de defeitos detectados depois do lançamento de uma *release*, relação entre defeitos resolvidos e não resolvidos, número de defeitos detectados depois de um período de tempo determinado e porcentagem de defeitos documentados.

Cenário 3: Analisar Repositórios de Processos de Desenvolvimento

O usuário poderá escolher entre o grupo de ferramentas de análise de informações ou o grupo de gerenciamento de processos. No primeiro grupo têm-se as ferramentas *DIG* e *Disco*. No segundo grupo tem-se a ferramenta *Issue Player*.

Com a ferramenta *DIG* é possível obter dados sobre o processo de desenvolvimento de software, bem como quais são as tarefas concluídas e as não concluídas, além do quanto falta para que sejam finalizadas. Há também a coleta de medidas de esforço por membro da equipe no projeto. Entretanto, a documentação

disponível para essa ferramenta não é clara em expor como tais dados são calculados. Há apenas a citação textual de que são coletados, mas não há detalhes, por exemplo, de como é medido o esforço de um membro na equipe. Com a ferramenta *Disco* tem-se a mesma situação, a falta de clareza ao expor como os dados são obtidos. *Disco*, em seu *site*, apresenta uma lista de dados sobre o processo de desenvolvimento que podem ser obtidos, mas não apresenta uma descrição do que significam. Apesar disso, tais ferramentas não foram excluídas do *framework*, pois se encaixam em todos os critérios pré-estabelecidos no mapeamento sistemático. Os dados possíveis de se obter são, entre outros: frequência absoluta, número máximo de repetições, duração total (não está explícito se considera duração de uma tarefa ou projeto) e a duração máxima. Já com a ferramenta *Issue Player* é possível extrair dados como média de tempo dos artefatos no repositório, média de tempo dos defeitos cadastrados pertencentes a um projeto, média de tempo dos arquivos no repositório, média de tempo de construção das funcionalidades de um software.

Além da extração dos dados propriamente ditos, pode-se ainda trabalhar aspectos de visualização de informações, selecionando estratégias para visualizar os resultados obtidos na etapa de extração. Exemplos de estratégias podem ser vistos em [Ribeiro et al., 2013], onde são apresentadas heurísticas utilizadas no cenário de dados governamentais para verificar se as visualizações escolhidas são as mais adequadas aos dados que estão sendo analisados no momento.

3.2 Evolução do *framework GiveMe Metrics*

Em [Tavares et al., 2014] foi apresentada uma prova de conceitos que mostrou a utilização do *framework GiveMe Metrics* na extração de dados históricos de um repositório de defeitos. Isso foi possível graças ao estabelecimento de parceria com o instituto SINAPAD (Sistema Nacional de Processamento de Alto Desempenho). O próximo passo do projeto que engloba o *framework GiveMe Metrics* foi o de estabelecer mais duas parcerias, visando extrair mais dados históricos sobre o ciclo de vida de produtos de software. O objetivo era usá-los, juntamente com os dados já extraídos sobre defeitos, para obter maior conhecimento sobre como se deu a evolução dos produtos que geraram tais dados.

As novas parcerias se deram com empresas de desenvolvimento de software para gestão de empresas, como ERP's. Por questões de confidencialidade, ambas serão chamadas de Empresa Parceira 1 e Empresa Parceira 2, durante todo o texto. Após a definição das novas parcerias, o que se pôde notar é que o *framework GiveMe Metrics* não é capaz de guiar usuários na tarefa de extrair dados históricos como de, por exemplo, repositórios customizados ou desenvolvidos pelas empresas que os possuem. A Empresa Parceira 1 é um exemplo, pois possuía uma solução particular (um repositório desenvolvido e utilizado somente por eles). Isso inviabilizou, a princípio, a utilização de uma das ferramentas contidas no *GiveMe Metrics* para extrair dados sobre o processo de desenvolvimento, pois nenhuma delas se mostrou apta a manipular repositórios próprios ou customizados. Nesse sentido, o *framework* evoluiu, objetivando suportar a extração de dados de repositórios próprios ou customizados, através de uma subcategoria Customizada de ferramentas. Através dessa subcategoria, *drivers* podem ser construídos de forma a estender cada uma das categorias de ferramentas disponíveis no *framework*. A Figura 2 mostra, de forma realçada, a evolução sofrida pelo *framework*, em comparação com primeira versão.

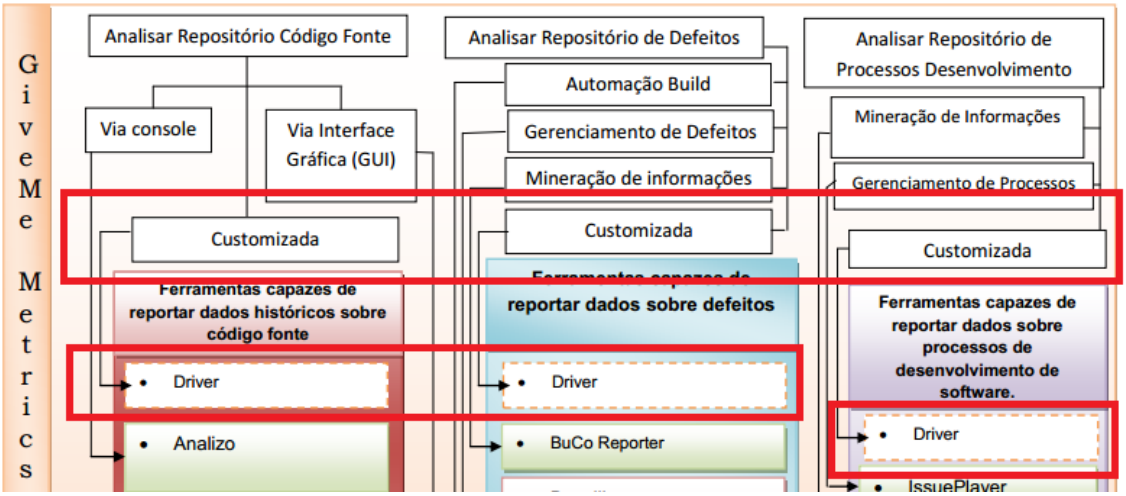

Figura 2. Evolução do *framework GiveMe Metrics*

Em todas as categorias do *GiveMe Metrics*, tais como: (i) Analisar Repositório de Código Fonte, (ii) Analisar Repositório de Defeitos e (iii) Analisar Repositório de Processo de Desenvolvimento de Software, foi criada uma subcategoria chamada Customizada. Essa subcategoria permite a definição de *drivers*, juntamente com as ferramentas de cada uma das categorias citadas. *Driver* é a notação que é dada para ilustrar a possibilidade da implementação de um mecanismo que permita a extração de dados em casos onde os repositórios encontrados sejam próprios ou customizados.

Para que os dados sobre processo de desenvolvimento, e também de defeitos (pois estão no mesmo repositório, no caso da Empresa Parceira 1), pudessem ser extraídos, foi implementado o *DriverEmpresa1*, que permitiu que a condução da extração de dados pudesse ser realizada sempre que necessário. A Figura 3 ilustra o processo utilizado para extrair esses dados.

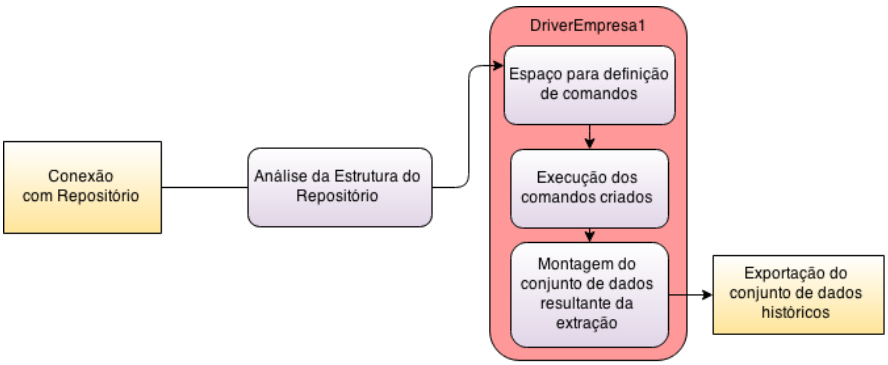

Figura 3. Processo de extração de dados usando o *DriverEmpresa1*

O repositório foi analisado objetivando obter conhecimento sobre sua estrutura e, assim, possibilitar a construção do *driver*. O *DriverEmpresa1* foi definido de forma a permitir a definição de comandos que serão executados sobre a base de dados do repositório customizado da Empresa Parceira 1. A execução dos comandos informados permite que um conjunto de dados históricos seja retornado. A atividade seguinte do *driver* consiste na extração de um conjunto de dados, formatando-os de tal forma que se adequem aos tipos dados suportados pela primeira versão do *framework*. O último passo do processo descrito consiste na exportação dos dados criados, tal como já era suportado pela primeira versão do *GiveMe Metrics*.

