UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE
CENTRO DE CIÊNCIAS EXATAS E DA TERRA
DEPARTAMENTO DE INFORMÁTICA E MATEMÁTICA APLICADA
PROGRAMA DE PÓS-GRADUAÇÃO EM SISTEMAS E COMPUTAÇÃO

UMA ABORDAGEM DE APOIO À GERÊNCIA DE PROJETOS DE SOFTWARE PARA ANÁLISE DA CONTRIBUIÇÃO DE DESENVOLVEDORES

Jalerson Raposo Ferreira de Lima

Natal – RN

2014

UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE
CENTRO DE CIÊNCIAS EXATAS E DA TERRA
DEPARTAMENTO DE INFORMÁTICA E MATEMÁTICA APLICADA
PROGRAMA DE PÓS-GRADUAÇÃO EM SISTEMAS E COMPUTAÇÃO

UMA ABORDAGEM DE APOIO À GERÊNCIA DE PROJETOS DE SOFTWARE PARA ANÁLISE DA CONTRIBUIÇÃO DE DESENVOLVEDORES

Jalerson Raposo Ferreira de Lima

Dissertação de mestrado apresentada ao Programa de Pós-Graduação em Sistemas e Computação do Departamento de Informática e Matemática Aplicada da Universidade Federal do Rio Grande do Norte como requisito para a obtenção do grau de Mestre em Sistemas e Computação.

Orientador: Prof. Dr. Uirá Kulesza

Co-orientador: Prof. Dr. Fernando Figueira

Natal – RN

2014

AGRADECIMENTOS

Primeiramente agradeço a Deus, por incontáveis graças que me concede diariamente, mas também por me rodear de pessoas maravilhosas e que agora tenho o prazer de agradecer.

Agradeço profundamente à minha amada esposa, Jeovana Lima, pelo carinho, a ternura, os cuidados, a paciência e o eterno companheirismo, sobretudo nos momentos difíceis.

Agradeço à minha família, em especial aos meus pais, Jezualdo Lima e Maria Rosa, por terem sempre me educado com os melhores valores. Agradeço também à família da minha esposa, em especial aos meus sogros, Jeovani Araújo e Rosa Maria, por sempre me receberem com tanto carinho e afeto.

Também sou muito grato ao meu orientador, prof. Uirá Kulesza, pela paciência e as valiosas orientações dadas durante este percurso. Agradeço também ao meu co-orientador, prof. Fernando Figueira, pelo auxílio e orientações dadas para a conclusão deste trabalho.

Agradeço também aos colegas da Superintendência de Informática da UFRN (SINFO), em especial a aqueles que participaram do estudo de avaliação conduzido nesta dissertação.

Gostaria de agradecer também aos membros do grupo de pesquisa no qual pertenço, sob a liderança do prof. Uirá Kulesza, que contribuíram para a conclusão desta dissertação de forma direta ou indireta.

RESUMO

A produtividade é um importante atributo para equipes de desenvolvimento e empresas de software manterem a competitividade no mercado. Contudo, a produtividade de uma equipe depende diretamente da contribuição individual de seus desenvolvedores. Contudo, não há consenso, na literatura, sobre como medir essa contribuição. Avanços nessa área incluem trabalhos que propõem utilizar métricas baseadas em mineração de repositórios como indicadores de contribuição. No entanto, esses trabalhos ainda carecem de estudos para validar a utilidade e aplicabilidade de tais métricas no contexto de gerência de software. Este trabalho de dissertação apresenta: (i) uma abordagem de apoio à gerencia de projetos para acompanhamento da contribuição de desenvolvedores; e (ii) a condução de um estudo de avaliação da abordagem proposta no contexto de uma instituição de desenvolvimento de software realizada em colaboração com gerentes de projetos reais. Os resultados do estudo apontam que as métricas que compõem a abordagem são utéis para análise da contribuição individual de desenvolvedores, mas não devem ser usadas de forma isolada, devendo ser também considerados atributos subjetivos relativos a outras atividades não relacionadas a programação.

Palavras-chave: Gerência de projetos, produtividade de software, métricas de contribuição, mineração de repositórios de software.

ABSTRACT

Productivity is an important aspect for development teams and software companies to maintain their competitivity in the market. However, the productivity of a software team depends directly on the individual contribution of its developers. Unfortunately, there is no consensus in the literature on how to measure that contribution. Advances in this area include studies that propose using metrics based on mining software repositories as contribution indicators. However, these studies still lack the validation of the applicability and usefulness of such metrics in the context of project management. This dissertation proposes: (i) an approach that provides support to the analysis of the developers' contributions; and (ii) an evaluation study of the proposed approach in the context of a software development organization. The results of this evaluation study show that the proposed metrics should not be used in isolation, but associated with other attributes that consider more subjective attributes, and the metrics contribute in different levels to the evaluation of the developers' contributions.

Keywords: Software project management, software productivity, contribution metrics, software mining repositories.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

A produtividade é um atributo considerado dos mais importantes para a manutenção da competitividade e da rentabilidade de qualquer empresa no mercado. Contudo, no contexto de desenvolvimento de software, ainda não há consenso entre os pesquisadores sobre o que é produtividade e sobre como medí-la, dificultando, portanto, a sua avaliação de forma apropriada (COSTA, 2012; DE O MELO et al., 2013). Essa falta de consenso é provocada, principalmente, porque seu significado pode variar de acordo com o contexto, e também devido aos vários conceitos envolvidos na sua definição, como eficácia, eficiência, desempenho, qualidade, entre outros (DE O MELO et al., 2013).

Uma medida de produtividade apresentada por alguns autores é a razão entre aquilo que foi produzido (saída) e os insumos (entrada) necessários para produzi-los (KITCHENHAM; MENDES, 2004). Entretanto, não há consenso na literatura sobre como medir aquilo que foi produzido (DE AQUINO; DE LEMOS MEIRA, 2009; PETERSEN, 2011; TRENDOWICZ; MÜNCH, 2009). Estudos recentes (MACCORMACK et al., 2003) (RAMASUBBU et al., 2011), usaram linhas de código-fonte (LOC) para quantificar o que foi produzido. Em contrapartida, vários trabalhos (JONES, 1978) (BOEHM, 1987) (DUNCAN, 1988) apresentam diversos problemas no uso do LOC para tal fim. Por exemplo, não há consenso sobre o que considerar na contagem de linhas de código. Algumas técnicas consideram cada instrução de código, outros desconsideram comentários e declarações de dados (JONES, 1978). Outro problema é que LOC não necessariamente reflete o valor agregado do que está sendo produzido e nem a qualidade do software (BOEHM, 1987).

(PREMRAJ et al., 2005) usaram uma variação dos pontos de função, denominada Pontos de Experiência 2.0, para medir o que foi produzido (saída). (ANSELMO; LEDGARD, 2003), por sua vez, criticam o uso de pontos de função e pontos de caso de uso, alegando que essas medidas não consideram a complexidade funcional de forma apropriada. Nesse mesmo estudo, (ANSELMO; LEDGARD, 2003) ainda sugerem que a qualidade do que foi produzido deva ser considerado para aferir a produtividade. Os trabalhos citados refletem a diversidade de medições que podem ser usadas para aferir aquilo que foi produzido. Exemplo disso é que, (DE AQUINO; DE LEMOS MEIRA, 2009) em seu estudo, analisou as diferentes métricas para medição da saída (*output*) do processo de desenvolvimento, dividindo tais abordagens em quatro categorias: *physical size*, *design size*, *functional size* e *value size*.

Na realidade existe uma infinidade de medições que poderiam ser utilizadas para tal fim, como aquelas que foram mencionadas anteriormente, mas também medições que avaliem aspectos sociais e de integração entre os indivíduos, pois todos esses aspectos podem influenciar positivamente ou negativamente no processo de desenvolvimento de software, e portanto, devem ser considerados na medição da produtividade. Estudos recentes (GOUSIOS; KALLIAMVAKOU; SPINELLIS, 2008) (COSTA, 2012) (RASTOGI; GUPTA; SUREKA, 2013) (NAGWANI; VERMA, 2012) consideram como medição de produtividade diferentes tipos de contribuições de desenvolvedores. (GOUSIOS; KALLIAMVAKOU; SPINELLIS, 2008) argumentam que a contribuição deve refletir a combinação de todas as ações que um desenvolvedor pode desempenhar no processo de desenvolvimento de software, permitindo quantificar a produção de cada um individualmente. A avaliação da contribuição permite a identificação de gargalos na implementação, além de contribuir no planejamento do projeto e futuras estimativas.

