Um *framework* para apoiar estimativa de esforço em atividades de manutenção e evolução de software

Alternative Title: A framework to support effort estimation on software maintenance and evolution activities

Marcos Alexandre Miguel[1], Marco Antônio P. Araújo[1,2], José Maria N. David[1], Regina Braga[1]

[1]Programa de Pós Graduação em Ciência da Computação – Universidade Federal de Juiz de Fora (UFJF)
[2]Instituto Federal de Educação, Ciência e Tecnologia do Sudeste de Minas Gerais – Campus Juiz de Fora (IF Sudeste MG)

marcos.miguel@ice.ufjf.br, {marco.araujo, jose.david, regina.braga}@ufjf.edu.br

RESUMO

O surgimento de métodos ágeis no desenvolvimento de software tem apresentado muitas oportunidades e desafios para pesquisadores e profissionais da área. Um dos principais desafios é a estimativa de esforço para desenvolvimento ágil de software. Quando a estimativa de esforço não está bem definida ou é imprecisa, os resultados obtidos podem refletir diretamente na entrega do software, causando insatisfação do cliente ou a diminuição da qualidade do produto, levando assim a necessidade de novos mecanismos que possam auxiliar nesse processo. Diante dessa necessidade, este trabalho apresenta um *framework*, chamado *GiveMe Effort*, para apoiar as atividades de estimativa de esforço na manutenção e evolução de software em métodos ágeis. A solução se baseia em dados históricos de requisições de mudanças, associados à manutenção e evolução de software.

Palavras-Chave

Estimativa de Esforço, Manutenção de Software, Evolução de Software, Visualização de Software.

ABSTRACT

The emergence of agile methods in software development has brought many opportunities and challenges for researchers and professionals. A major challenge is the effort estimate for agile software development. When the estimation of effort is not well defined or is inaccurate, the results can directly reflect the software delivery, causing customer dissatisfaction or decrease the quality of the product, thus leading to the need for new mechanisms that can assist in this process. Given this need, this paper presents a framework, called *GiveMe Effort* to support the effort estimation activities in the maintenance and evolution of software in agile methods. The solution is based on historical data change requests associated with the maintenance and evolution of software.

Categories and Subject Descriptors

D.2.6 **[Programming Environments]**: Integrated Environments; D.2.7 **[Software Engineering]**: Distribution, Maintenance, and Enhancement - *Version control*;

H.4.2 **[Information Systems Applications]**: Types of Systems - decision support; K.6.3 **[Software Management]**: *Software maintenance*

General Terms

Management, Measurement, Experimentation, Verification.

Keywords

Effort Estimation, Software Maintenance, Software Comprehension, Software Visualization.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. *SBSI 2016*, May 17–20, 2016, Florianópolis, Santa Catarina, Brazil. Copyright SBC 2016.

1. INTRODUÇÃO

Planejar atividades de manutenção de software significa definir estratégias, para estimar o esforço de manutenção ao longo do ciclo de desenvolvimento [7]. Nesse contexto, o sucesso de projetos depende da precisão do esforço e do cronograma das estimativas, entre outras coisas.

O surgimento de métodos ágeis no campo de desenvolvimento de software tem apresentado muitas oportunidades e desafios para pesquisadores e profissionais da área. Um dos principais desafios é a estimativa de esforço para o desenvolvimento ágil de software. Embora as abordagens tradicionais de estimativa de esforço sejam usadas em projetos que utilizam métodos ágeis, observa-se que as mesmas têm resultado em estimativas imprecisas [3].

Quando a estimativa de esforço não ocorre, não se tem uma visão geral de todos os projetos em desenvolvimento. Além disso, pode-se dimensionar, equivocadamente, o projeto atribuindo prazos imprecisos para atividades de manutenção de software. Informações importantes podem ser perdidas quando não acontece o planejamento efetivo, e não se tem uma visão de desenvolvimento futura [7]. Adicionalmente, devido à evolução e manutenção constantes em um produto de software, torna-se fundamental a construção de teorias e modelos que permitam a compreensão do passado, presente e futuro do ciclo de manutenção e evolução do software [3].

Informações históricas, armazenadas ao longo do ciclo de vida, podem auxiliar as equipes na criação de mecanismos de dimensionamento de tempo e esforço das atividades a serem desenvolvidas [7]. Essas medidas, se corretamente tratadas, podem permitir que iniciativas sejam tomadas objetivando melhorar a qualidade do software, bem como auxiliar as equipes no dimensionamento de suas atividades. Para um acompanhamento adequado dessas atividades, uma quantidade significativa de dados devem ser coletadas, processadas e armazenadas ao longo do tempo. No entanto, o valor desses dados

depende das possibilidades de extrair e entender as informações a partir deles, de modo a controlar e melhorar o processo de manutenção de software.

Nesse contexto, a manutenção e evolução de software necessitam de ferramentas para apoiar a estimativa de esforço de suas atividades. Para atingir o objetivo de dimensionamento de esforço usando uma base histórica, os dados devem ser apresentados em uma interface que promova iteração com o desenvolvedor. Um Ambiente Interativo baseados em Múltiplas Visões (AIMV) [1] pode oferecer recursos e mecanismos de visualização para apoiar a análise de dados e também a descoberta e o reconhecimento de informação relevante oferecendo suporte às atividades de manutenção e evolução de software.

O objetivo desse trabalho é apresentar um *framework*, denominado *GiveMe Effort*, integrado a um AIMV [1]. Este *framework* auxilia no processo de estimativa de esforço usando dados históricos de requisições de mudanças e elementos visuais, que podem apoiar a estimativa de esforço de atividades de manutenção e evolução de software.

Este artigo está organizado da seguinte forma. A seção 2 descreve os trabalhos relacionados ao tema proposto e faz uma comparação entre os mesmos ressaltando as lacunas que justificam esta proposta de pesquisa. A seção 3 descreve o *framework*, sua arquitetura, sua relação com sistemas de informação e um estudo de caso do mesmo. A seção 4 apresenta as conclusões do trabalho e perspectivas futuras.