Além do desenvolvimento do *DriverEmpresa1*, foi desenvolvido um novo *driver*, dessa vez focando na extração de dados históricos de código fonte. Isso foi necessário dado que nenhuma das ferramentas da categoria que analisa repositórios de código fonte é capaz de retornar *logs* em formato texto contendo todos os métodos e classes que foram alterados na implementação de uma dada versão de software que sofrera manutenção [SVN Book, 2015]. O *driver* implementado para esse objetivo é capaz de extrair dados de repositórios de código fonte SVN [2015] e GIT [2015]. A Figura 4 ilustra o processo de extração de dados usando o *DriverEmpresa2*. Nela é possível ver que as entradas para o *driver* são informações como intervalo de versões a serem analisadas (*range*) e o tipo de repositório que se deseja manipular para obter os dados históricos (SVN ou GIT). Uma implementação que permite a análise de código fonte na linguagem Java foi utilizada.

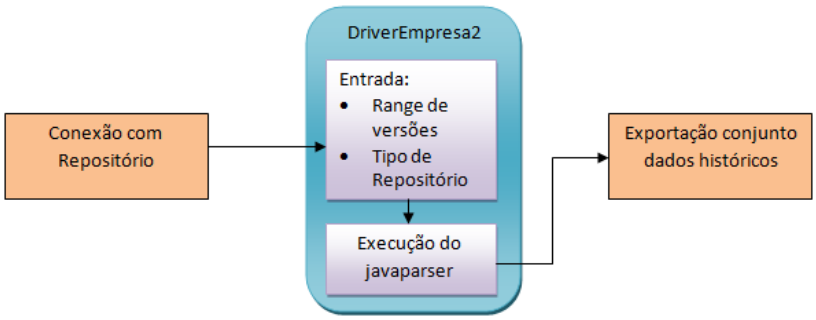

Figura 4. Processo de extração de dados usando o `DriverEmpresa2`

Atualmente, o *framework GiveMe Metrics* está sendo utilizado de forma integrada com uma solução desenvolvida com o objetivo de analisar dados históricos de software chamada *GiveMe Views* [Tavares et al., 2015], que é parte da continuação desta pesquisa. A integração foi necessária dado que a *GiveMe Views* não é capaz de extrair dados históricos, necessitando que outra solução o faça.

GiveMe Views é um *plugin* do ambiente *Eclipse,* uma ferramenta que, ao analisar dados históricos, é capaz de realizar processamentos estatísticos gerando, dentre outras coisas, indicações de classes e métodos que poderão ser afetados caso um método selecionado seja alterado. O processamento estatístico leva em consideração os dados históricos extraídos a partir de *drivers* construídos para o *framework GiveMe Metrics* e permite ao usuário visualizar, além dos possíveis métodos afetados, também a probabilidade estatística (em termos de porcentagem) de um método ser afetado.

4. *GiveMe Infra*: uma infraestrutura baseada em múltiplas visões

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 infraestrutura deve permitir:

1. manter solicitações de mudanças;
2. associar informações de rastreabilidade à solicitação de mudança;

3. manter atividades de manutenção e evolução de software;
4. apoiar tomada de decisão sobre alterações em nível de código;
5. analisar o impacto de alterações em outros módulos/componentes;
6. manter equipes de manutenção;
7. manter dados históricos de projetos;
8. potencializar atividades de compreensão, manutenção e evolução de software.

Como requisito não funcional tem-se a integração com ambiente de compreensão e colaboração de software.

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 e contemplados por 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. 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 5 fornece uma visão geral das principais linhas de atuação da *GiveMe Infra*, separadas por duas categorias, sendo uma delas composta por recursos disponíveis graças à integração com ambiente de compreensão e colaboração.

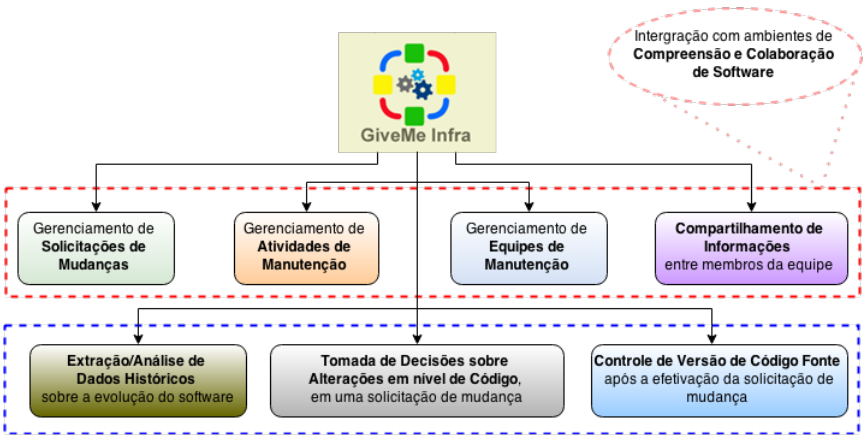

Figura 5. Visão geral das principais linhas de atuação da *GiveMe Infra*.

A Figura 6 mostra uma visão geral da integração de *GiveMe Metrics* e *GiveMe Views*, através da infraestrutura *GiveMe Infra*. Na figura é possível perceber que estão representadas as parcerias com o SINAPAD, com a Empresa Parceira 1 e com a Empresa Parceira 2.

Dependendo do contexto escolhido, dados históricos de software, ou dados de código fonte, serão a entrada para a *GiveMe Infra*, 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.

Os recursos disponíveis na *GiveMe Infra* visam apoiar diferentes atividades de manutenção, estejam elas sendo executadas observando o contexto atual ou o contexto histórico. 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. Essas características são implementadas através das ferramentas de terceiros, descritos em [Carneiro et al., 2012], [Carneiro e Mendonça, 2013] e [Silva et al., 2012].

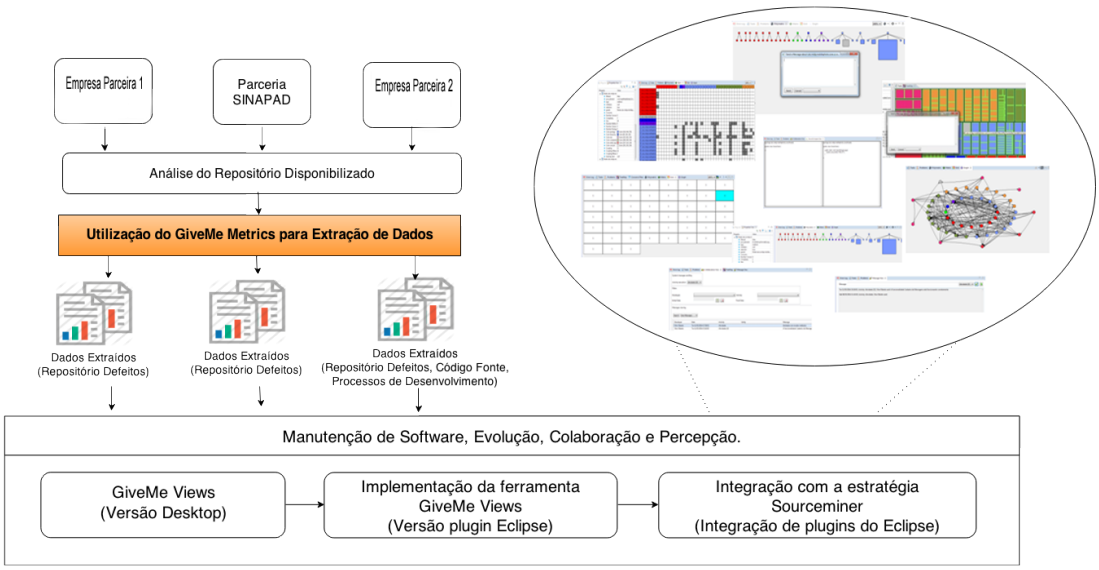

Figura 6. Visão geral da solução

Após a utilização do *GiveMe Metrics* para a extração dos dados históricos, conjuntos de dados são formados representando a saída do *framework*. Os dados extraídos do SINAPAD e da Empresa Parceira 1 passaram pela análise de pesquisadores da Universidade Federal de Juiz de Fora, bem como representantes da Empresa Parceira 1 e do SINAPAD, o que resultou em um conjunto de análises estatísticas que inspiraram a definição de uma solução que engloba manutenção de software, evolução, colaboração e percepção. O primeiro passo no desenvolvimento da solução citada é a implementação da versão *desktop* da ferramenta *GiveMe Views*.