Para (SOMMERVILLE et al., 2007), medidas de produtividade podem dar suporte aos gerentes no processo de tomada de decisão, pois tais medidas ajudam a definir custos, prazos, justificar decisões de investimentos como capacitação de pessoal, avaliar melhorias no processo e identificar quais tecnologias foram mais eficazes. Vale salientar, contudo, que apesar da presente dissertação ter foco em medições individuais de contribuição em desenvolvimento de software, existem outros tipos de medições que avaliam a produtividade sob a perspectiva de equipes de software, e não individualmente.

1.1 Problema de Pesquisa

Os trabalhos atuais refletem a importância da medição da produtividade em projetos de desenvolvimento de software, sendo este um atributo fundamental para o sucesso de qualquer projeto. Contudo, poucos estudos se propõem a fornecer métricas aos gerentes de software, que permitam acompanhar o desenvolvimento de sua equipe, bem como usá-las como indicadores de suporte à tomada de decisão. Nenhum estudo atual se propõe a avaliar a aplicabilidade e utilidade de tais métricas num contexto real de desenvolvimento junto aos gerentes de software.

Estudos recentes (COSTA, 2012; GOUSIOS; KALLIAMVAKOU; SPINELLIS, 2008; NAGWANI; VERMA, 2012; RASTOGI; GUPTA; SUREKA, 2013) mostram que não há consenso sobre como medir a contribuição individual dos desenvolvedores, muito menos sobre a sua produtividade. Isso se deve ao fato de que as atribuições de um desenvolvedor,

atualmente, vão muito além do que a simples produção de código-fonte. Participação em reuniões, levantamento e análise de requisitos, produção de documentos e relatórios, comunicação através de e-mails, fóruns, Wiki, gerenciadores de tarefas, são apenas alguns exemplos das diversas atividades que os desenvolvedores podem desempenhar.

Dessa forma, a contribuição individual dos desenvolvedores de software deve ser medida através de vários aspectos de participação, e não apenas o LOC produzido ou a correção de *bugs* (COSTA, 2012; GOUSIOS; KALLIAMVAKOU; SPINELLIS, 2008). Além disso, a contribuição individual deve ser quantificada levando em consideração atributos de qualidade do que foi produzido, bem como a sua complexidade (ANSELMO; LEDGARD, 2003).

Outro problema é a carência de estudos que avaliem, dentro de um contexto industrial de produção de software, a utilidade e aplicabilidade de métricas de contribuição individual como indicadores de desempenho para dar suporte à tomada de decisão por parte de gerentes de software, além da carência de estudos de medição da contribuição que lidem com a definição de valores de referência para suas métricas considerando a realidade dos projetos e equipes de desenvolvimento de uma dada empresa (ALVES; YPMA; VISSER, 2010).

1.2 Limitações dos Trabalhos Existentes

No estudo apresentado por (GOUSIOS; KALLIAMVAKOU; SPINELLIS, 2008), propõe-se utilizar informações coletadas através de mineração de repositórios para levantar dados relevantes para a definição da contribuição do desenvolvedor. A mineração pode ser realizada em diversas fontes, tais como repositórios de código, listas de e-mails, fóruns, repositórios de *bugs*, Wiki e IRC. As contribuições dos desenvolvedores incluem linhas de código, *commits*, reportar um *bug*, fechar um *bug*, etc. É interessante notar que, nesse trabalho, as ações do desenvolvedor, que são mineradas, podem ser contabilizadas de forma negativa ou positiva no cálculo da contribuição final do desenvolvedor.

O cálculo da contribuição do desenvolvedor envolve somar valores de diferentes formas de contribuição. Mas a soma de métricas heterogêneas pode não ser adequada, uma vez que elas quantificam diferentes aspectos de contribuição. O trabalho de tais autores também não demonstra como tais métricas podem ser utilizadas pelos gerentes de software para dar suporte à tomada de decisão, pois a técnica não oferece valores de referência para a adequada interpretação de suas métricas. O estudo de avaliação do modelo proposto também

não analisa a utilidade e aplicabilidade de tais métricas, no contexto de gerência de projetos de software, como forma de acompanhamento da contribuição dos desenvolvedores em equipes de software reais.

(COSTA et al., 2014) conduziram dois estudos experimentais para avaliação da contribuição individual de desenvolvedores através de técnicas de mineração de repositórios e mineração de processos. Um dos estudos foi conduzido no contexto de uma empresa privada de desenvolvimento de software, e o outro foi realizado no contexto de um projeto *open-source*. Para a coleta de dados, são utilizados repositórios de código e de *bugs*. A contribuição do desenvolvedor foi avaliada através de três perspectivas: *commits* que introduziram *bugs*, tamanho dos *commits* e resolução de *bugs* prioritários. Além disso, o trabalho também propôs o projeto e implementação de uma infraestrutura extensível para realização de mineração em repositórios, denominada Backhoe[1]. Tal infraestrutura é também utilizada no contexto desta dissertação de mestrado.

Contudo, os estudos conduzidos não avaliaram a aplicabilidade e utilidade de tais métricas como indicadores de contribuição de desenvolvedores a partir do *feedback* de gerentes de projeto de software. (COSTA et al., 2014) também não trataram sobre como interpretar os valores das métricas através de valores de referência. A proposta também não usa nenhuma métrica que avalia a complexidade da contribuição do desenvolvedor.

1.3 Trabalho Proposto

Este trabalho de mestrado propõe uma abordagem de apoio à gerência de projetos de software para análise da contribuição individual dos desenvolvedores, a qual é composta por: (i) um conjunto de métricas que são produzidas através da mineração de repositórios; (ii) uma técnica de identificação e descarte de *outliers*; e (iii) uma técnica para definição de valores de referência.

Para oferecer meios adequados para a tomada de decisão para os gerentes de software, este estudo também objetiva explorar e adotar uma técnica de derivação de valores de

[1] https://code.google.com/p/backhoe-miner/

referência para as métricas que compõem a abordagem. Tais valores são importantes para a adequada interpretação dos dados levantados pelas métricas. Finalmente, este trabalho também propõe a realização de um estudo de avaliação da utilidade e aplicabilidade da abordagem, em um contexto industrial e com a participação de gerentes de software, para avaliá-la como meio de acompanhamento da contribuição individual de desenvolvedores.

Vale ressaltar que, dada a heterogeneidade das possíveis atividades realizadas pelos desenvolvedores, muitas delas difíceis de serem quantificadas, as métricas da abordagem não têm como objetivo substituir a avaliação subjetiva realizada pelos gerentes de software, mas complementar o suporte a essa avaliação subjetiva.

1.4 Objetivos Geral e Específicos

Esta dissertação de mestrado tem como objetivo principal propor e avaliar uma abordagem, voltada para gerentes de projetos de software, para acompanhamento da contribuição individual de seus desenvolvedores. Além disso, esta dissertação avalia, através de um estudo num contexto industrial de produção de software, a aplicabilidade e utilidade dessa abordagem no contexto de gerência de software.

Os objetivos específicos desse trabalho são:

(i) Explorar, na literatura, métricas para análise da contribuição individual de desenvolvedores, que possam compor a abordagem proposta na dissertação;

(ii) Desenvolver métricas para avaliação da complexidade ciclomática introduzida pelo desenvolvedor;

(iii) Investigar técnicas de derivação de valores de referência para métricas de software com o objetivo de adotar uma técnica para compor a abordagem proposta;

(iv) Aplicar a abordagem dentro de um contexto de um estudo de avaliação de projetos de software reais e com a participação de seus respectivos gerentes, para analisá-la como ferramenta de apoio à análise da contribuição individual dos desenvolvedores;

(v) Compreender e analisar, através do estudo de avaliação, as limitações e benefícios da abordagem para acompanhamento da contribuição individual dos desenvolvedores, através dos *feedbacks* fornecidos pelos gerentes de projeto de software;

(vi) Compreender, através do *feedback* dos gerentes de projeto de software, quais qualidades eles julgam ser importantes para um bom desenvolvedor, e como as métricas podem auxiliá-lo a avaliar essas qualidades.

1.5 Organização do Trabalho

Além do capítulo de introdução, esta dissertação de mestrado está organizada em outros cinco capítulos, que foram constituídos da forma apresentada a seguir.