2. TRABALHOS RELACIONADOS

Com o intuito de conhecer os trabalhos existentes sobre o planejamento das atividades no contexto da Manutenção e Evolução de Software, foi realizada uma Revisão Sistemática de Literatura (RSL) [9]. Após a definição do protocolo do mapeamento sistemático, as *strings* de busca foram definidas e executadas nas bases (IEEE, ACM, SCOPUS, ScienceDirect, Engineering Village, ISI) e suas publicações, e o retorno das buscas foram analisados. As buscas ocorreram ao longo do mês de dezembro de 2014, e os idiomas considerados foram Inglês e Português. Foram recuperados 692 artigos sendo que destes, 72 foram aceitos considerando-se os critérios definidos no protocolo. Neste artigo, devido à questão de espaço, foram abordados 5 (cinco) trabalhos que possuem maior relevância com o tema apresentado. Artigos duplicados foram eliminados, bem como aqueles que não descreviam as tecnologias investigadas, ou não se relacionavam diretamente com o tema da revisão. Foram realizadas análises adicionais, tais como as variações de publicações ao longo dos anos, os locais de publicações, os principais autores da área, e os dez artigos mais referenciados, bem como as ameaças à validade da pesquisa. O conteúdo detalhado da RSL está disponível em [1].

Ao analisar os resultados obtidos pela RSL, pôde-se identificar uma lacuna referente aos trabalhos que abordam a estimativa de esforço nas atividades de Manutenção e Evolução de Software. A revisão apresentou também alguns trabalhos com uma maior afinidade com a proposta do *GiveMe Effort*, e a escolha dos pré-requisitos se baseia nas condições necessárias que uma tecnologia necessita para resolver o problema apresentado nesse trabalho. Esses requisitos são importantes pelo fato de fornecerem uma motivação ao trabalho, visto que, nenhuma das ferramentas analisadas trata do tema de Estimativa de Esforço nas Atividades de Manutenção e Evolução de Software. Os pré-requisitos são: **(i) Análise de código fonte**: tem como objetivo descrever se a ferramenta analisa o código fonte de aplicações, para gerar a visualização para apoiar a manutenção e evolução do software; **(ii) Análise de repositório de defeitos**: tem como objetivo descrever se a ferramenta oferece suporte à análise de repositório de defeitos; **(iii) Análise de repositórios de dados de processos:** tem como objetivo descrever se a ferramenta possui suporte para a análise de repositórios de dados de processos; **(iv) IDE:** tem como objetivo descrever a IDE em que o *framework* se baseia para seu funcionamento e exibição das visualizações propostas; **(v) Suporte a Paradigmas de LP:** tem como objetivo descrever o paradigma de desenvolvimento de software (Orientado a Objetos, Funcional, Imperativo, entre outros) através do qual o *framework* ou ferramenta é estruturado; **(vi) Suporte a Plataformas:** define o tipo de plataforma de visualização (Web/Desktop) em que a ferramenta ou *framework* foi desenvolvido; **(vii) Análise de Estimativa de Esforço**: objetiva analisar se o *framework* ou ferramenta, possui algum tipo de mecanismo de Estimativa de Esforço nas atividades de Manutenção e Evolução do software.

Os trabalhos selecionados conforme os critérios anteriores são apresentados a seguir.

Carneiro et al. [10] descrevem uma arquitetura cujo resultado é a combinação de um AIMV [1], com elementos de percepção para apoiar a compreensão de software no desenvolvimento distribuído. O objetivo é fornecer informações que permitam aos integrantes das equipes conhecerem, através do ambiente, o que os demais pesquisaram, manipularam ou alteraram em um projeto de software. A partir desse objetivo, é construído um *framework* (*Collaborative SourceMiner*), integrado a IDE de desenvolvimento Eclipse, e focado em código fonte, para apoiar a percepção no domínio da visualização de informações em um ambiente de desenvolvimento colaborativo de software. Nesse contexto, o *GiveMe Effort*, também permite a análise do código fonte, para que o mesmo possa apoiar o processo de planejamento das atividades de manutenção e evolução de software. Porém, a arquitetura apresentada nesse trabalho não apoia a estimativa de esforço em atividades de manutenção e evolução de software.

Storey et al. [8] descrevem uma pesquisa (*Survey*) sobre as ferramentas de visualização existentes e propõem um *framework* conceitual que apoia o desenvolvimento e a criação de novos *frameworks*. Além disso, esse trabalho possibilita uma análise qualitativa da ferramenta, avaliando as características apropriadas da visualização colaborativa de software. Foi possível observar também que o *framework* não apoia o processo de estimativa de esforço nas atividades de manutenção e evolução de software.

Churrasco [5] é uma ferramenta para apoiar a análise da evolução colaborativa de software. Destacam-se como pontos positivos da ferramenta a análise do código fonte do repositório em diversos repositórios. Ainda possui uma interface com recursos, tais como: anotação colaborativa e visualização gráfica de processo e de métricas. Oferece o suporte de visualizações para que os usuários, de forma colaborativa, interajam durante o processo de manutenção e evolução de software. Churrasco possui semelhanças com a solução proposta neste trabalho, como a capacidade de obter automaticamente dados históricos de repositórios de código fonte e capacidade de apoiar equipes distribuídas. Adicionalmente, apoia a colaboração entre membros,

[1] RSL - Disponível em http://givemeinfra.com.br/files/slr01.pdf, Acessado em 01 de fevereiro de 2015.

em atividades de compreensão sobre a evolução do software, entretanto não apoia a estimativa de esforço em atividades de manutenção e evolução de software.

FRASR [13] é um *framework* que permite a análise de processos de desenvolvimento de software armazenados em repositórios de dados. Dentre suas características, destacam-se a capacidade de extrair informações de sistemas de controle de versão e *bug trackers*. Especialmente, é capaz de contabilizar os defeitos encontrados e verificar o estado em que se encontram, como criado, fechado, comentado ou reaberto. No contexto de repositório de defeitos, o *GiveMe Effort* também é capaz de extrair dados sobre defeitos como, por exemplo, seu *status*, observações e, principalmente, a relação entre o defeito informado e a parte do código fonte que resolveu o mesmo, mas diferentemente da ferramenta analisada, apoia a estimativa de esforço no contexto de e manutenção e evolução de software.