Após a implementação dessa primeira versão, a mesma foi disponibilizada para a Empresa Parceira 1, que a está utilizando principalmente no setor de qualidade. A empresa necessitava de uma solução que permitisse avaliar as solicitações de mudanças concluídas pelos desenvolvedores e, devido à complexidade de uma alteração no código fonte, e ao alto acoplamento entre classes e métodos, muitas vezes determinadas alterações no código fonte de um projeto ficavam sem ser verificadas, e o setor de qualidade tinha dificuldade para identificar quais delas ficaram de fora da manutenção realizada. *GiveMe Views* apoia esse ponto, fornecendo indicações estatísticas de classes e métodos que poderão ser alterados mediante uma solicitação de mudança.

GiveMe Views, por ser uma solução que provê diferentes visualizações gráficas, fornecendo estatísticas provenientes das análises sobre os dados históricos, foi integrada ao *toolkit* de ferramentas do *Sourceminer* [Carneiro e Mendonça, 2013] e ao *Collaborative Sourceminer* [Carneiro et al., 2012]. O *toolkit* contempla a ferramenta AIMV - Ambiente Interativo de Múltiplas Visões [Silva et al., 2012], que fornece múltiplas visões gráficas sobre um conjunto de dados, cujo objetivo é apoiar a análise de código fonte e a geração de informações como, por exemplo, complexidade, número de linhas de código fonte e acoplamento entre classes. Já o *Collaborative Sourceminer*

(2015) é a versão do *Sourceminer* que oferece elementos de colaboração para apoiar a compreensão de software entre equipes geograficamente distribuídas ou co-localizadas.

Silva et al. [2012] afirmam que os ambientes interativos baseados em múltiplas visões 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 permite a análise de dados em diferentes perspectivas. No caso do estudo experimental apresentado neste trabalho, as perspectivas são código fonte e dados históricos sobre a evolução. A integração da *GiveMe Views* com o *Collaborative Sourceminer* possibilita a utilização das visualizações fornecidas pelo *toolkit* com esses objetivos. Além disso, é possível também fornecer ao usuário a visão histórica do software, com base nos dados históricos da sua evolução, e a visão em nível de código fonte, que juntas apoiarão nas tomadas de decisão rumo a manutenções futuras.

É importante destacar que a evolução do *framework* permite que dados históricos sejam extraídos de um conjunto maior de repositórios de dados, e que tais dados podem ser usados não somente pela solução integrada ao *framework* apresentada neste trabalho, que é a ferramenta *GiveMe Views*.

A figura 7 ilustra as dependências entre os componentes da *GiveMe Infra*. Através desta figura é possível destacar as relações com os recursos visuais desenvolvidos no contexto deste projeto. Adicionalmente, são destacadas as integrações com outras ferramentas e ambientes que apoiarão nas atividades de manutenção e evolução colaborativa de software.

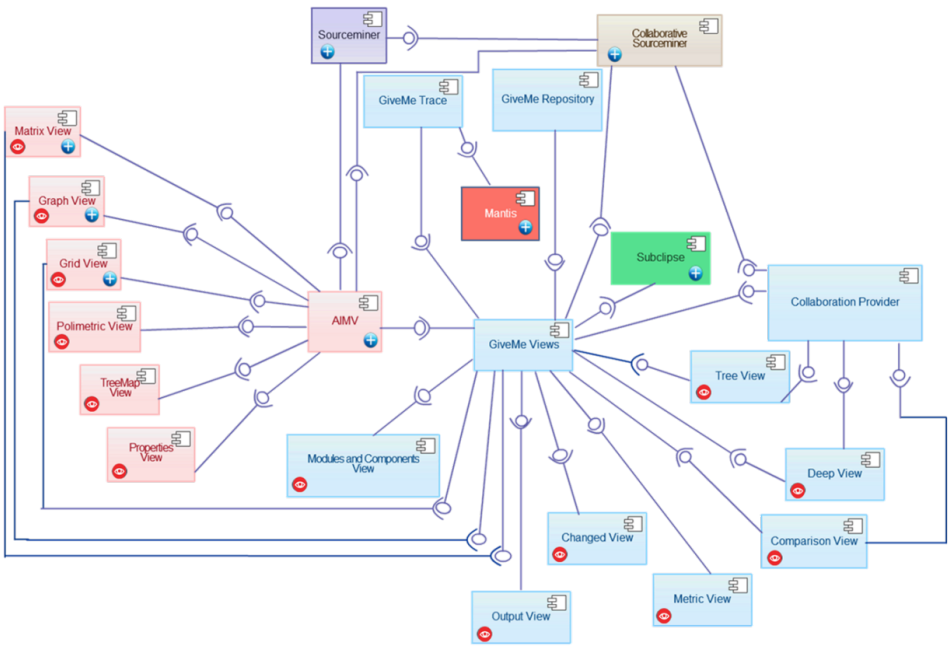

Figura 7. Diagrama de componentes da infraestrutura

O grupo das ferramentas em azul identifica 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 à infraestrutura. Ao todo, *GiveMe*

Infra possui 22 *plugins* integrados com o objetivo de resolver o problema geral deste trabalho.

5. Integrando compreensão e evolução de software: um exemplo de utilização

Os dados históricos que são extraídos com o uso do *GiveMe Metrics* são a entrada para a ferramenta *GiveMe Views*, que os analisa objetivando fornecer subsídios para usuários na tomada de decisões que envolvem novas manutenções. Para tanto, *GiveMe Views* apoia a exploração da evolução do software por meio dos dados analisados, permitindo uma melhor compreensão sobre a evolução do mesmo.

A Figura 8 apresenta a tela principal da ferramenta *GiveMe Views*. Na parte A é possível observar o projeto que será alvo de uma manutenção. Ao clicar com o botão direito sobre ele, aparecerá um menu de opções onde é possível escolher entre duas perspectivas para se obter a compreensão do projeto de código: (i) Perspectiva da Evolução: analisa a evolução do ciclo de vida do software através da utilização de dados históricos extraídos com a ferramenta *GiveMe Views*, ou (ii) Perspectiva Estrutural: analisa a estrutura do código fonte visando obter, por exemplo, métricas de acoplamento e de complexidade ciclomática (possível graças a integração com a ferramenta *Sourceminer* e com o ambiente AIMV). Mais detalhes sobre a utilização da *GiveMe Views* podem ser obtidos em [Tavares et al., 2015].