O Capítulo 2 apresenta a fundamentação teórica deste estudo, abordando os conhecimentos fundamentais para a compreensão dos assuntos aqui discutidos, tais como mineração de repositórios, técnicas para definição de valores de referência e identificação de *outliers* em métricas de software.

O Capítulo 3 apresenta a abordagem de acompanhamento da contribuição individual dos desenvolvedores, detalhando elementos que fazem parte da mesma: (i) as métricas pesquisadas na literatura e escolhidas para compor a abordagem; (ii) a nova métrica, proposta por este trabalho, que mede a complexidade ciclomática média por método da contribuição do desenvolvedor; (iii) a técnica específica da abordagem para definição dos valores de referência das métricas; (iv) a técnica específica da abordagem para identificação e descarte de *outliers*.

O Capítulo 4 discorre sobre o estudo de avaliação da abordagem num contexto de gerência de software industrial, para analisar a aplicabilidade e utilidade dela como meio de acompanhamento e avaliação da contribuição individual dos desenvolvedores.

O Capítulo 5 apresenta os estudos que têm relação com o tema abordado por esta pesquisa e serviram de base para formatar a abordagem de acompanhamento da contribuição individual dos desenvolvedores.

O Capítulo 6 apresenta as conclusões do estudo, as considerações finais, as contribuições da presente dissertação, bem como as perspectivas de trabalhos futuros deixados por esse estudo para novas pesquisas.

1.6 Sumário

Este capítulo apresentou uma breve contextualização do trabalho, do problema de pesquisa abordado, as limitações dos trabalhos existentes, a proposta de pesquisa desta

dissertação, os objetivos gerais e específicos e, finalmente, como estão organizados os próximos capítulos deste trabalho.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo visa apresentar os conceitos teóricos que fundamentam a abordagem proposta nesta dissertação, abordando temas relativos à (i) mineração de repositórios de software, (ii) métricas de software, (iii) o Backhoe, que é a infraestrutura utilizada por este estudo para produzir as métricas da abordagem, (iv) técnicas de definição de valores de referência e (v) técnicas de identificação de *outliers*.

2.1 Mineração de Repositórios de Software

Segundo (HASSAN, 2006), repositórios de software contêm informações valiosas sobre a evolução de projetos de software. As informações armazenadas nesses repositórios podem ser utilizadas por pesquisadores para estudar os processos de desenvolvimento sem interferir neles. Alguns tipos de repositórios específicos têm recebido mais atenção dos pesquisadores, tais como (i) sistemas de controle de versão, que armazenam todas as mudanças realizadas em artefatos de software do projeto; (ii) *bug tracking systems*, que guardam informações sobre *bugs* e as ações necessárias para corrigí-los; (iii) mecanismos de comunicação, como e-mail, *chat*, etc., que armazenam a troca de mensagens entre os desenvolvedores.

Esses repositórios têm sido utilizados pelos pesquisadores da área de MSR (*Mining Software Repositories*) com dois propósitos principais: (i) elaborar técnicas para automatizar e melhorar a extração de dados dos repositórios; (ii) descobrir e validar novas técnicas e abordagens para minerar informações importantes de repositórios (HASSAN, 2008). As minerações de repositórios coletam, analisam e cruzam informações de diferentes repositórios com o propósito de descobrir informações sobre o sistema de software e seus projetos. (HASSAN, 2008) destaca alguns tipos de repositórios usados com tais fins:

(i) ***Historical repositories***: como sistemas de controle de versão, repositórios de *bugs*, que guardam informações sobre a evolução e o progresso do projeto;

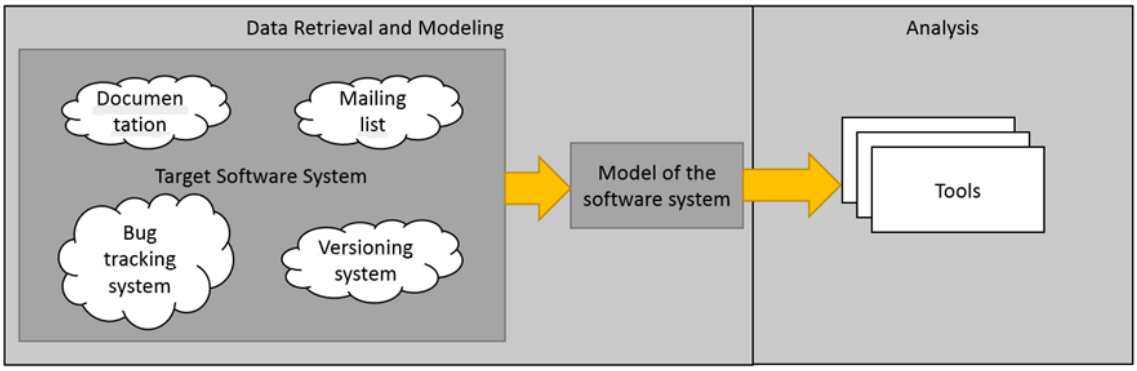
(ii) ***Run-time repositories***: como, por exemplo, *deployment logs*, que armazenam dados sobre a execução e o uso da aplicação;

(iii) ***Code repositories***: como o Sourceforge.net[2] e Google Code[3], que contêm o código-fonte de várias aplicações.

(D'AMBROS et al., 2008) destacam alguns tópicos de pesquisas recentes na área de mineração de repositórios de software, tais como: (i) análise de impacto de mudanças e propagação; (ii) previsão de falhas e defeitos; (iii) análise de tendências e *hotspots*; (iv) análise do esforço de desenvolvimento e da rede de relacionamentos. O tema do presente trabalho de mestrado se encaixa no item (iv), pois versa sobre a contribuição individual de desenvolvimento, que reflete diretamente sobre o esforço despendido pelos desenvolvedores.

2.1.1 Abordagem tradicional

Figura 1 – Abordagem tradicional. Adaptada de (D'AMBROS et al., 2008)



A Figura 1, adaptada de (D'AMBROS et al., 2008), ilustra a abordagem tradicional de mineração de repositórios, que consiste em três passos:

(i) **Modelagem de dados**: o modelo de dados representa as informações que serão colhidas durante a mineração, que podem ser: a última versão de um artefato de código, o histórico de revisões, os *commits* de um desenvolvedor, *bug reports*, etc.;

(ii) **Coleta e processamento de dados**: uma vez definido o modelo de dados, o segundo passo consiste em criar uma instância concreta desse modelo através

[2] http://www.sourceforge.net
[3] https://code.google.com/

de consultas aos repositórios definidos no modelo. Os dados podem ser processados para posterior análise;

(iii) **Análise de dados**: uma vez construída a instância concreta do modelo, os dados processados contidos nela podem ser analisados para levantamento, permitindo desvendar informações antes desconhecidas.

É importante salientar que existem áreas correlatas a esta, como a *Software Analytics* (MENZIES; ZIMMERMANN, 2013), que também visam analisar grande quantidade de dados, principalmente através de mineração de repositórios, com o objetivo de obter informação útil que pode contribuir na melhoria do processo de desenvolvimento de software.

2.2 Métricas de Software

Segundo (FENTON; PFLEEGER, 1998), as métricas de software são importantes mecanismos para aferir as propriedades de um projeto de software. Eles acrescentam que as métricas podem auxiliar os gerentes a responder diversos tipos de perguntas, tais como:

(i) **Quanto custa cada processo?** É possível medir o tempo e o esforço gasto no processo, permitindo que seja calculado o custo de cada um e facilitando realizar previsões futuras;

(ii) **Quão produtiva é a equipe?** É possível medir o tempo despendido pela equipe em cada uma das etapas do processo. Associando isso ao tamanho do que foi produzido, é possível determinar a produtividade da equipe;

(iii) **Quão bom é o código que está sendo desenvolvido?** Através do registro de falhas, defeitos e mudanças, é possível medir a qualidade do software;

(iv) **O usuário ficará satisfeito com o produto?** Medições de usabilidade, confiabilidade, tempo de resposta e outras características podem ajudar a predizer se os usuários ficarão satisfeitos com o produto;

(v) **Como podemos melhorar?** Através das métricas é possível comparar os custos e os benefícios de cada prática para tentar determinar se vale a pena aplicá-la. Também é possível tentar variações de uma prática e determinar se essa variação foi positiva, negativa ou irrelevante, permitindo melhorar as práticas desempenhadas durante o projeto.