BugManagement [14] permite a análise de *bugs* cadastrados em um sistema de *bug tracking* objetivando encontrar similaridades entre eles. Foi desenvolvida para ser utilizada de forma colaborativa, através das quais desenvolvedores são capazes de encontrar similaridades entre *bugs*. A ferramenta atua considerando dados históricos, tanto de defeitos como de alterações em código fonte. Possui ainda a capacidade de apoiar a colaboração entre equipes de manutenção. Similarmente, o *framework GiveMe Effort* possibilita a análise dos defeitos de forma histórica, entretanto, a ferramenta *Bug Management*, assim como as demais apresentadas, não avança para apoiar a estimativa de esforço em atividades de manutenção e evolução de software.

A Tabela 1 sintetiza as soluções previamente apresentadas, e as relaciona com o conjunto de pré-requisitos que foram analisados no inicio dessa seção.

Tabela 1. Tabela comparativa das soluções

Ferramenta/ Framework	***Código Fonte***	***Repositório de Defeitos***	***Repositório de Processos***	***IDE***	***Paradigma***	***Plataforma***	***Estimativa de Esforço***
Collaborative SourceMiner [10]	Sim	Não	Não	Eclipse	OO	Desktop	**Não**
Storey [8]	Não	Não	Não	Conceitual	Conceitual	Ambos	**Não**
Churrasco [5]	Sim	Sim	Não	Browser Web	OO	Web	**Não**
FRASR [13]	Não	Sim	Não	Própria	Ambos	Desktop	**Não**
BugManagement [14]	Sim	Sim	Não	Própria	Ambos	Desktop	**Não**

Conforme a Tabela 1, os *frameworks* e ferramentas apresentados reforçam a contribuição deste trabalho, visto que nenhum deles apoia algum tipo de modelo para a estimativa de esforço em atividades de Manutenção e Evolução de software ao longo do seu ciclo de vida.

A próxima seção apresenta a arquitetura da solução proposta. Nela foi contemplada a estimativa de esforço nas atividades, com o suporte à visualização de software na manutenção corretiva, evolutiva e adaptativa de software.

3. *GIVEME EFFORT*

GiveMe Effort é parte de uma estratégia de estimativa de esforço nas atividades de Manutenção e Evolução de software, e foi projetada para ser integrada à *GiveMe Infra* [1]. Além disso, a proposta da solução disponibiliza visualizações e acompanhamentos das atividades de estimativas de esforço e seu planejamento. A arquitetura em módulos do *framework* possibilita a customização das visualizações e das métricas suportadas pelo *GiveMe Effort*, e a integração com diferentes ferramentas.

Alguns requisitos não funcionais merecem destaque no modelo arquitetural da ferramenta: (i) **portabilidade,** foi considerada para que o sistema fosse implementado de forma a funcionar com diversos sistemas operacionais (Windows, Linux, MacOS), (ii) **reutilização,** possibilitará a utilização dos módulos do framework em outras aplicações da *GiveMe Infra*; (iii) **interoperabilidade,** possibilitar a troca de dados com a infraestrutura *GiveMe Infra*.

Como requisitos funcionais na arquitetura do framework pode-se destacar: (i) prover **múltiplas técnicas** para o calculo da estimativa de esforço, (ii) prover **múltiplas visualizações** para auxiliar na compreensão do problema apresentado, (iii) permitir a **integração** com outros *frameworks* como o *GiveMe Infra* [1], (iv) possibilitar a **modularidade** das técnicas permitindo sua substituição.

3.1 Solução Proposta

O *framework GiveMe Effort* se baseia no conjunto histórico dos dados analisados e disponibilizados pela arquitetura da *GiveMe Infra*[1] para gerar visualizações. Como resultado, apoia a estimativa de esforço nas atividades de manutenção e evolução colaborativas de software.

3.2 Arquitetura do Ambiente

A Figura 1 apresenta uma visão geral da arquitetura da *GiveMe Effort*, integrada à infraestrutura *GiveMe Infra*.

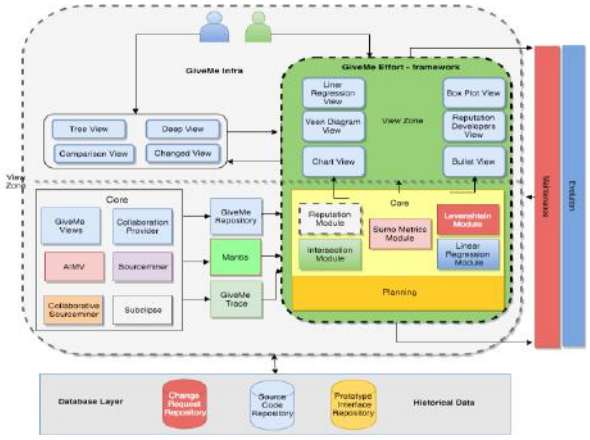

Figura 1. Visão Geral da Arquitetura do *GiveMe Effort*

A *GiveMe Infra* [1] possui mecanismos para extração de dados, bem como auxilia na criação de um padrão dos dados coletados de diversas fontes para que os mesmos possam ser utilizados pelas visualizações disponíveis e que serão utilizados pelo *framework GiveMe Effort*. Possui ainda visualizações que são integradas ao *framework* e que podem auxiliar nos processos de compreensão e de manutenção e evolução do software. Não é objetivo deste trabalho discutir os aspectos relacionados à compreensão de software para apoiar a estimativa de esforço.

GiveMe Effort framework – apoia o planejamento das atividades de manutenção e evolução de software, por meio da sugestão de

estimativas de esforço, levando em consideração os dados extraídos das requisições de mudança ao longo do tempo. Esses dados são analisados quanto à similaridade, que são calculadas utilizando as técnicas apresentadas no *framework*, da requisição de mudança solicitada em relação à base histórica analisada pelo *framework*. É possível, por meio da *View Zone*, área contendo as visualizações apresentadas na arquitetura, obter visualizações das atividades de planejamento de esforço que podem ser úteis no processo de tomada de decisão e planejamento do software. O *core* do *framework* possui vários módulos que auxiliam nos cálculos de estimativa e que somam forças, para que seja possível gerar uma estimativa de esforço nas atividades de manutenção (corretivas, adaptativas e evolutivas) de um software. Será detalhado a seguir o funcionamento de cada um dos componentes apresentados na infraestrutura, bem como sua contribuição para o processo de estimativa de esforço.