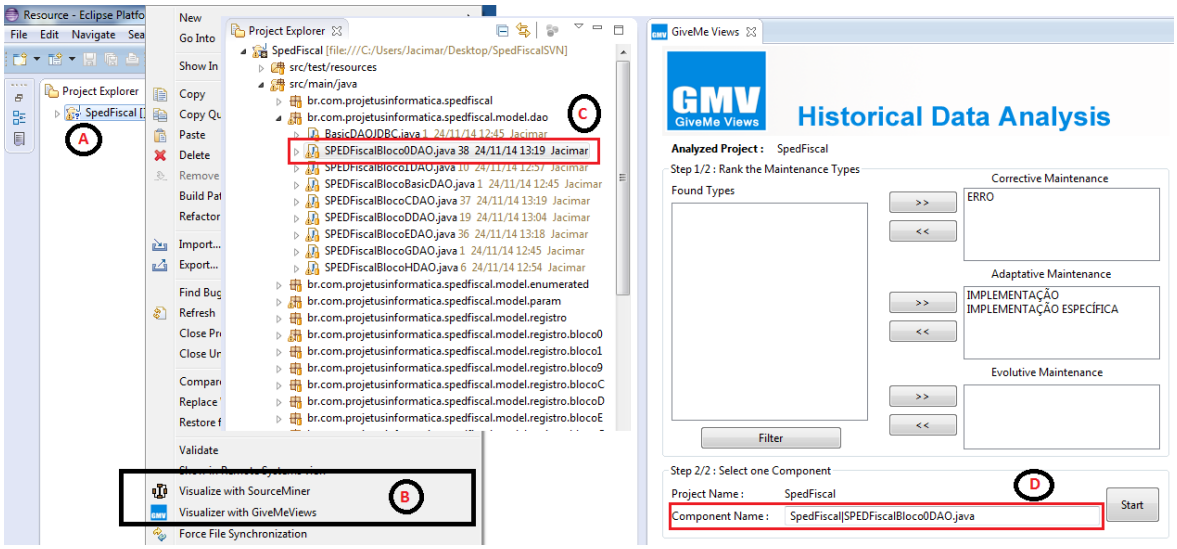

Figura 8. Tela principal da *GiveMe Views*

A Figura 9 mostra a visualização *Graph View,* fornecida pelo AIMV, integrada à *GiveMe Views*. Nela, tem-se um grafo onde os nós representam as classes e as arestas representam a existência de acoplamento entre elas. A visualização foi obtida após a execução da Perspectiva Estrutural no código fonte de um projeto chamado *SpedFiscal*, um ERP contábil, fornecido pela Empresa Parceira 2. Pode-se então obter um auxílio na compreensão de algumas questões relacionadas ao projeto de código fonte analisado, em um dado instante do seu ciclo de vida. A figura ilustra os resultados obtidos ao analisar uma das versões do código fonte do projeto *SpedFiscal*.

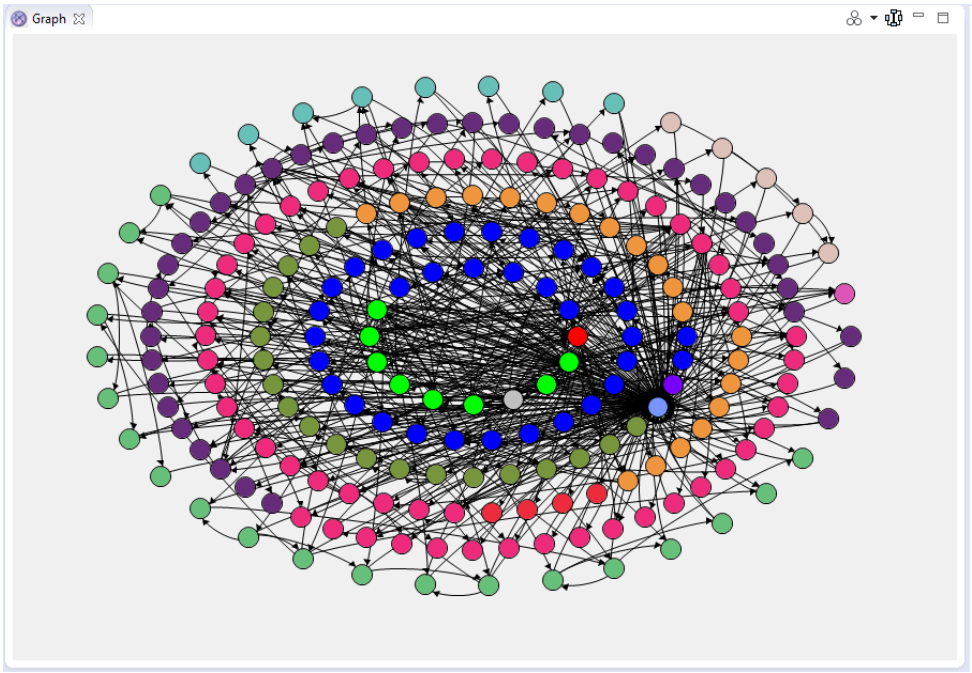

Figura 9. *Graph View* no contexto de uma versão do código fonte do projeto *SpedFiscal*

A Figura 10 mostra a mesma visualização da figura anterior, mas agora exibindo dados na Perspectiva da Evolução. Isso é possível graças aos dados extraídos com o *GiveMe Metrics* e à integração entre as ferramentas *Sourceminer*, *Collaborative Sourceminer* e AIMV com a ferramenta *GiveMe Views*.

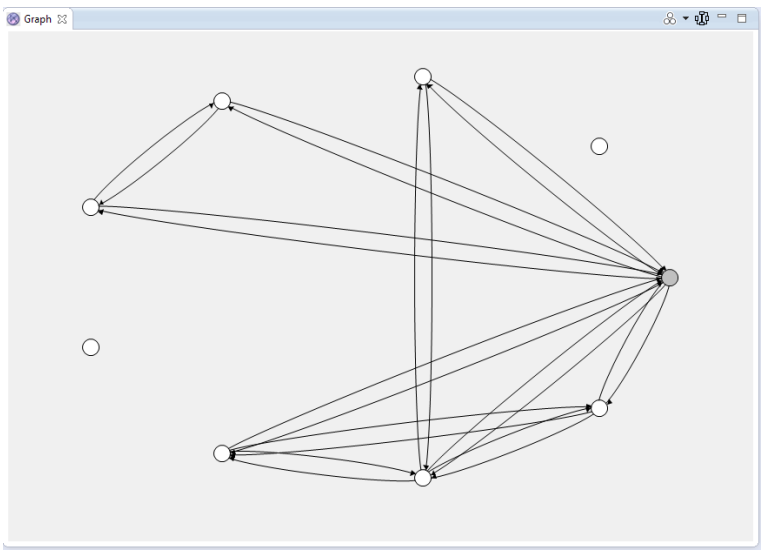

Figura 10. *Graph View* no contexto da evolução do software *SpedFiscal*

Na figura é exibida a relação de acoplamento existente entre todas as classes que sofreram manutenção ao longo das versões do código fonte do projeto *SpedFiscal*. O número de classes representadas é menor na Figura 10 do que na Figura 9. Isto se dá pelo fato de que, no contexto da *GiveMe Views*, se está interessado apenas nas relações entre classes que sofreram algum tipo de manutenção ao longo das versões do projeto ou de um conjunto específico delas. Analisando a evolução do software *SpedFiscal*, quando se está tratando da visualização *Graph View*, é possível descobrir relações entre

classes que não mais existem numa determinada versão porque, por alguma decisão de projeto, foram removidas. No contexto da *GiveMe Views*, essa ligação deletada seria descoberta e exibida, objetivando auxiliar a compreensão sobre a evolução do software.

6. Estudo Experimental

O estudo experimental realizado no contexto deste trabalho tem como premissa a utilização de dados históricos de código fonte extraídos de um repositório real, e tem por objetivo avaliar a solução apresentada. O objetivo é verificar sua viabilidade de uso da infraestrutura em um contexto real de manutenção, através da análise de uma atividade de manutenção e evolução de software. Essa atividade foi conduzida no ambiente de uma empresa de desenvolvimento de software, a Empresa Parceira 2.

Em resumo, será analisada, com o apoio da *GiveMe Views*, uma atividade de manutenção e evolução realizada por profissionais de TI, onde modificações no código de um projeto real foram realizadas. O objetivo é verificar se a infraestrutura oferecida é capaz de indicar as mesmas modificações conduzidas pelos profissionais de TI da Empresa Parceira 2. Ao final, espera-se entender como a utilização dos dados históricos extraídos com a *GiveMe Metrics* apoiaram a realização da avaliação experimental.