A abordagem avaliada por este trabalho de mestrado tenta auxiliar os gerentes de software a responderem as perguntas (ii) e (iii). (FENTON; PFLEEGER, 1998) explicam que

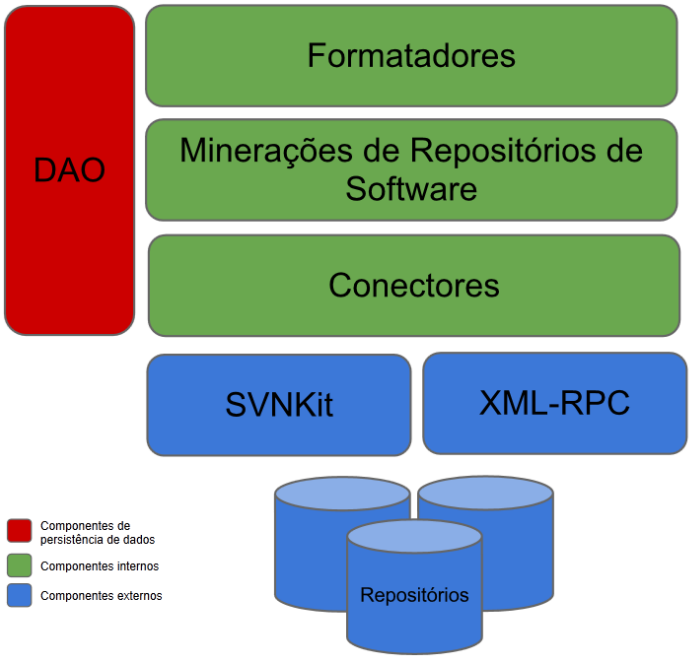
as medições podem ser classificadas entre direta ou indireta. A medição direta de um atributo de uma entidade não envolve outro atributo ou outra entidade. Por exemplo: o tamanho de um objeto pode ser medido sem a necessidade de nenhum outro atributo. A Engenharia de Software normalmente lida com medições diretas, tais como o tamanho do código-fonte, tempo de duração dos testes, quantidade de *bugs* encontrados, tempo consumido para conclusão de uma tarefa, etc.

A medição indireta, também conhecida como medição derivada, por sua vez, é a composição de medições diretas. Um exemplo é a densidade de problemas em um módulo, que é calculada através da razão entre a quantidade de problemas encontrados no módulo, dividido pelo tamanho do módulo, que pode ser medido em linhas de código, por exemplo (FENTON; PFLEEGER, 1998).

2.3 Backhoe

(COSTA, 2012) desenvolveu uma infraestrutura extensível para realização de minerações de repositórios diversos, denominada Backhoe. Este projeto facilita a implementação de novas minerações, bem como o reúso daquelas já existentes, e é importante para a abordagem proposta por esta dissertação, uma vez que as métricas que compõem a abordagem se baseiam naquelas apresentadas por (COSTA et al., 2014), e também foi usada para a implementação das novas métricas discutidas na Seção 3.3.

Figura 2 – Arquitetura do Backhoe (COSTA, 2012)



Conforme ilustrado na Figura 2, a arquitetura do Backhoe está dividida em componentes de persistência de dados (em vermelho), componentes internos (em verde) e componentes externos (em azul). Os componentes de persistência de dados consistem naqueles responsáveis pelo armazenamento permanente dos dados colhidos durante as minerações, bem como informações descobertas após o processamento. Os componentes externos ao projeto são as bibliotecas auxiliares utilizadas pelos outros componentes. Os repositórios alvos das minerações também estão representados como componentes externos.

Os principais componentes do Backhoe são os internos, marcados em verde na Figura 2. São eles:

(i) **Conectores**: são componentes responsáveis por realizar a conexão com o repositório externo. Essa conexão é utilizada pelas minerações para realizar as consultas necessárias aos repositórios externos, que podem ser bancos de dados, sistemas de controle de versão, repositórios de bugs, etc. Comumente os conectores utilizam bibliotecas externas para realização da conexão. Exemplos dessas bibliotecas são o SVNKit[4], para conexão com o Subversion[5], e o XMLRPC, para extrair dados do Bugzilla[6];

(ii) **Minerações**: utilizam os conectores para acessar os repositórios externos e realizar consultas sobre eles. São os componentes responsáveis por implementar as estratégias de mineração e processamento dos dados colhidos. As minerações também utilizam os componentes de persistência para armazenar as informações processadas;

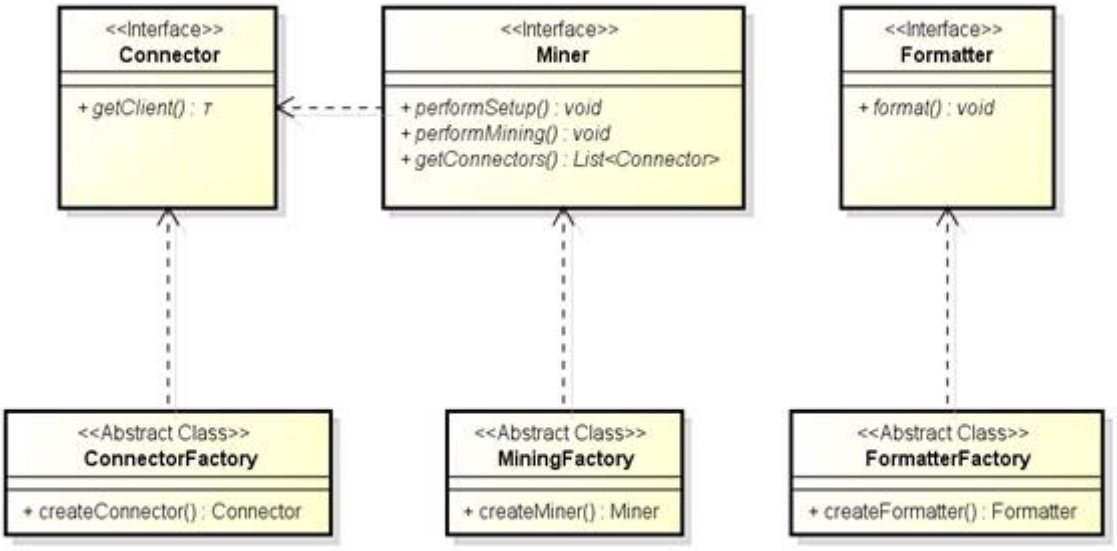
(iii) **Formatadores**: utilizam os componentes de persistência para consultar as informações processadas pelos mineradores, e convertem tais informações para um formato específico, como por exemplo XML, CSV e MXML, para que possam ser usadas em ferramentas de análise de dados.

A arquitetura do Backhoe foi projetada e implementada para permitir acoplar novos conectores, minerações e formatadores. Para isso, basta implementar uma nova classe

[4] http://svnkit.com/
[5] http://subversion.apache.org/
[6] http://www.bugzilla.org/

utilizando a interface apropriada: Connector, Miner ou Formatter. A Figura 3 ilustra o diagrama de classes do Backhoe com as referidas interfaces.

Figura 3 – Diagrama de classes do Backhoe (COSTA, 2012)



2.3.1 Minerações e Métricas

(COSTA, 2012) utilizou técnicas de mineração de repositórios e mineração de processos para avaliação da contribuição de desenvolvedores de software. A abordagem proposta por esse estudo permite avaliar o desenvolvedor através de três perspectivas de contribuição: (i) *commits* defeituosos, (ii) tamanho dos commits e (iii) *bugs* prioritários. Ele conduziu dois estudos empíricos de avaliação: o primeiro deles dentro de um contexto de projeto *open-source*, o ArgoUML, e o outro conduzido dentro de uma empresa privada. A estratégia de implementação das minerações depende muito do contexto em que elas vão auxiliar, pois elas dependem da dinâmica de trabalho. Dessa forma, a estratégia de implementação de uma métrica pode variar quando ela for aplicada em diferentes contextos, como o projeto *open-source* e a empresa privada.

Um exemplo dessa diferença é a estratégia para identificar se um *commit* corrigiu um erro. No contexto do projeto do ArgoUML, um *commit* que corrigiu um erro é identificado verificando se ele possui a palavra "*fix*" (que inclui suas variações, como "*fixed*") nos comentários. No contexto da empresa privada, é verificado o Id da requisição presente no comentário do *commit* (todos os *commits* possuem esse Id no comentário), e em seguida,

consulta-se o gerenciador de tarefas para checar se a tarefa referente àquele *commit* demanda a correção de um erro.

Portanto as métricas e minerações do Backhoe apresentadas a seguir foram aquelas aplicadas na empresa privada, pois esse contexto se assemelha mais ao ambiente onde avaliação da abordagem discutida nesta dissertação foi realizada, conforme discutido no Capítulo 4.