Historical Data - Database Layer – é composto por repositórios como *Source Code Repository*, que é um repositório de código fonte como GIT[2] e SVN[3] que possuem dados históricos e evolutivos de software. O repositório *Change Request Repository* possui as requisições de mudanças do software ao longo do tempo, e que são amplamente utilizadas pelo *core* do *framework* para a busca de informações históricas, que servem de comparação e base para o cálculo da estimativa de esforço. Por fim, o *Prototype Interface Repository* possui os protótipos de interface armazenados de forma histórica, e permite a análise de similaridade entre os protótipos existentes na base e o protótipo da funcionalidade a ser implementada. A partir desses dados, é possível descobrir os componentes principais da aplicação, a concentração de alterações no sistema, os códigos fonte relacionados às solicitações de mudança e prover, a partir desses dados, um cenário a respeito da aplicação e seu ciclo de vida, baseado em suas mudanças.

O componente *Sumo Metrics Module* é um módulo de cálculo de similaridade textual entre a requisição de mudança solicitada e o histórico de dados coletado. Para essa busca, e comparação das descrições textuais com similaridades, é utilizada a técnica *Sumo-metric* [28]. O destaque para essa técnica se deu em função da característica de comparação de *strings* que, mesmo que grafadas de uma forma ligeiramente diferente, transmitissem a mesma ideia ou informação. A *Sumo-metric* busca estabelecer a noção de ligações lexicais exclusivas em um par de frases (*strings*) e define um grau de relevância entre as mesmas. Esse grau de relevância entre as requisições de mudança permite selecionar quais requisições da base histórica possuem maior semelhança com a nova requisição a ser implementada.

O componente *Levenshtein Module* é outro módulo de cálculo de similaridade textual entre a requisição solicitada e a base histórica existente. À distância de Levenshtein [6], ou distância de edição entre duas *strings* (duas sequências de caracteres), é dada pelo número mínimo de operações necessárias para transformar uma *string* em outra, com as operações de edição, inserção, eliminação ou substituição de caracteres. Um menor número de trocas entre as *strings* que compõem a descrição da requisição de mudança solicitada para implementação, e as *strings* da base histórica das requisições de mudança, apontam para uma maior semelhança entre elas. As requisições de mudança selecionadas na base histórica por meio da técnica apresentada, fornecem dados como a média do tempo gasto por essas requisições para sua conclusão, fornecendo assim, uma previsão para a nova requisição a ser implementada.

O componente *Reputation Module* é um módulo de cálculo de reputação dos desenvolvedores. Esse módulo leva em consideração para o cálculo do índice de reputação o total de requisições de mudanças realizadas pelo usuário ao longo do tempo. O índice de cada desenvolvedor é normalizado em função dos demais desenvolvedores, utilizando a Norma Euclidiana [4] de um vetor para que os valores estejam em um índice com base 0 a 1. Após isso cada requisição de mudança recebe o índice de reputação do seu desenvolvedor. A partir das técnicas Sumo e Levenshtein, todas as requisições encontradas possuem o índice da reputação do usuário que desenvolveu a requisição e pode-se configurar para filtrar, por exemplo, apenas as requisições de desenvolvedores que possuem um melhor índice geral de reputação.

O componente *Intersection Module* é o módulo que faz a interseção entre as três métricas calculadas (Sumo, Levenshtein e Reputação) para encontrar a interseção entre elas. O objetivo de se usar a interseção de conjuntos é que, com essa técnica, podem-se extrair as requisições de mudanças que atendem, simultaneamente, às três técnicas já descritas e, portanto, encontrar elementos que possuem uma maior afinidade e probabilidade de serem mais parecidos em suas descrições em relação a uma nova requisição de mudança.

O componente *Linear Regression Module* é um módulo que utiliza a análise por Regressão Linear [15] a partir de uma equação para se estimar a condicional (valor esperado) de uma variável *y*, dados os valores de outras variáveis *x*. A partir das requisições de mudança, selecionadas pelas técnicas Sumo e Levenshtein, e seus tempos gastos para o desenvolvimento das requisições encontradas, é calculado o valor esperado para uma nova requisição utilizando a técnica em questão.

A *View Zone*, apresenta 6 (seis) visualizações para acompanhamento das atividades de estimativas de esforço e de planejamento que auxiliam a manutenção e evolução do software, detalhadas a seguir.

A visualização *Pie Chart View* (Fig. 2) permite representar, de forma visual, a quantidade de requisições de mudanças selecionadas pelo *framework* em cada uma das técnicas apresentadas. Essa informação pode ser útil para que se possa determinar rapidamente, e de forma visual, qual técnica está selecionando o maior número de requisições e como está a proporcionalidade de uma determinada técnica em relação às demais.

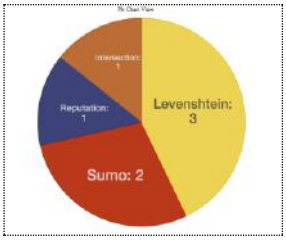

Figura 2. ***Pie Chart View***

A visualização *Bubble View* (Fig. 3) permite representar, de forma visual, o agrupamento das requisições de mudanças em cada uma das técnicas definidas no *framework*. O diagrama usa uma representação circular de cada técnica para que seja possível

[2] Git – Disponível em: https://git-scm.com

[3] SVN – Disponível em: https://subversion.apache.org

representar a hierarquia de valores de cada requisição. O tamanho do círculo e de cada nó revela uma dimensão quantitativa de cada ponto de dados em relação à técnica selecionada. Os círculos envolventes mostram o tamanho cumulativo aproximado de cada subárvore de uma determinada técnica. Essa visualização é útil para definir quais requisições de mudança, dentro de cada técnica, são mais relevantes em questão de esforço gasto para sua execução. Dessa forma, a distribuição dos valores no gráfico pode auxiliar na tomada de decisão ou na investigação de uma determinada requisição, visto que a mesma pode ser rejeitada pelo usuário do *framework*.