6.2 Planejamento do Estudo Experimental

A atividade de manutenção e evolução realizada pelos profissionais de TI da Empresa Parceira 2 está relacionada ao projeto *SpedFiscal*. Esse projeto trata, entre outras funcionalidades, da extração de dados e construção 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. 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 projeto *SpedFiscal*.

6.2.1 Contexto do Estudo Experimental

Trata-se de um estudo controlado, que visa avaliar a efetividade da *GiveMe Infra* no apoio às 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 do *SpedFiscal* e, (ii) as indicações de alterações a serem realizadas no código fonte, fornecidas pela *GiveMe Views,* para o mesmo cenário. A implementação do Bloco K se deu na versão 47 do *SpedFiscal* e os dados históricos considerados pela *GiveMe Infra* compreende às versões 1 a 46 (versões que antecedem o desenvolvimento do Bloco K).

6.2.2 Objetivos e questões de pesquisa

Seguindo o modelo GQM (*Goal – Question - Metric*) [Basili et al., 1994] foram definidos os objetivos deste estudo experimental e, em seguida, as questões de pesquisa. Posteriormente, métricas foram definidas para cada uma das questões de pesquisa.

O objetivo (*Goal*) desse estudo experimental é 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. Para tanto, cenários reais de manutenção foram escolhidos dado que permitem visualizar atividades de manutenção executadas por profissionais da área, em um ambiente real de manutenção. Espera-se, com isso, obter resultados mais condizentes com o cenário real da empresa alvo.

Seguindo o modelo GQM, tem-se: "**Analisar** atividades de manutenção e evolução realizadas em ambientes reais de evolução de software no contexto de manutenção e evolução de software com a **finalidade** de avaliá-las com respeito à corretude e completude das alterações realizadas em **comparação** com as indicações estatísticas de alterações fornecidas pela infraestrutura *GiveMe Infra*". A Tabela 6 apresenta as questões de pesquisa e as métricas associadas.

Tabela 6. Questões e métricas de pesquisa

Questões (*Questions*) e Métricas (*Metrics*)
Q1: o uso da *GiveMe Infra* provê meios de identificar quais módulos/componentes podem ser afetados, 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. • Métrica 01: porcentagem de acerto obtida através do número de indicações de alterações de métodos a partir de modificações no código fonte, em comparação com o número correto de indicações de alterações fornecidas pela *GiveMe Infra;* • Métrica 02: número de indicações extras, referente ao conjunto de indicações de alterações em métodos que pode ser fornecido pela *GiveMe Infra,* mas que não refletem alterações realizadas pela equipe de manutenção.
Q2: *GiveMe Infra* apoia os desenvolvedores nas atividades de manutenção e evolução de software futuras? Objetivo: verificar como se dá o apoio a atividades de manutenção e evolução de software usando a *GiveMe Infra*. • Métrica 03: número de indicações não fornecidas pela *GiveMe Infra* mas que resultaram em alterações no software analisado.

Para a realização do cálculo das métricas, visando responder às questões de pesquisa, estão previstas a utilização de métodos estatísticos específicos, sendo (i) análise de correlação de dados [Levin e Forde, 2012] que permite estabelecer a correlação existente entre diferentes conjuntos de dados. Esse estudo experimental será útil para definir a correlação existente entre o número de indicações de alterações fornecidas pela *GiveMe Infra* com o número de indicações corretas fornecidas, se comparado às alterações realizadas pela equipe de TI da Empresa Parceira 2; e (ii) distribuição de frequência simples e conjunta [Meyer, 1983], que permite descobrir quais métodos poderiam ser afetados a partir das alterações realizadas no código fonte por parte da equipe de TI da Empresa Parceira 2.

Com relação às questões de pesquisa, ainda serão consideradas as seguintes variáveis dependentes e independentes:

Variáveis Independentes: as versões do projeto *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 de mudança) 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, e os

tratamentos sendo o conjunto das versões anteriores ao desenvolvimento do Bloco K (compreende as versões 1 a 46), e a última versão do *SpedFiscal* (versão 47) que corresponde a implementação do Bloco K.

Variáveis Dependentes: a manutenção realizada pela equipe de TI da Empresa Parceira 2, que é uma medida do apoio prestado pela *GiveMe Infra* na avaliação das atividades de manutenção e evolução analisadas.

6.2.3 Ferramenta Base

O objetivo é utilizar os recursos da infraestrutura *GiveMe Infra* para indicar alterações a serem feitas no 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, a partir de seus dados históricos. Trata-se de um ERP contábil para empresas, desenvolvido e licenciado pela empresa.

6.2.4 Sujeitos e equipes

Neste estudo experimental, relacionado à verificação das manutenções realizadas em um contexto real, não foram selecionados sujeitos e equipes na execução, além dos pesquisadores responsáveis por este estudo. É importante ressaltar que as atividades de verificação de manutenções em outros contextos podem implicar na necessidade da escolha de sujeitos, diferentemente desses.

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 11) mostra o fluxo das atividades.

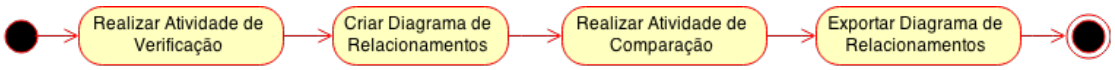

Figura 11. Diagrama de atividades

Assim, as atividades são:

- **Realizar Atividade de Verificação**: diz respeito à análise das manutenções executadas no código fonte do *SpedFiscal* pela equipe de manutenção e catalogação dos métodos afetados. Isso foi possível analisando os históricos das alterações conduzidas, utilizando para tal as indicações dos *diffs* [SVN Book, 2015] fornecidos pela ferramenta Tortoise SVN [2014];
- **Criar Diagrama de Relacionamentos**: trata-se de uma representação em forma de grafos constituída por vértices (representando métodos alterados) e arestas (representando os relacionamentos entre eles). Com isso, pode-se criar grafos que mostrem quais métodos foram alterados na implementação do Bloco K e quais métodos foram afetados com as alterações. Os relacionamentos serão calculados utilizando distribuição de frequência discutidas em [Meyer, 1983];
- **Realizar Atividade de Comparação**: diz respeito à análise, através da *GiveMe Infra,* de cada um dos métodos alterados pela equipe de manutenção na implementação do Bloco K. O objetivo dessa atividade é gerar um diagrama de relacionamentos com as indicações da *GiveMe Infra* com o intuito de apoiar a avaliação das atividades de manutenção e evolução analisadas. 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 nessa versão;

- **Explorar Diagrama de Relacionamentos**: 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 por uma visualização pertencente à *GiveMe Infra.* O objetivo é também apoiar a avaliação das atividades de manutenção e evolução.

6.2.5 Operacionalização

O primeiro passo deste estudo experimental consiste 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 utilizando a 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*, de forma a poder comparar as alterações feitas pela equipe de TI com as indicações feitas pela infraestrutura.

Após a catalogação dos métodos alterados na implementação do Bloco K na versão 47, foi criada uma matriz de relacionamentos (usando distribuição de frequência simples e conjunta [Meyer, 1983]). Foi possível estabelecer as relações entre os métodos alterados para a implementação do Bloco K. De posse dessas informações, foi possível responder perguntas relativas a quantos métodos podem ter sido foram afetados quando um método foi alterado na implementação do Bloco K (podem ser afetados por possuírem algum tipo de relacionamento com o método alterado). Nesse sentido, o próximo passo consistiu na comparação com as indicações fornecidas pela *GiveMe Infra* para os mesmos métodos alterados pela equipe de TI (atividade de comparação). Cada uma das comparações seguiu o modelo de execução definido da seguinte forma: (i) quando a equipe de TI realizou alterações em um método, foram conduzidas modificações também em outros métodos; (ii) através da *GiveMe Infra* verifica-se os métodos que podem ser afetados quando um determinado método sofrer alteração; (iii) estabelece-se um comparativo entre o que foi modificado na prática (na versão 47 do *SpedFiscal*), e o que foi indicado pela *GiveMe Infra* como possíveis alterações.

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

6.2.6 Coleta de dados

A coleta de dados no estudo experimental foi utilizada para registro dos dados obtidos na atividade de verificação. Todos os métodos alterados pela equipe de TI na implementação do Bloco K foram registrados.