2.3.1.1 *Commits* Defeituosos

Segundo (COSTA, 2012), um *commit* defeituoso é aquele que demanda, posteriormente, um outro *commit* para a realização de correções de problemas introduzidos pelo primeiro *commit*. Para identificar um *commit* defeituoso, o estudo segue os passos a seguir.

1) **Identificar os *commits* que corrigiram defeitos**: capturar, nos comentários do *commit*, o Id da tarefa no gerenciador de tarefas, e verificar se ela demanda a correção de um erro. Em caso positivo, o *commit* em questão foi necessário para a correção de um defeito;
2) **Identificar o local das modificações feitas por eles**: realiza um *diff* para determinar quais linhas de código precisaram ser alteradas pelos *commits* que corrigiram os defeitos. O *diff* é executado usando a revisão atual do arquivo alterado no *commit* e a revisão anterior dele;
3) **Identificar os autores dos *commits* que introduziram as linhas que tiveram de ser corrigidas**: uma vez identificadas as linhas alteradas para a correção do *bug*, a mineração executa um *blame* para identificar quem foram os últimos desenvolvedores a manipular essas linhas. Esses desenvolvedores foram os responsáveis por introduzir os *bugs*.

2.3.1.2 Tamanho dos *Commits*

Essa perspectiva de contribuição é avaliada sob a ótica de sete métricas, ilustradas na Tabela 1.

Tabela 1 – Métricas usadas para mensurar o tamanho dos *commits* (COSTA, 2012)

Nome	Descrição
Linhas adicionadas	Linhas de código adicionadas pelo desenvolvedor. Aquelas que possuem o símbolo "+" na descrição das alterações do *commit*.
Linhas removidas	Linhas de código removidas pelo desenvolvedor. Aquelas que possuem o símbolo "-" na descrição das alterações do *commit*.
Classes adicionadas	Classes adicionadas pelo desenvolvedor.
Classes removidas	Classes removidas pelo desenvolvedor.
Classes modificadas	Classes modificadas pelo desenvolvedor.
Métodos adicionados	Métodos adicionados pelo desenvolvedor.
Métodos modificados	Métodos modificados pelo desenvolvedor.

A mineração, inicialmente, busca pelos commits realizados e verifica quais classes foram adicionadas, modificadas e removidas. Em seguida, realiza um *diff* para determinar quais linhas foram adicionadas e removidas em relação a revisão anterior. Com essas informações, a mineração é capaz de determinar os métodos adicionados e modificados pelo *commit*.

2.3.1.3 *Bugs* Prioritários

A última perspectiva de contribuição avaliada pelo Backhoe é uma métrica que quantifica o número de *bugs* prioritários, ou seja, aqueles que têm maior urgência na sua correção, realizados por cada desenvolvedor. Para contabilizar essa métrica, a mineração realiza uma consulta ao gerenciador de tarefas do projeto, em busca de tarefas de correção de erros. Uma vez localizadas, verifica-se, em cada tarefa, quem foi o desenvolvedor que marcou a tarefa como concluída. Por fim, tem-se uma contagem de quantas tarefas de correção de *bugs* foram concluídas por cada desenvolvedor.

2.4 Identificação de *Outliers*

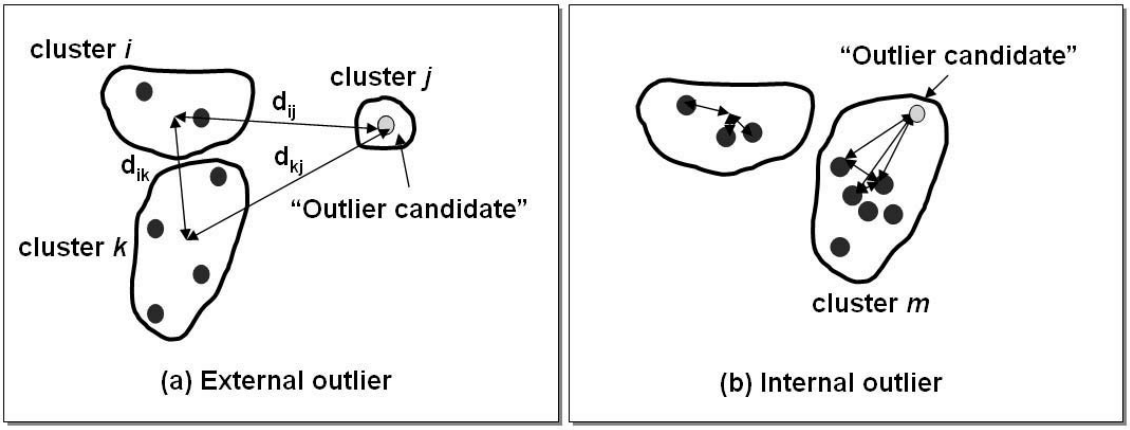
Segundo (HAN; KAMBER, 2006), *outliers* são dados que não estão em conformidade com o comportamento geral ou modelo de dados. Ou seja, são dados consideravelmente inconsistentes ou incompatíveis com o conjunto de dados restante. (YOON; KWON; BAE,

2007) explicam que os *outliers* podem comprometer a qualidade dos dados, portanto é importante a identificação e o descarte dos *outliers* antes da definição dos valores de referência.

(HAN; KAMBER, 2006) apresentam várias técnicas para identificação de *outliers*. Uma abordagem simples e funcional é proposta por (YOON; KWON; BAE, 2007). Ela consiste em utilizar um algoritmo de clusterização, o K-Means, para a identificação dos *outliers*. O processo de agrupar um conjunto de objetos similares em classes ou grupos é denominado clusterização. A similaridade entre os objetos é calculada através dos atributos de cada um deles. Existem vários tipos de métodos de clusterização, como por exemplo os métodos de particionamento, hierárquicos, baseados em densidade, baseados em grade, baseados em modelos, etc. (HAN; KAMBER, 2006).

O algoritmo de clusterização K-Means é um dos mais difundidos e comumente usados. A letra "k" representa o número de *clusters*, ou grupos, a serem formados. Esse algoritmo agrupa os elementos movendo-os entre *clusters* em diferentes etapas do processo. Essas etapas são executadas até que a distância euclidiana total entre os elementos do mesmo *cluster* e seu centro seja mínima.

Figura 4 – Identificação de *outliers* através do K-Means (YOON; KWON; BAE, 2007)



A Figura 4 ilustra a identificação de *outliers* através do algoritmo de clusterização K-Means. Uma vez formados os *clusters*, é possível identificar os elementos que ficam distantes dos outros grupos (*outlier candidate*). Esses elementos são os *outliers* que podem ser descartados antes da definição de valores de referência.

2.5 Definição de Valores de Referência

Segundo (ALVES; YPMA; VISSER, 2010), métricas têm sido utilizadas com sucesso para quantificar diversos tipos de atributos em entidades de software, contudo, têm falhado em servir de apoio para a tomada de decisão. Para que as métricas possam dar esse tipo de suporte aos gerentes de software, é importante que elas estejam acompanhadas de valores de referência, pois apenas com eles o gerente poderá classificar, interpretar e compreender os resultados das métricas de forma apropriada.

Alguns estudos apontam que as métricas de software estão em conformidade com distribuições de cauda pesada (*heavy-tailed distributions*). Nesse tipo de distribuição a média é pouco representativa e, portanto, não deve ser usada para classificação dos valores das métricas (OLIVEIRA; VALENTE; LIMA, 2014). Três técnicas de derivação de valores de referência são discutidas a seguir: (i) baseada na opinião de especialistas; (ii) baseada em *benchmark*; (iii) baseada em algoritmos de clusterização.

2.5.1 Baseada na Opinião de Especialistas

Segundo (ALVES; YPMA; VISSER, 2010), muitos autores definem valores de referência para métricas com base em sua própria experiência. Por exemplo, (MCCABE, 1976) definiu o valor dez como sendo o limiar mínimo para classificar um método muito complexo. Valores de referência definidos com base na experiência de especialistas carecem de fundamento científico e podem ter sido definidos num contexto específico, portanto, podem não ser universalmente aplicáveis (ALVES; YPMA; VISSER, 2010).

É provável que um especialista, com base na sua experiência, consiga definir valores de referência para métricas diretas, como a quantidade de linhas de código produzidas por semana por um desenvolvedor, ou a quantidade de *bugs* gerados num mês. Todavia, definir valores de referência para métricas derivadas, ou seja, composta por uma ou mais operações matemáticas entre métricas diretas, parece ser uma tarefa mais complicada do que derivar valores de referência para métricas diretas.