Figura 3. *Bubble View*

A representação *Veen Diagram View* (Fig. 4) permite representar, de forma visual, a relação de interseção entre as técnicas. A partir desse diagrama pode-se visualizar em cada uma das técnicas a quantidade de requisições de mudanças associadas a ela, bem como a quantidade de requisições que participam da interseção entre elas.

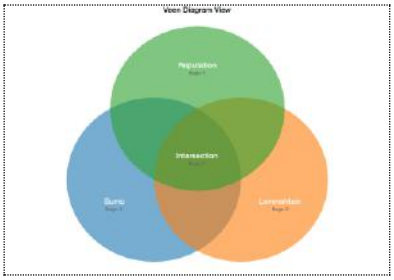

Figura 4. *Veen Diagram View*

A representação *Box Plot View* (Fig. 5) permite representar, de forma visual, o *Box Plot* [11]. Os pontos que estão fora desses limites são considerados *outliers* [11], ou seja, anormalidades do conjunto de dados que merecem uma analise mais detalhada pelo usuário. O *Box Plot View* apresenta cada uma das técnicas utilizadas (Sumo, Levenshtein, Reputação e Interseção) e permite, para o conjunto de dados apresentados, as verificações dos valores máximos, mínimo, terceiro quartil, mediana, primeiro quartil e a média das estimativas de esforço das requisições de mudança que compõem cada uma das técnicas.

O gráfico *Reputation Developers View* (Fig. 6) permite representar, de forma visual, a reputação dos desenvolvedores em função do tempo. Essa informação pode ser útil para a tomada de decisão, à medida que torna possível analisar de forma gráfica como o índice da reputação de cada desenvolvedor está se comportando ao longo do tempo, e como o mesmo se comporta em função dos demais desenvolvedores. Com essas informações, o usuário do *framework* pode excluir ou permitir requisições de mudanças históricas de um determinado desenvolvedor ou aumentar o grau de confiança, para que apenas usuários melhores ranqueados, apareçam na seleção de requisições de mudanças feitas pelo *framewok*.

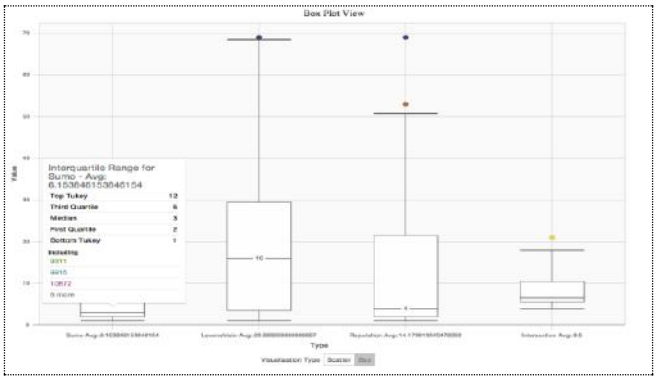

Figura 5. *Box Plot View*

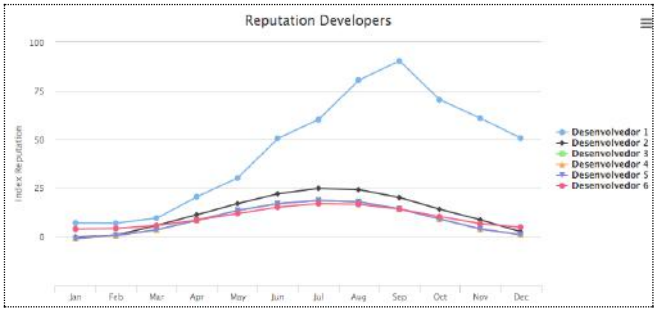

Figura 6. *Reputation Developers View*

A visualização *Linear Regression View* (Fig. 7) permite a representação gráfica dos dados em um sistema cartesiano. No diagrama de dispersão dos dados, apresentado por essa visualização, é possível analisar o comportamento das requisições de mudanças em função do tempo. Esse gráfico também permite a análise da direção e de pontos discrepantes dos dados, que podem ser analisados individualmente e removidos, caso necessário, do cálculo da estimativa pelo usuário do *framework*. Quanto maior a correlação entre o esforço obtido nas requisições de mudança e seu tempo, mais próxima de uma reta a 45° ou 135° será a distribuição. Uma reta em 45° é sugestiva de que o esforço gasto nas requisições de mudanças está aumentando, já uma reta em 135° é um indicativo que o esforço nas requisições de mudanças está diminuindo.

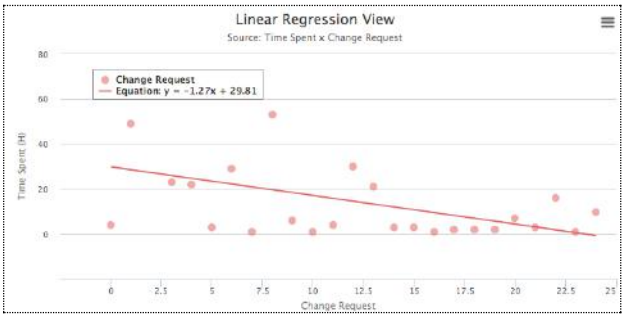

Figura 7. *Linear Regression View*

A última parte do *framework* é o componente *Planning* (Fig. 8), que permite ao usuário selecionar o cálculo de esforço efetuado para uma determinada requisição de mudança, utilizando as técnicas apresentadas na arquitetura, e armazenar esses dados para

que seja possível, de forma incremental, gerar um *Sprint* ou manter um histórico dos cálculos efetuados para diversas requisições de mudanças.

Figura 8. Componente *Planning*

O *GiveMe Effort* (Fig. 9) permite a seleção da requisição de mudança (1) e, ao se clicar no botão (2), inicia-se uma busca nos repositórios de dados pelas requisições que textualmente se assemelham a requisição selecionada para análise. Em seguida, é apresentado na guia (3) de cada uma das técnicas analisadas (Sumo, Levenshtein, Reputation e Linear Regression) bem como o total de requisições de mudanças encontradas (4).