Os dados de corretude, que são usados na validação de informações a serem analisadas, foram fornecidos pelas ferramentas *Tortoise* SVN e pela própria *GiveMe Infra*, e foram utilizados na validação dos dados do estudo experimental e na avaliação da atividade de manutenção. Como exemplo, tem-se a lista de métodos alterados na implementação do Bloco K.

Com o planejamento do estudo experimental definido, bem como as questões de pesquisa estabelecidas, foi realizada a execução do estudo experimental, e os seus resultados são descritos a seguir.

6.3 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 informações sobre ela.

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 afetados. Após a identificação, foram estabelecidas as relações entre os métodos que foram alterados em conjunto, seguindo o princípio da distribuição de frequência conjunta proposta por Meyer [1983]. Em resumo, foram identificados 5 métodos alterados pela equipe de TI durante a implementação do Bloco K. A Figura 12 exibe os cinco métodos alterados e os métodos que eles afetariam, dada a distribuição de frequência calculada.

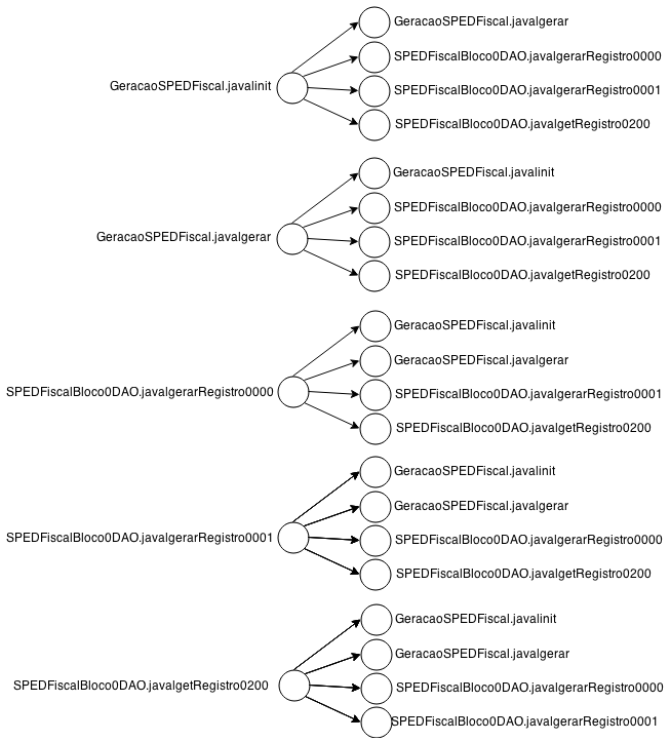

Figura 12. Relacionamentos entre métodos alterados pela equipe de TI

O próximo passo consiste na execução da Atividade de Comparação utilizando a *GiveMe Infra*, que mostrou que, quando a equipe de TI realizou alterações no método GeracaoSPEDFiscal.java|init (Figura 13) foram conduzidas modificações, por exemplo, nos métodos getRegistro0200 e gerarRegistro1001 da classe SPEDFiscalBloco0DAO.java. A verificação na *GiveMe Infra* da influência de uma alteração no método GeracaoSPEDFiscal.java|init (considerando dados históricos das versões anteriores, 1 a 46) mostra quais dos métodos de fato alterados foram indicados

pela *GiveMe Infra* e quais dentre os indicados não foram alterados na implementação do Bloco K. A figura foi gerada com as exportações dos diagramas de relacionamentos obtidos após as verificações de métodos afetados realizadas com a *GiveMe Infra*. Os métodos marcados por um quadro são os que correspondem às modificações realizadas de fato pela equipe de TI da Empresa Parceira, na versão 47.

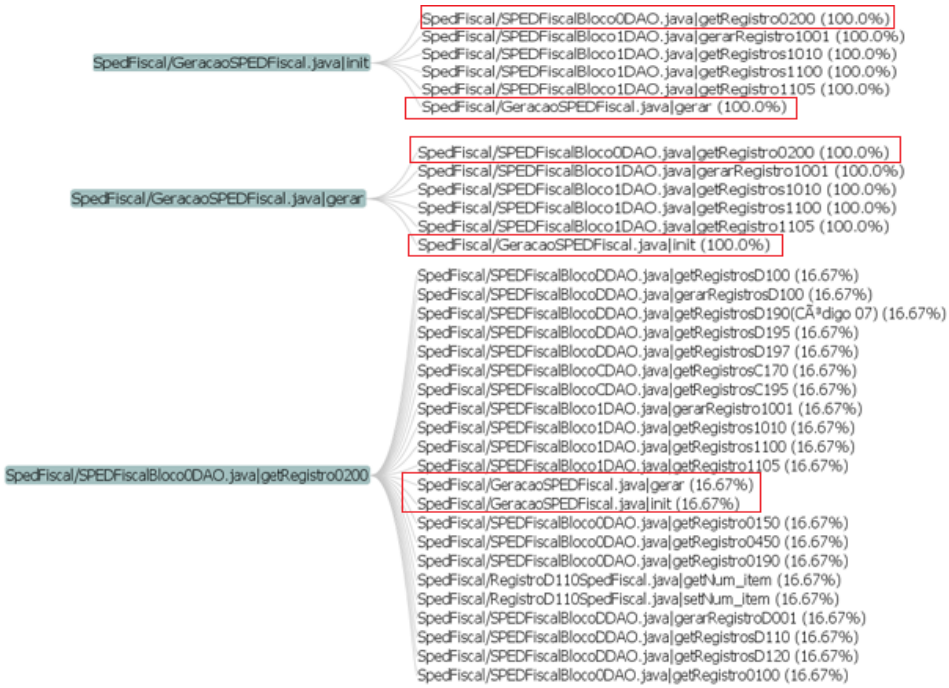

Figura 13. Verificação das relações entre os métodos com a *GiveMe Infra*

Ao analisar a figura, percebe-se que não foi possível verificar, a partir da *GiveMe Infra*, alterações a partir dos métodos gerarRegistro0001 e gerarRegistro0000 da classe SPEDFiscalBloco0DAO.java, indicados no cálculo de distribuição de frequência da Figura 12. Isso ocorreu porque a *GiveMe Infra* não foi capaz de indicar possíveis pontos que podem ser afetados quando esses métodos forem alterados. O motivo é que o conjunto dos dados históricos, das versões 1 a 46, não apresentaram nenhuma manutenção nesses métodos no projeto *SpedFiscal*. Uma investigação no repositório de versionamento de código fonte do projeto mostrou que a primeira manutenção realizada nesses métodos foi após o início da atividade de manutenção que levou à implementação do Bloco K, ou seja, na versão 47, justificando a inexistência de indicações de mudança. Caso o conjunto de dados históricos a ser analisado não contemple alterações em um dado método ao longo do ciclo de vida do software, não será possível estimar uma alteração nele. Entretanto, esse problema pode ser minimizado à medida que, ao longo tempo, novos registros de manutenções sejam armazenados.

Outro dado a ser considerado na figura refere-se à porcentagem calculada para os métodos marcados em vermelho, que apresentaram a maior porcentagem dentre todos os demais que podem ser afetados. Ao mesmo tempo em que isso é um indicativo de que as chances deles serem afetados (como de fato foram na implementação do Bloco K) são as maiores dentre todos, a mesma figura mostra que outros métodos têm chances, estatisticamente iguais, de serem afetados, 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, podendo comprometer a completude das atividades de manutenção e evolução realizadas.

A comparação entre os métodos que foram apresentados nas Figuras 12 e 13 mostra que o número de métodos indicados com a *GiveMe Infra* foi maior, neste caso, que o número de métodos alterados na versão 47, como nos métodos *init* e *gerar* da classe GeracaoSPEDFiscal.java, e o método getRegistro0200 da classe SPEDFiscalBloco0DAO.java, sendo o último o que sofreu maior variação (2 indicações de métodos realmente afetados em um total de 22 indicados na verificação).

A Tabela 7 apresenta um resumo das informações geradas na Atividade de Comparação. Através dela é possível observar que o número de métodos alterados mediante a implementação do Bloco K é menor que o número de indicações de métodos afetados, em relação à 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 extras fornecidas pela *GiveMe Infra*. Na tabela é utilizado o conceito de métodos afetados em relação aos métodos que aparecem como afetados mediante as alterações em outros métodos. Como exemplo, tem-se o método GeracaoSPEDFiscal.java|gerar que foi afetado por alterações no método GeracaoSPEDFiscal.java|init.