2.5.2 Baseada em *Benchmark*

(ALVES; YPMA; VISSER, 2010) propuseram uma técnica de derivação de valores de referência que atendesse aos seguintes requisitos: (i) não pode ser baseada na opinião de especialistas, mas sim baseada nos dados de um conjunto representativo de sistemas; (ii) deve

respeitar as propriedades estatísticas da métrica, como a escala e a distribuição; (iii) deve ser repetível, transparente e simples de realizar.

A técnica proposta por (ALVES; YPMA; VISSER, 2010) usa os próprios valores das métricas para a derivação dos valores de referência, por isso é dita como baseada em *benchmark*. Portanto, para usar essa técnica, é preciso ter um conjunto representativo de valores das métricas. Quanto mais valores, mais confiáveis são os valores de referência. Em seu estudo, (ALVES; YPMA; VISSER, 2010) derivam tais valores de referência para a métrica de complexidade ciclomática de McCabe usando um *benchmark* de cem sistemas.

A técnica em si consiste em seis passos descritos abaixo.

1) **Extração da métrica**: para cada entidade de cada sistema do *benchmark*, extrair a métrica e o peso da entidade. Exemplo: sendo a entidade um método, a métrica poderia ser a complexidade ciclomática de McCabe, e o peso seria a quantidade de linhas de código do método;
2) **Cálculo do peso relativo**: para cada entidade, calcula-se o peso relativo da entidade dentro do sistema. Exemplo: Divide-se a quantidade de linhas do método pela quantidade de linhas do sistema;
3) **Agregação das entidades**: agrega-se os pesos de todas as entidades por valor da métrica. Exemplo: agrega-se a quantidade de linhas de código de todas as entidades do sistema que possuem a mesma complexidade de McCabe;
4) **Agregação entre sistemas**: a agregação do passo anterior é realizada novamente, contudo, dessa vez entre as entidades dos sistemas do *benchmark*. Exemplo: agrega-se a quantidade de linhas de código de todas as entidades de todos os sistemas do *benchmark*, que tenham a mesma complexidade de McCabe;
5) **Agregação dos pesos relativos**: ordena-se os valores das métricas de forma ascendente, e identifica-se o maior valor da métrica que represente 1%, 2%, 3%, ..., 100% do peso. Exemplo: ordena-se os valores de complexidade de McCabe de forma ascendente, e identifica-se o maior valor que represente cada percentual;
6) **Derivação dos valores de referência**: os valores de referência são derivados escolhendo o percentual de código que se deseja representar.

2.5.3 Baseada em Algoritmos de Clusterização

O propósito dos valores de referência é dividir os dados das métricas em grupos, permitindo que o usuário da métrica possa classificá-los e melhor compreendê-los. No entanto, não é estritamente necessário ter-se valores de referência para dividir os dados em grupos, pois os algoritmos de clusterização são capazes de dividir um conjunto de objetos em grupos sem necessariamente definir quais são os valores de referência.

Exemplo disso é a abordagem qualitativa, baseada em métricas de software, proposta por (OLIVEIRA et al., 2013) para identificar a similaridade entre softwares orientados a objeto.

> Defining threshold for software metrics at class level is not a trivial task [1, 5]. For example, to obtain a threshold to the LOC metric is complex. The main reason is that LOC follows a heavy-tailed distribution. In other words, systems have many classes with few LOC (e.g., less than a hundred lines) and few classes with many LOC (e.g., in some cases, thousands of lines). Therefore, the mean of LOC is unrepresentative, which makes it more challenging to analyze these systems. (OLIVEIRA et al., 2013)

Para contornar esse problema, (OLIVEIRA et al., 2013) propõe utilizar um algoritmo de clusterização, o K-Means, para dividir os valores em grupos automaticamente. Portanto, apesar da técnica usada não derivar valores de referência, ainda assim é possível dividir os valores em grupos para melhor analisá-los e compreendê-los.

2.6 Sumário

Este capítulo apresentou os conceitos teóricos que fundamentam a abordagem proposta no próximo capítulo. Foram abordados temas relativos a mineração de repositórios de software, métricas de software, a infra-estrutura sobre a qual as métricas integrantes da abordagem foram implementadas, técnicas de derivação de valores de referência e também técnicas de identificação de *outliers*.

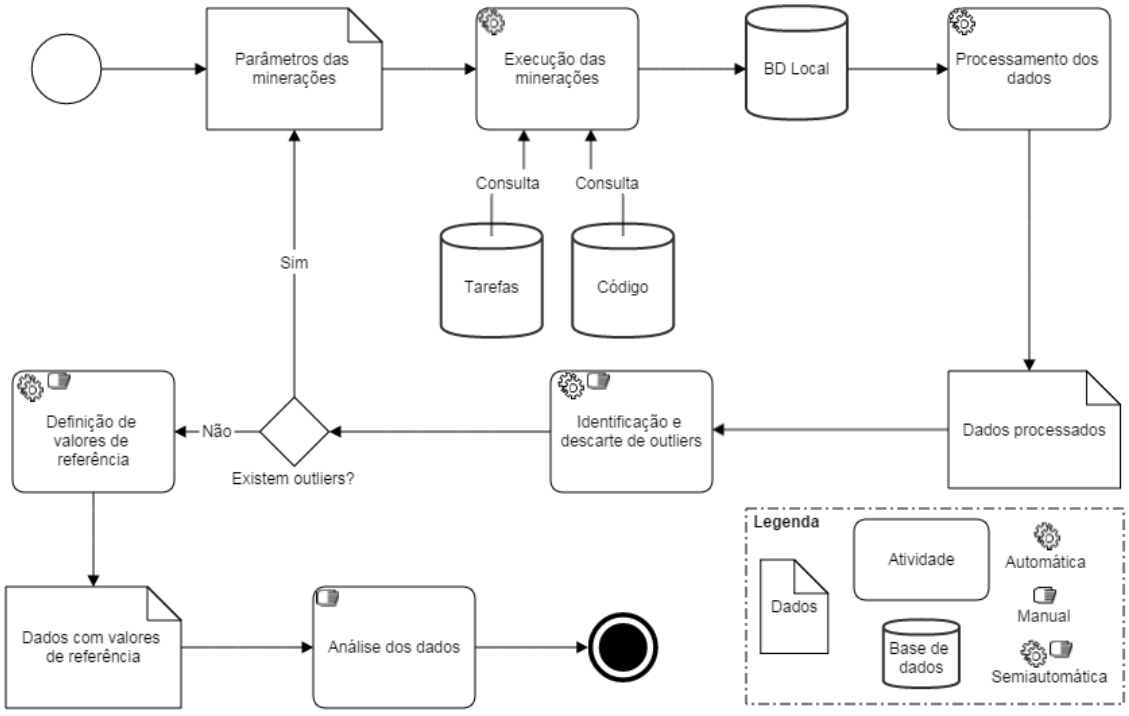
3 ABORDAGEM DE ANÁLISE DA CONTRIBUIÇÃO INDIVIDUAL DE DESENVOLVEDORES

Este capítulo apresenta a abordagem de apoio à gerência de projetos de software para análise da contribuição individual dos desenvolvedores. A Seção 3.1 apresenta uma visão geral da abordagem. A Seção 3.2 detalha os principais elementos que compõem a abordagem, e também de que forma tais elementos foram desenvolvidos.

3.1 Visão Geral da Abordagem

Esta seção apresenta uma visão geral da abordagem de avaliação da contribuição de desenvolvedores proposta nesta dissertação. O objetivo principal da abordagem é fornecer, ao gerente de software, indicadores relativos à contribuição individual de seus desenvolvedores, oferecendo assim suporte à tomada de decisão. Ela consiste em cinco atividades principais que são detalhadas a seguir. A Figura 5 apresenta a organização sequencial de tais atividades utilizando a notação *Business Process Model and Notation* (BPMN) (OMG, 1997).

Figura 5 – Organização das atividades da abordagem de acompanhamento



A primeira atividade da abordagem é a execução das minerações para extração dos dados. Os dados são extraídos, de forma automatizada, do repositório de tarefas e do sistema

de controle de versão do projeto, e posteriormente são persistidos num banco de dados para processamento na atividade seguinte. Apesar dessa primeira atividade ser automatizada, é necessário fornecer alguns parâmetros para que as minerações executem de forma apropriada.