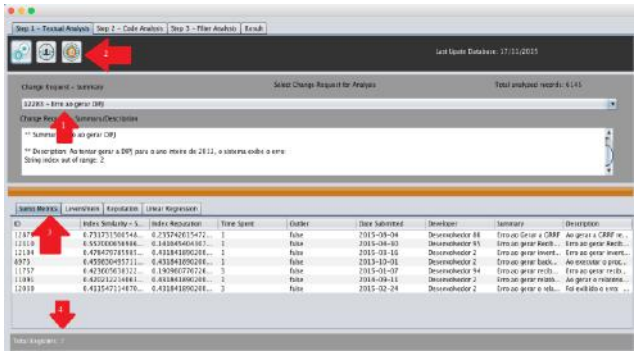

Figura 9. *GiveMe Effort* - Passo 1 – Análise Textual

O *framework* permite na guia *Filter Analysis* (Fig. 10) filtrar requisições de mudança processadas (1), podendo excluir uma requisição, caso o usuário ache necessário. Essa guia apresenta ainda várias visualizações (2), bem como, a interseção entre as técnicas (3). Pode-se ainda analisar as requisições de mudança que foram descobertas por cada uma das técnicas (4), o índice de similaridade das mesmas (5), o índice de reputação aplicado ao desenvolvedor (6) e o tempo gasto (7) para o desenvolvimento da atividade de software. Por fim, pode-se analisar se essa requisição é um *outlier* (8) em relação às demais selecionadas, permitindo assim, a exclusão do mesmo caso o usuário julgue necessário. É possível por meio das opções de configuração do *framework*, remover automaticamente os *outliers* no processamento de cada técnica apresentada, ou fazer uma exclusão manual de cada um na guia apresentada.

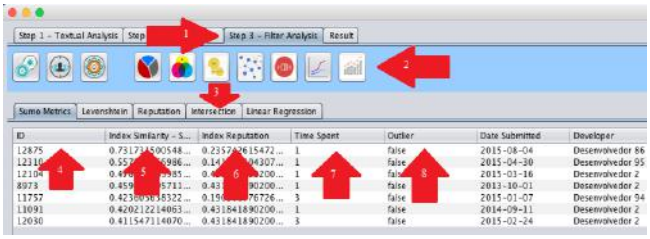

Figura 10. Guia *Filter Analysis*

3.3 Avaliação da Solução

Com o objetivo de avaliar a solução proposta, inicialmente foi executado um estudo exploratório com a duração de 6 (seis) meses na base de dados de uma empresa parceira[4], com o intuito de investigar os problemas reais apresentados com relação a estimativas de esforço, bem como identificar lacunas que, porventura, o presente trabalhado poderia auxiliar nas manutenções evolutivas e corretivas da empresa. A empresa parceira atua no ramo de automação contábil. Atualmente a empresa não possui nenhuma certificação em qualidade de processos, mas a equipe de desenvolvimento trabalha com metodologia ágil SCRUM, para produzir seu ERP (*Enterprise Resource Planning*) Contábil. O software analisado, possui uma base de dados com 10 anos de requisições de mudanças armazenadas e que foram disponibilizadas para análise e estudo na concepção do *framework GiveMe Effort.*

Com o intuito de verificar a viabilidade de utilização do *GiveMe Effort*, uma Prova de Conceito (*Proof of Concept - PoC*) foi executada. A PoC foi criada em um cenário real, de uma empresa parceira de desenvolvimento de software. O objetivo dessa PoC foi definido de acordo com a abordagem *Goal/Question/Metric* (GQM) [12] e foram definidos os objetivos da prova de conceito apresentada e, em seguida, as questões de pesquisa, juntamente com a hipótese. Posteriormente, métricas foram definidas para cada uma das questões de pesquisa.

Goal (Objetivo): Analisar o *framework GiveMe Effort* com a finalidade de verificar a viabilidade para a estimativa de esforço em projetos de software no contexto de desenvolvimento ágil, e se o mesmo contribui para a manutenção e evolução de software.
Questions (Questões):
Q1: *Como GiveMe Effort proporciona à equipe de desenvolvimento mecanismos para que se possa estimar esforço de atividades de manutenção e evolução de software?*
Q2: *Como GiveMe Effort apoia a visualização, manutenção e evolução do software?*
Metrics (Métricas): baseando-se nas questões de pesquisa Q1 e Q2, as seguintes métricas foram definidas, sendo que as métricas 1 e 2 referem-se à questão 1 e a métrica 3 à questão 2. **M1**: percentual de proximidade do cálculo da estimativa de esforço realizada pelo *framework GiveMe Effort*, em relação ao tempo real gasto pelos desenvolvedores durante o ciclo de desenvolvimento analisado; **M2**: percentual de proximidade do cálculo da estimativa de esforço em relação ao previsto pela equipe, sem utilização do *framework GiveMe Effort*; **M3**: quantidade de visualizações utilizadas para auxiliar a equipe de desenvolvimento no cálculo da estimativa de esforço, através do *framework GiveMe Effort.*
O primeiro passo, para conduzir a prova de conceito, foi determinar um *Sprint* [2] para o qual a equipe de desenvolvimento da empresa parceira precisava planejar a estimativa de esforço. Em seguida, a equipe deveria fazer a mesma medição utilizando o *framework GiveMe Effort*. A equipe de desenvolvimento em sua reunião de planejamento definiu de forma manual a estimativa de esforço de cada requisição de mudança, com base na técnica da metodologia ágil SCRUM, *Planning Poker*[5], conforme o

[4] O nome da empresa, desenvolvedores e outras informações confidenciais foram removidos por motivos de sigilo.

[5] Planning Poker - Scrum Guide. Disponível em: https://www.scrum.org/Scrum-Guide. Acessado em: 30 de janeiro de 2015.

procedimento padrão da empresa. Após a coleta de informações, a equipe iniciou o desenvolvimento do *Sprint* e, após sua conclusão, foi coletado o tempo efetivamente gasto em cada requisição de mudança. O resultado das estimativas e dos valores realizados pela equipe encontra-se na Tabela 2. De acordo com a Tabela, **Dev.** refere-se ao desenvolvedor atribuído para a requisição de mudança, **Tarefa** o número da requisição de mudança solicitada para o desenvolvimento; **Tempo Estimado (Horas)** indica o tempo que a equipe sugeriu utilizando a técnica *Planning Poker*. **Tempo Gasto (Horas)** relaciona-se ao tempo real gasto para cada requisição de mudança.