Tabela 7. Resumo final dos dados da verificação realizada

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

A tabela ainda mostra que o somatório de todos os métodos afetados na manutenção realizados na implementação do Bloco K é igual a 20. Isto quer dizer que, a cada método modificado, foram ou não conduzidas modificações nos métodos que possuem algum tipo de relacionamento com ele, representando o somatório das modificações conduzidas (Atividade de Verificação). Já o somatório das 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 afetados. Há também a possibilidade de a equipe que efetuou a implementação do Bloco K, ter analisado todos os pontos indicados pela *GiveMe Infra* mesmo sem ter tido suporte da infraestrutura, mas julgaram que não eram necessárias mais alterações do que aquelas realizadas.

O próximo passo consiste nos cálculos das métricas previstas para as questões de pesquisa definidas no planejamento do estudo, objetivando avaliar as atividades de manutenção e evolução realizadas. A Tabela 8 mostra o resultado das métricas calculadas. Nela também é utilizado o conceito de métodos afetados em referência aos métodos que aparecem como afetados mediante as alterações em outros métodos. Como

exemplo, tem-se o método GeracaoSPEDFiscal.java|gerar que foi afetado por alterações no método GeracaoSPEDFiscal.java|init.

Tabela 8. Métricas calculadas

	[∑ (número de métodos afetados)] - 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 número de afetados por alterações no código fonte em comparação com o número correto de indicações de métodos afetados fornecidos pela *GiveMe Infra*) é possível perceber que, dos 20 métodos afetados na implementação do Bloco K, a *GiveMe Infra* foi capaz de indicar 6, o que corresponde a cerca de 30% do total. Na prática, isso 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 e também a corretude da atividade de manutenção e evolução.

A Métrica 2 (número de indicações extras, refere-se ao conjunto de indicações de métodos afetados que pode ser fornecido pela *GiveMe Infra* mas que não refletem alterações realizadas pela Empresa Parceira 2) mostra que o número de indicações extras fornecidas pela *GiveMe Infra* é igual a 28. Este número se refere às indicações de alterações feitas pela infraestrutura, mas que não foram realizadas pela equipe de TI, provavelmente por dois motivos: (i) pelo fato da equipe não ter de fato considerado esses pontos, ou (ii) por não ter analisado esses métodos afetados, dado que sua experiência mostra que não seria necessário. Em particular, no primeiro caso, isso pode ser um problema, pois pode levar à inserção de defeitos no software que, se não forem detectados nos testes ou pelo setor de qualidade, poderão chegar até o usuário final.

As análises realizadas até o presente momento visam dar sustentação às respostas das questões de pesquisa deste estudo experimental. A questão de pesquisa Q1 diz respeito ao apoio da *GiveMe Infra* na tarefa de identificar quais pontos podem ser afetados, mediante à necessidade de se alterar um módulo ou componente específico. Os resultados obtidos com o cálculo das métricas mostram que a *GiveMe Infra* foi capaz de indicar 30% das verificações realizadas. É importante destacar que 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*. Esse valor foi obtido para este estudo em particular, e considerou a base de dados existente e a natureza da manutenção realizada.

Para a questão de pesquisa Q2 foi investigado se a *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 e as verificações feitas pela *GiveMe Infra*. Com o resultado da Métrica 2, não é possível afirmar que a *GiveMe Infra* apoiaria em menor grau a equipe de TI 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 necessários. Outro fator que impede afirmar que a *GiveMe Infra* apoiaria em menor grau a equipe de TI, mesmo indicando 30% dos métodos alterados, está ligado ao conjunto de dados históricos utilizados, uma vez que o apoio dado pode ser maior ou menor, caso esses dados históricos 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 por Levin e Forde [2012]. Esse cálculo buscou descobrir se, na 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 alterado na implementação do Bloco K. Coeficientes de correlação indicam a intensidade e direção da correlação dos dados [Levin e Forde, 2012]. Nesse sentido, foi calculado o coeficiente da correlação de *Pearson* entre o número de indicações totais por método (NIE) e número de indicações corretas sobre o que de fato foi alterado (NIC), através da fórmula:

$$\frac{r = \Sigma(A)(B)}{\sqrt{\Sigma(A^2)\Sigma(B^2)}} = SP/\sqrt{SQ(NIE)SQ(NIC)}$$

Onde, A é o valor do desvio padrão obtido pela subtração do NIE e a média do somatório dos seus valores. Já o valor de B é calculado pelo desvio padrão obtido pela subtração do NIC e a média do somatório de seus valores. Considera-se então a Tabela 9, que contém a lista de métodos alterados na implementação do Bloco K. Os dados do NIE e NIC foram obtidos com base na contagem do número de métodos afetados.

Tabela 9. Lista dos métodos alterados

Métodos alterados	NIE	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 10 é complementar à Tabela 9, pois apresenta os cálculos dos coeficientes da correlação. A última coluna da tabela 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 última coluna da tabela.

Tabela 10. Cálculos do coeficiente da correlação

Métodos alterados	NIE	NIC	A	B	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
Somatório (SP)	34	6	-	-	27,2
Média	6,8	1,2	-	-	-

Dado que o valor de SP é um valor igual a 27,2 (maior que zero), indica que há uma associação dita positiva, entre NIE e NIC. Resta agora descobrir a intensidade da correlação entre eles.

Nesse ponto, levando-se em consideração os dados da Tabela 11, têm-se as variáveis necessárias para a efetivação do cálculo da Correlação de *Pearson* (r):

$$r = 27{,}2/\sqrt{(324{,}8)(4{,}8)}$$

$$r = 27{,}2/39{,}48$$

$$r = +0{,}68$$

Tabela 11. 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
$SQ(NIE) = \sum(A^2) = 324{,}8$	$SQ(NIC) = \sum(B^2) = 4{,}8$

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

Ao final do estudo experimental foi possível observar as situações nas quais os dados históricos extraídos com o *framework GiveMe Metrics* apoiaram a compreensão da manutenção realizada em um ambiente real na indústria através da infraestrutura *GiveMe Infra*. Foi possível verificar que as limitações dos dados históricos usados influenciaram diretamente nos resultados do estudo, mas à medida que novos conjuntos de dados forem obtidos e analisados, outros resultados poderão ser obtidos em avaliações futuras.

6.4 Ameaças à Validade

As ameaças à validade deste estudo experimental são relacionadas à 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 seriam considerados satisfatórios.

7. Considerações finais e trabalhos futuros

Este trabalho apresentou a evolução do *framework GiveMe Metrics*, capaz de extrair dados de três diferentes tipos de repositórios de dados históricos, bem com sua integração com a ferramenta *GiveMe Views*. Para tanto, foi criada uma infraestrutura baseada em múltiplas visões interativas para apoiar a evolução distribuída de software, chamada *GiveMe Infra*. Seu desenvolvimento foi baseado em um problema inicialmente encontrado na Empresa Parceira 1, o qual se relacionava à falta de uma solução 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 para as atividades de manutenção. Foi verificado que esse mesmo problema acontecia com a Empresa Parceira 2 a qual disponibilizou projetos de software e repositórios de dados históricos, como repositórios de código fonte e de solicitação de mudanças.

Através da infraestrutura *GiveMe Infra* é possível calcular, inclusive, o acoplamento existente entre classes e métodos de diferentes sistemas, e não somente entre classes e métodos do mesmo projeto. Esse recurso está disponível para utilização pela Empresa Parceira 1, dado que possuem dados históricos que permitem esse tipo de análise; e (iii) a redução do número de reaberturas de solicitações de mudança, por não terem sido aprovadas pela equipe de qualidade, comprometendo a corretude da manutenção realizada. Com a utilização da infraestrutura, indicações de pontos a serem alterados no código reduziram as chances da ocorrência de reaberturas.