(i) **Uma lista com os nomes de usuário dos desenvolvedores no sistema de controle de versão**: necessário para que a coleta dos dados seja realizada apenas para os desenvolvedores desejados;

(ii) **Uma lista com os caminhos (*paths*) a serem ignorados pela mineração**: esse é um parâmetro opcional, que informa para a mineração quais são os caminhos (*paths*) que devem ser ignorados por ela. Essa informação é útil para o descarte de *outliers*;

(iii) **A data de início do período a ser minerado**: indica a partir de qual data a mineração deverá coletar dados;

(iv) **A data de término do período a ser minerado**: indica até qual data a mineração deverá coletar dados.

As minerações que executam nessa atividade permitem extrair dados dos repositórios do projeto para que as métricas sejam produzidas. As métricas avaliam a contribuição individual dos desenvolvedores através de quatro atributos: (i) volume de contribuição, discutido na Seção 3.3.1; (ii) complexidade média por método, apresentado na Seção 3.3.2; (iii) introdução de *bugs*, discutido na Seção 3.3.3; e (iv) contribuição em correção de erros, apresentado na Seção 3.3.4. Vale salientar que vários outros atributos poderiam ser considerados na avaliação da contribuição individual dos desenvolvedores, contudo, a escolha de muitos atributos de avaliação pode não ser interessante, uma vez que eles poderiam sobrecarregar o gerente de projeto de software com muitas informações, dificultando a interpretação, uma vez que a presente abordagem não oferece, para o gerente de projeto de software, meios para interpretar as métricas de forma associada ou conjunta. Por esse motivo optou-se por um conjunto limitado de atributos que analisam diferentes aspectos de contribuição.

Uma vez minerados os dados de contribuição dos desenvolvedores nos repositórios, a segunda atividade da abordagem é o processamento dos dados extraídos pelas minerações. Essa atividade envolve a consulta dos dados persistidos no banco de dados pela atividade anterior, o cálculo das métricas através dos dados consultados e a posterior exportação das

métricas para um formato (CSV, XML, XLS, etc.) que permita a análise através de outras ferramentas.

A terceira atividade é a identificação de *outliers*, que consiste em uma etapa semi-automatizada, pois parte dela pode ser realizada automaticamente, como a ordenação dos valores da métrica, mas a inspeção e análise é realizada, de forma manual, pelo gerente de software, para determinar se a contribuição do desenvolvedor de fato possui *outliers*. Caso sejam constatados *outliers* na contribuição do desenvolvedor, os caminhos (*paths*) de onde essa falsa contribuição foi realizada devem ser incluídos no minerador, para que ele possa ignorá-los durante a mineração. Em seguida, a mineração deve ser re-executada. Os detalhes dessa atividade são detalhados na Seção 3.2.2.

A quarta atividade da abordagem consiste em definir os valores de referência para as métricas. Essa também é uma etapa semi-automatizada do processo pois, embora gráficos possam ser gerados automaticamente para auxiliar os gerentes, a escolha dos valores de referência específicos para as métricas é feita de forma manual, seguindo as orientações discutidas na Seção 3.2.3.

De posse dos dados das métricas e de seus respectivos valores de referência, o gerente de software pode analisá-los e confrontá-los para tirar conclusões sobre a contribuição individual de seus desenvolvedores. A Seção 3.3 apresenta as orientações de interpretação das métricas tendo em vista o suporte para a adequada tomada de decisão no que diz respeito a contribuição dos desenvolvedores.

3.2 Componentes Principais da Abordagem

A abordagem proposta nesta dissertação é composta por três elementos principais: (i) as minerações e as métricas de contribuição dos desenvolvedores; (ii) a técnica de identificação e descarte de *outliers*; e (iii) a técnica de definição de valores de referência. Esta seção explica, em detalhes, os fundamentos e como foram desenvolvidos cada um desses componentes.

3.2.1 Minerações e Métricas de Contribuição

A abordagem de acompanhamento objetiva avaliar a contribuição individual do desenvolvedor através de quatro atributos: (i) volume da contribuição; (ii) complexidade média por método; (iii) introdução de *bugs*; (iv) contribuição em correção de *bugs*. Cada

atributo é avaliado através de um conjunto de métricas que, por sua vez, são extraídas através de minerações de repositórios de software. As métricas e as minerações usadas pela abordagem são baseadas naquelas apresentadas por (COSTA et al., 2014), com exceção das métricas que avaliam o atributo (ii) de complexidade média por método, que foi elaborado no contexto desta dissertação de mestrado.

A escolha das métricas baseadas em (COSTA et al., 2014) se justifica por duas razões: (i) permitem avaliar a contribuição individual dos desenvolvedores através de diferentes atributos, como produção de código-fonte, participação em correção de *bugs* e introdução de *bugs* no sistema; e (ii) foi um trabalho conduzido por um membro do mesmo grupo de pesquisa do qual em que o autor e orientador desta dissertação fazem parte, havendo, portanto, motivações relacionadas à continuidade da pesquisa realizada anteriormente. Diversas outras métricas poderiam ser usadas, contudo, com o intuito de não sobrecarregar o usuário da abordagem com muitas informações, optou-se por escolher um conjunto de métricas que avaliassem os quatro atributos apresentados na segunda coluna da Tabela 2.

Tabela 2 - Equivalência entre os atributos

Atributos avaliados por Costa (2012)	**Atributos avaliados pela abordagem**
Tamanho dos *commits*	Volume de contribuição
Resolução de *bugs* prioritários	Contribuição em correção de *bugs*
Commits defeituosos	Introdução de *bugs*
-	Complexidade média por método

A Tabela 2 apresenta a equivalência entre os atributos avaliados por (COSTA, 2012) e a abordagem apresentada nesse capítulo. É importante salientar que as implementações das minerações desenvolvidas por (COSTA et al., 2014) sofreram algumas melhorias ao serem incorporadas na abordagem discutida neste capítulo, conforme discutido a seguir.

As primeiras melhorias foram realizadas nas minerações que quantificam o tamanho dos *commits*. O principal problema com as minerações originais era o fato de utilizarem o *diff* do próprio repositório de código (Subversion), que é limitado, pois não indica quais linhas de código foram modificadas. No Subversion, quando uma linha de código é modificada, ele considera que a linha original (antes de ser modificada) foi removida, e a linha modificada foi adicionada logo abaixo, conforme ilustrado no *commit* hipotético ilustrado na Figura 6.

Figura 6 - Modificando uma linha de código no Subversion (Costa, 2012)

```
Revisão : 130
Autor : Alice <alice@projeto.com>
Comentário : Eu corrigi um defeito
@@-50,1 + 50,1 @@
- if ( i <= 128 ) {
+ if ( i < 128 ) {
```

No *commit* hipotético ilustrado na Figura 6, Alice, autora do *commit*, modificou o operador menor-ou-igual para o operador menor. O Subversion, contudo, não indica que aquela linha foi modificada, mas indica, com um sinal de menos, a linha antes da modificação, e com um sinal de mais indica a linha após a modificação. Tal mineração empregada por (COSTA, 2012) pode resultar em falsos-positivos, pois ela pode contabilizar que linhas foram adicionadas e removidas, quando na realidade elas foram apenas modificadas.

Esse problema foi corrigido quando essas métricas foram incorporadas à abordagem desde trabalho. Para solucioná-lo, a mineração não utiliza mais o *diff* do Subversion, mas sim uma biblioteca externa de *diff* textual[7]. Ou seja, quando a mineração identifica um artefato Java modificado no *commit*, ela faz o *download* do conteúdo desse artefato antes e depois da modificação, e usa esse *diff* para contabilizar exatamente quantas linhas foram adicionadas, modificadas e removidas. Dessa forma, a melhoria realizada nessas métricas não só corrige o problema dos falso-positivos, como também oferece um novo e importante indicador, que é a quantidade exata de linhas modificadas.

Outro indicador que foi incorporado a abordagem, mas que não existia na suíte de métricas proposta originalmente por (COSTA, 2012) é a métrica que quantifica o número de métodos removidos pelo desenvolvedor. Outra melhoria realizada foi em relação à identificação de *commits* defeituosos. A mineração usada por (COSTA, 2012) (Seção 2.3.1.1) possui apenas uma única estratégia para identificação de *commits* defeituosos. Uma nova estratégia foi elaborada e incorporada à abordagem desta dissertação, que identifica *commits*

[7] https://code.google.com/p/java-diff-utils/

defeituosos de código que ainda não foram para produção. Os detalhes dessa nova estratégia são apresentados na Seção 3.3.3.