Tabela 2. Requisições de mudanças e estimativas realizadas pela equipe da empresa

Sprint version 3.0.04 Data 11/11/2015							
Dev.	**Tarefa**	**Estimado (H)**	**Gasto (H)**	**Dev.**	**Tarefa**	**Estimado (H)**	**Gasto (H)**
Desenvolvedor 1	13357	2,5	0,3	Desenvolvedor 3	13186	26,3	23,2
	13076	2,5	0,6		13452	2,5	0,3
	13427	2,5	0		13436	2,5	0,2
	13346	2,5	10,4		13411	2,5	3
	13455	2,5	1,4		13410	2,5	1,1
	13446	2,5	0,2		13241	2,5	6,7
	13441	2,5	0		13169	2,5	1,1
	13372	2,5	5,6		13387	2,5	0,1
	13266	2,5	0		13386	2,5	0,6
	13393	2,5	0		13049	0	0,1
	13413	2,5	0		13358	2,5	0
	13365	2,5	0	Desenvolvedor 4	12693	26,3	24,6
	13214	2,5	0		13242	2,5	6,6
	13401	2,5	0		13360	2,5	6,7
Desenvolvedor 2	12897	0	0,4		13348	2,5	6,3
	12590	26,3	24,4		13347	2,5	6,1
	13437	2,5	1,9				
	13250	2,5	2				
	13422	2,5	3,3	**TOTAL**	**40**	**190,2**	**151,1**
	13400	2,5	0,4				
	13398	2,5	0,6				
	13395	2,5	0,3				
	13282	26,3	12,2				
	13381	2,5	0,4				

A partir dessas informações, iniciou-se a calibração do *framework GiveMe Effort*. Foram realizadas várias calibrações na ferramenta, e por questões de espaço, apenas duas calibrações diferentes, foram apresentadas, para que se pudesse analisar e medir os resultados. O *framework* possibilita, por meio de um painel de configuração, ajustar todos os parâmetros de medição encontrados nas tabelas apresentas a seguir. A primeira calibração configurou os seguintes dados no *framework*, conforme demonstrado na Tabela 3.

Tabela 3. Medição 1 – Calibração do *framework*

Configuração		**Medição 1**
% Similaridade SUMO	50%	Tipo Análise: Descrição sumarizada da requisição de mudança
Distância Levenshtein	20 Trocas	Quantidade de anos considerados: **99**
Reputação	40%	Remoção *Outliers*: **NÃO**

A Fig. 11 apresenta um gráfico comparativo entre as técnicas. São apresentados também o valor calculado e o total de estimativa de esforço para cada técnica, conforme legenda, bem como a apresenta a média geral entre as técnicas. Pode-se analisar que a técnica Levenshtein estimou uma quantidade significativa de horas a mais que o real gasto. No entanto, a técnica de Regressão Linear se aproximou bastante do tempo real gasto com uma margem de erro de aproximadamente 9,74 %, bem próximos da estimativa anteriormente feita de forma *ad-hoc*.

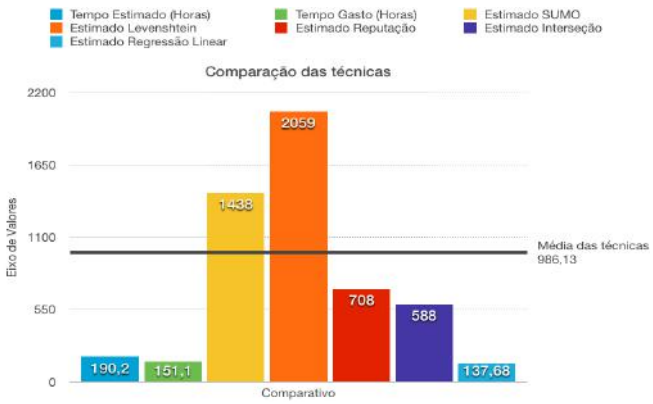

Figura 11. Medição 1 - Gráfico comparativo das técnicas

A segunda calibração configurou os dados no *framework* conforme apresentado na Tabela 4. A diferença dessa calibração para a anterior está no filtro relacionado aos anos considerados para a análise das requisições. Foi incluído apenas 1 (um) ano no histórico de requisições para a análise das técnicas, visto que, na base de dados da empresa parceira, o cálculo do tempo real gasto pelos desenvolvedores foi computado apenas nos dados históricos de aproximadamente de 1(um) ano e meio anterior à análise. Os tempos gastos pelas requisições anteriores a esse período foram estimados e incluídos na base de dados manualmente, levando em consideração o tamanho da requisição de mudança (Pequeno, Médio e Grande), informados historicamente pelos desenvolvedores. A calibração pelo último ano das requisições de mudança permitiu a busca de dados com as estimativas reais feitas pelos desenvolvedores. O percentual de similaridade para a técnica SUMO foi aumentado para 50%, denotando uma maior similaridade textual entre a requisição solicitada e a base histórica. A distância de Levenshtein foi alterada para 20 trocas, permitindo assim, um número maior de requisições de mudanças no espectro de busca da ferramenta.

Tabela 4. Medição 2 – Calibração do *framework*

Configuração		**Medição 2**
% Similaridade SUMO	**50%**	Tipo Analise: Descrição sumarizada da requisição de mudança
Distância Levenshtein	**20 Trocas**	Quantidade de anos considerados: **1**
Reputação	40%	Remoção *Outliers*: **SIM**

O gráfico apresentado na Fig. 12, ressalta os valores totais de cada uma das técnicas geradas pelo *GiveMe Effort* para a quarta medição de dados. Pode-se analisar que, como a remoção dos *outliers* estava ativa, a quantidade de horas apresentada nas técnicas SUMO, Levenshtein e Reputação, continuou abaixo do tempo estimado pelos desenvolvedores e o tempo real gasto. A média geral estimada pelo *framework* foi de 91,38 horas, aproximadamente 65,35%, a menos que o tempo real gasto. No entanto, com a técnica de Regressão Linear se obteve uma margem de erro de aproximadamente 2% do real gasto, e uma margem de 22,78% aproximadamente para o tempo estimado pelos desenvolvedores.