As contribuições deste trabalho estão na integração com diferentes ferramentas que apoiam a compreensão de software em dois diferentes contextos: (i) que se refere à análise do projeto de código de um software disponível no espaço de trabalho do ambiente de desenvolvimento *Eclipse*; e (ii) que se refere à análise de informações obtidas ao longo do ciclo de manutenção de um software. Sendo assim, equipes de manutenção e evolução de software têm 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*.

Adicionalmente, 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 sejam inseridas diretamente nas visualizações integradas à infraestrutura. Nesse sentido, os recursos disponíveis na infraestrutura são passíveis de serem utilizados em um ambiente distribuído de manutenção de software.

Outra contribuição deste trabalho está na possibilidade de integração, na *GiveMe Infra*, de outras ferramentas e recursos visuais. 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.

Como trabalhos futuros, espera-se estabelecer novas parcerias com empresas que possuam repositórios de código fonte, de defeitos e de processos de desenvolvimento de software, para que seja possível a extração de dados pelo *framework GiveMe Metrics* e, com isso, ampliar a avaliação dessa proposta. Uma base de dados, contendo métricas provenientes dos dados extraídos com o *framework GiveMe Metrics*, será criada e as informações obtidas através da análise dos dados, poderá ser armazenada e mantida para consulta por pesquisadores, profissionais da indústria ou através de ferramentas que necessitem acessar as informações contidas nessa base de dados.

Referências

Anslow, C. (2010) “Co-located collaborative software visualization”. Human Aspects of Software Engineering, ACM.

Anslow, C., Marshall, S., Noble, J., and Biddle, R. (2013). “Sourcevis: Collaborative software visualization for co-located environments”. In: Software Visualization (VISSOFT), 2013 First IEEE Working Conference On, p. 1-10.

Araújo, M. A. P., Monteiro, V. F., Travassos, G. H. (2012) “Towards a model to support studies of software evolution”. In: Proceedings of the ACM-IEEE international symposium on Empirical software engineering and measurement - ESEM '12, ACM Press, p. 281-290.

Basili, V. R., Caldiera, G. H., Rombach, D. (1994) “The Goal Question Metric Approach”. Chapter in Encyclopedia of Software Engineering, Willey.

Bugzilla (2013). Disponível em http://www.bugzilla.org/features/. Acesso em: novembro de 2013.

Carneiro, G. F. ; Mendonça, M. G. . (2013) "SourceMiner: Towards an Extensible Multi-perspective Software Visualization Environment". In: Enterprise Information Systems - 15h International Conference, ICEIS 2013, Angers, France, July 4-7, Revised Selected Papers. 1ed.: Springer International Publishing, v. 190, p. 242-263.

Carneiro, G. F., Conceição, C. F. R., David, J. M. N. (2012) "A Multiple View Environment for Collaborative Software Comprehension". In: International Conference on Software Engineering Advances (ICSEA), p. 15-21.

Collaborative Sourceminer (2015). Disponível em http://www.sourceminer.org/collaborativeVersion.html. Acesso em: maio de 2015.

Dambros, M., Lanza, M. (2010) "Distributed and Collaborative Software Evolution Analysis with Churrasco". Sci. Comput. Program, p. 276-287.

GIT (2015). Disponível em https://git-scm.com/. Acesso em: maio de 2015.

GiveMe Infra (2015). Disponível em http://www.givemeinfra.com.br. Acesso em: junho de 2015.

Hudson (2015). Disponível em http://hudson-ci.org/PluginCentral3/. Acesso em: Maio de 2015.

Jenkins (2015). Disponível em https://wiki.jenkins-ci.org/display/JENKINS/Plugins. Acesso em Maio de 2015.

Kevic, K., Muller, S. C., Fritz, T., Gall, H. C. (2013) "Collaborative bug triaging using textual similarities and change set analysis". Cooperative and Human Aspects of Software Engineering (CHASE), 2013 6th International Workshop on, p. 17-24.

Kitchenham, B. (2004) "Procedures for performing systematic reviews". Technical Report, TR/SE-0401, Department of Computer Science, Keele University, Keele,UK.

Kitchenham, B. A., Charters, S. (2007) "Guidelines for performing systematic literature reviews in software engineering". Tech. Rep. EBSE-2007-01. KeeleUniversity.

Levin, J., Fox, J. A., Forde, D. R. (2012) "Estatística para Ciências Humanas". 11ª ed. São Paulo: Pearson Education do Brasil.

Ligu, E., Chaikalis, T., Chatzigeorgiou, A. (2013) "BuCo Reporter: Mining Software and Bug Repositories", Sixth Balkan Conference in Informatics, p. 121-127.

Lungu, M., Lanza, M., Nierstrasz, O. (2014) "Evolutionary and collaborative software architecture recovery with Softwarenaut". Sci. Comput. Program, p. 204-223.

Mantis Bug Tracker (2013). Disponível em http://www.mantisbt.org/. Acesso em: novembro de 2013.

Mao, C. (2011) "Structure visualization and analysis for software dependence network". Granular Computing (GrC), 2011 IEEE International Conference on, p. 439-444.

Meyer, P. L. (1983) "Probabilidade – Aplicações à Estatística". 2ª ed. LTC – Livros Técnicos e Científicos Editora LTDA.

Pedroso, L., Revoredo, K., Baião, F. (2013). “Uma Abordagem Baseada em Mineração de Dados para Apoio ao Ciclo de Vida de Projetos de Pesquisa e Inovação”. IX Simpósio Brasileiro de Sistemas de Informação. João Pessoa, PB, p. 710-721.

Poncin, W., Serebrenik, A., Van den Brand, M., (2011) “Process Mining Software Repositories”. Software Maintenance and Reengineering (CSMR), 15th European Conference on, p. 5-14.

Redmine (2013). Disponível em http://www.redmine.org/. Acesso em: novembro de 2013.

Ribeiro, F., Caetano B., Paula, M., Chaves, M., Silva, V., Rodrigues, S., Souza, J. M. (2013). “Heurísticas para Visualização de Dados”. IX Simpósio Brasileiro de Sistemas de Informação, João Pessoa, p. 744-755.

Silva, A. N., Carneiro, G. F., Zanin, R. B., Dal Poz, A. P., Martins, E. F. O. (2012). “Propondo uma Arquitetura para Ambientes Interativos baseados em Multiplas Visões”. II Workshop Brasileiro de Visualização de Software, p. 1–8.

Steinmacher, I., Chaves, A. P., Gerosa, M. A. (2012) “Awareness Support in Distributed Software Development: A Systematic Review and Mapping of the Literature”. Journal of Computer Supported Cooperative Work (JCSCW), v. 22, p. 113-158.

SVN Book (2015). Disponível em http://svnbook.red-bean.com/en/1.7/svn.ref.svn.c.log.html. Acesso em: maio de 2015.

SVN (2015). Disponível em https://subversion.apache.org/. Acesso em: maio de 2015.

Syeed, M. M., Mahbubul, I. H., Csaba, B. (2013) “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. ACM, Pages 273 , 8 pages.

Tavares, J. F., Braga, R., David, J. M. N., Araújo, M. A. P., Campos, F., Carneiro, G. F. (2015) “GiveMe Views: uma ferramenta de suporte a evolução de software baseada na análise de dados históricos”. XI Simpósio Brasileiro de Sistemas de Informação (SBSI), p. 55-62.

Tavares, J. F., David, J. M. N., Araújo, M. A. P., Braga, R., Campos, F. C. A., Carneiro, G. F. (2014), “GiveMe Metrics – Um framework conceitual para extração de dados históricos sobre evolução de software”. X Simpósio Brasileiro de Sistemas de Informação (SBSI), p. 44-55.

Telea, A.; Voinea, L. (2005) “Interactive Visual Mechanisms for Exploring Source Code Evolution”. Visualizing Software for Understanding and Analysis. VISSOFT 2005. 3rd IEEE International Workshop on, p. 1-6.

Telea, A.; Auber, D. (2008) “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, p. 831-838.

Tortoise SVN (2014). Disponível em http://tortoisesvn.net/. Acesso em: novembro de 2014.

Voigt, S.; Bohnet, J.; Dollner, J. (2009) “Object aware execution trace exploration”. Software Maintenance, ICSM 2009. IEEE International Conference on, vol., no., p. 201-210.