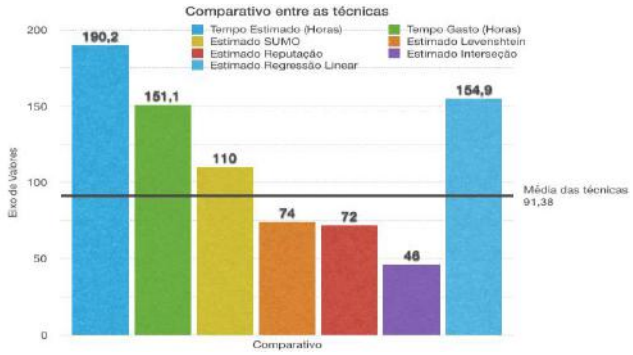

Figura 12. Medição 2 - Gráfico comparativo das técnicas

As análises realizadas tiveram como objetivo responder às duas questões estabelecidas no GQM, que são: **Q1:** Como o *GiveMe Effort* proporciona a equipe de desenvolvimento mecanismos para que se possa calcular a estimativa de esforço das atividades manutenção e evolução de software? Os dados apresentados pela prova de conceito reforçam a afirmação de que é possível que o *framework GiveMe Effort* proporcione mecanismos para calcular a estimativa de esforço, para equipes de desenvolvimento, apoiando as atividades de manutenção de software. **Q2:** Como o *GiveMe Effort* apoia a visualização, manutenção e evolução do softwares? A segunda questão levantada pode ser respondida pelas visualizações que o *framework* oferece, conforme abordado na Seção 3 deste trabalho. Essas visualizações permitem, a cada requisição de mudança calculada, uma gama de informações úteis que podem auxiliar na tomada de decisão por parte do usuário.

Os trabalhos relacionados, apresentados na seção 2, não apresentaram nenhum modelo de apoio à estimativa de esforço em atividade de manutenção e evolução de software ao longo do ciclo de vida de um software, reforçando assim a contribuição apresentada pelo presente trabalho.

4. CONCLUSÃO

Diante da necessidade levantada nesta pesquisa, que demonstrou que estimativas de esforço não bem definidas ou imprecisas, podem prejudicar a entrega do software, causando insatisfação do cliente ou a diminuição da qualidade do produto, o presente trabalho apresentou um *framework* para apoiar a estimativa de esforço no contexto de métodos ágeis.

A partir das avaliações realizadas, foi possível obter indícios de que *framework* proposto pode auxiliar a estimativa de esforço para equipes que utilizam desenvolvimento ágil, através da integração com a infraestura *GiveMe Infra*. A arquitetura do *framework GiveMe Effort* tem por objetivo apoiar equipes de desenvolvimento no planejamento da manutenção e evolução de software. Auxilia no dimensionamento do esforço gasto para cada atividade de software utilizando técnicas de rastreabilidade e similaridade. O tempo total de diversas atividades pode ser somado e adicionado ao planejamento da equipe e disponibilizado de forma histórica para consulta.

Como trabalhos futuros, avaliações experimentais necessitam ser realizadas buscando-se, por exemplo, avaliar os requisitos não funcionais aqui propostos, como portabilidade, reutilização e interoperabilidade. Além disso, outros experimentos em contextos reais de utilização necessitam ser conduzidos no sentido de buscar novos resultados utilizando o *framework.*

5. REFERÊNCIAS

[1] Tavares, J., David, J. M. N., Araújo, M. A.P., Braga, R., Campos, F. C. A. Carneiro, C. GiveMe Views: uma ferramenta de suporte a evolução de software baseada na análise de dados históricos. 2015. *In Simpósio Brasileiro de Sistemas de Informação (SBSI), 55-62.*

[2] Gestal, P. R. E, Barros, R. M. 2014 Proposta de Um Simulador para Auxiliar no Processo de Ensino do Scrum. *In Simpósio Brasileiro de Sistemas de Informação (SBSI), 723-736.*

[3] Paredes, J., Anslow, C., & Maurer, F. 2014. Information Visualization for Agile Software Development. In *Software Visualization (VISSOFT), 2014 Second IEEE Working Conference on.* 157-166.

[4] Franco, N. B. Cálculo numérico, 2006. *Pearson*, 5-14.

[5] D'Ambros M., Lanza, M. 2010. Distributed and Collaborative Software Evolution Analysis with Churrasco. *Experimental Software and Toolkits (EST 3): A special issue of the Workshop on Academic Software Development Tools and Techniques (WASDeTT)*, 276–287.

[6] Levenshtein V. Binary Codes Capable of Correcting Deletions, Insertions, and Reversals, *Soviet Physice-Doklady*, 10:707-710, 1966.

[7] Peixoto, C. E. L, Audy J. L. N., Prikladnicki, R. 2010, The importance of the use of an estimation process. *In: ICSE Workshop on Software Development Governance. ACM*.. 13-17.

[8] Storey, M. D., Čubranić D., German, D, M. 2005. On the use of visualization to support awareness of human activities in software development: a survey and a *framework. Proceedings of the 2005 ACM symposium on Software visualization*, 193-202.

[9] Kitchenham, B.A., Charters, S. 2007. Guidelines for performing systematic literature reviews in software engineering. *Tech. Rep. EBSE-2007-01, Keele University*.

[10] Carneiro, G., Conceição C. F., David, J. M. N. 2012. A Multiple View Environment for Collaborative Software Comprehension. *ICSEA: The Seventh International Conf. on Software Engineering Advances, Portugal*, 15-21.

[11] Massart, D. L., Smeyers-Verbeke, J. C. X., Schlesier, K. 2005. Visual presentation of data by means of box plot.

[12] Basili, V. R. C., Gianluigi, H. R. D. 1994. The Goal Question Metric Approach. *Chapter in Encyclopedia of Software Engineering, Wiley*.

[13] Poncin, W., Serebrenik, A., Van Den Brand, M. Process mining software repositories. 2011. *Software Maintenance and Reengineering (CSMR), 2011 15th European Conference on. IEEE*. 5-14.

[14] 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), 6th International Workshop on*, IEEE, 17-24.

[15] Fedotova, O.; Teixeira, L.; Alvelos, H. 2013. Software Effort Estimation with Multiple Linear Regression: Review and Practical Application. *Journal of Information Science and Engineering*, v. 29, n. 5, 925-945.