Dependendo da dinâmica de trabalho de onde as métricas estão sendo aplicadas, a estratégia adotada por (COSTA, 2012) para quantificar o número de *bugs* prioritários resolvidos por cada desenvolvedor pode apresentar alguns problemas. O primeiro problema pode ocorrer quando não for o desenvolvedor que finalize/conclua a tarefa de correção de erro prioritário. Dependendo da dinâmica de trabalho, isso pode ser realizado por um gerente de software, por um testador ou mesmo alguém externo à equipe.

Devido à estratégia de atribuir a conclusão da tarefa ao desenvolvedor que mudou seu estado para concluída/finalizada, outro problema que pode ocorrer é quando mais de um desenvolvedor tiver contribuído para a conclusão da tarefa. Apenas o desenvolvedor que alterou o estado da tarefa será considerado.

Dessa forma, outra mudança realizada em relação às métricas originais propostas por (COSTA, 2012) foi aplicada à métrica de resolução de *bugs* prioritários. Conforme o nome deixa claro, a métrica considera apenas tarefas de correção de *bugs* de alta prioridade, portanto não contabiliza tarefas de correção de menor prioridade, e dessa forma pode dar a falsa interpretação de que um dado desenvolvedor possa ter uma contribuição na correção de erros menor do que de fato ele teve durante o período analisado. A métrica relativa à contribuição em correção de *bugs*, proposta na abordagem avaliada por este estudo, leva em conta qualquer tarefa de correção, independente da prioridade atribuída a ela.

Finalmente, a abordagem proposta nesta dissertação é composta por uma nova métrica que é capaz de identificar como o desenvolvedor de software distribuiu a complexidade ciclomática de sua contribuição, permitindo que o gerente de software possa avaliar a qualidade da codificação de cada desenvolvedor, tendo em vista o esforço de manutenção futuro do código que aquele desenvolvedor produz. Os detalhes dessa mineração são apresentados na Seção 3.3.2.

3.2.2 Identificação e Descarte de *Outliers*

O segundo componente fundamental da abordagem é a técnica de identificação e descarte de *outliers*. Conforme explicado na Seção 2.4, um *outlier* é um dado consideravelmente inconsistente ou incompatível, que não representa a realidade, e pode comprometer a qualidade das informações extraídas do conjunto de dados coletados. Desse

modo é importante que a identificação e descarte dos *outliers* aconteça antes de qualquer análise dos dados, inclusive a definição dos valores de referência.

Um cenário que pode gerar a ocorrência de *outliers* com as métricas propostas por esse estudo é o seguinte: um desenvolvedor pode realizar um *commit* de um sistema ou módulo completo do sistema, mesmo que ele não o tenha desenvolvido, ou tenha desenvolvido num período maior do que aquele fornecido através dos parâmetros de entrada das métricas. Nesse caso, as métricas de volume de contribuição vão indicar grande quantidade de produção de código do desenvolvedor que incluiu o sistema/módulo no controle de versão, mesmo não sendo ele o responsável por tal feito. Um fator complicador é que a real contribuição de código desse desenvolvedor, aquela que de fato ele implementou durante o período, vai se confundir com o *outlier*.

Nesse contexto, a técnica de identificação de *outliers* apresentada por (YOON; KWON; BAE, 2007) (Seção 2.4), baseada no algoritmo de clusterização K-Means, não deve ser usada, uma vez que ela descartaria também a real contribuição de código realizada pelo desenvolvedor. Portanto, para evitar esse cenário, a abordagem proposta nesse trabalho oferece os seguintes passos para identificação dos *outliers* de suas métricas:

1) **Identificar contribuições com possíveis *outliers***: ordenar os valores das métricas de forma crescente, e verificar se existem valores que divergem do resto do conjunto de dados (possíveis *outliers*). Caso positivo, prosseguir com o passo 2;
2) **Determinar se a contribuição de fato possui *outliers***: através de uma análise manual dos *commits* realizados pelo desenvolvedor e pelas suas tarefas trabalhadas durante o período de interesse, o gerente de software deve analisar se os possíveis *outliers*, identificados no passo anterior, são de fato *outliers*, ou são contribuição real do desenvolvedor em questão. Caso seja constatado que os valores são *outliers*, prosseguir com o passo 3;
3) **Descartar os *outliers***: uma vez identificados *outliers*, incluir, na mineração, os caminhos (*paths*) que devem ser ignorados por ele para evitar a contaminação dos dados;
4) **Reexecutar a mineração**: executar a mineração novamente para o período desejado, de modo que a mineração não considere os *outliers* presentes nos caminhos ignorados.

Através dos passos descritos acima, o gerente de software pode identificar a ocorrência de *outliers* nos dados extraídos, permitindo que ele descarte-os de forma apropriada e considere apenas a contribuição do desenvolvedor.

3.2.3 Definição de Valores de Referência

O terceiro e último componente fundamental da abordagem é a técnica para definição de valores de referência. Segundo (FENTON; NEIL, 2000), métricas têm sido usadas com sucesso para quantificar atributos em entidades de projetos de software, contudo elas geralmente falham em dar suporte à tomada de decisão. Visando tal objetivo, as métricas devem estar acompanhadas por valores de referência, que permitam a classificação e interpretação apropriada por parte dos gerentes de software.

Uma vez que os valores das métricas estejam livres de *outliers*, o terceiro passo da metodologia é definir os valores de referência. A técnica apresentada por esta dissertação para definir os valores de referência das métricas propostas é uma simplificação da técnica proposta por (ALVES; YPMA; VISSER, 2010). Essa escolha se deve às características dessa técnica: (i) não depende da opinião de especialistas; (ii) é baseada num conjunto representativo de dados; (iii) respeita as propriedades estatísticas de cada métrica; (iv) é simples de realizar.

A técnica proposta por (ALVES; YPMA; VISSER, 2010) (Seção 2.5.2) possui passos que não são necessários para o contexto particular das métricas propostas nesta dissertação de mestrado. Exemplo disso é o cálculo do peso relativo (passo 2), que calcula o peso relativo do método dentro do sistema. Isso é feito através da divisão entre a quantidade de linhas de código do método pela quantidade de linhas de código do sistema.

Outro exemplo de passos que não se adequam nesse contexto particular são os passos que envolvem agregações (passos 3 e 4). No passo 3 ocorre uma agregação dos pesos (LOC) dos métodos de um sistema que possuem o mesmo valor de métrica (McCabe). No passo 4 ocorre essa mesma agregação, contudo entre sistemas diferentes. Dessa forma, os passos da técnica proposta por aquele trabalho e que se aplicam a esse contexto particular são os seguintes: (i) extração das métricas; (vi) agregação dos pesos relativos; (vii) derivação dos valores de referência.

Esses passos foram reorganizados para formar a sequência de quatro passos para a derivação de valores de referência proposta por esta dissertação.

1) Extrair os valores de cada métrica através das minerações;
2) Determinar os percentis de cada valor extraído através da ordenação dos mesmos de forma crescente, permitindo identificar o maior valor da métrica que represente 1%, 2%, 3%, ..., 100% dos valores;
3) Plotar tais valores num gráfico de dispersão, no qual o eixo X apresenta os valores percentuais, de 1% até 100%, e o eixo Y apresenta os valores da métrica;
4) Observar e identificar onde ocorrem as variações nos valores da métrica. Definir a quantidade desejada de valores de referência onde ocorrem essas variações.

Para melhor ilustrar como executar cada passo, apresenta-se a seguir um exemplo demonstrando como derivar três valores de referência para a métrica de quantidade de linhas de código adicionadas durante uma semana. O primeiro passo é a extração dos dados da métrica. Para tanto, o exemplo irá usar o seguinte conjunto de dados, que são uma fração de um conjunto de dados reais de uma equipe de desenvolvimento específica, durante um período específico. Tais dados são apresentados na Tabela 3.

Tabela 3 – Passo 1: extrair os dados da métrica

1	19	31	72	95	177	271	333	536
3	19	36	72	100	179	277	366	555
4	19	36	73	114	190	279	375	572
7	20	38	73	130	204	284	390	637
9	22	41	78	134	205	287	408	671
12	23	41	79	135	232	290	430	730
12	27	45	79	136	233	300	458	776
14	29	48	86	150	262	307	491	817
16	29	49	86	153	266	313	493	898
18	29	61	89	160	268	318	514	988

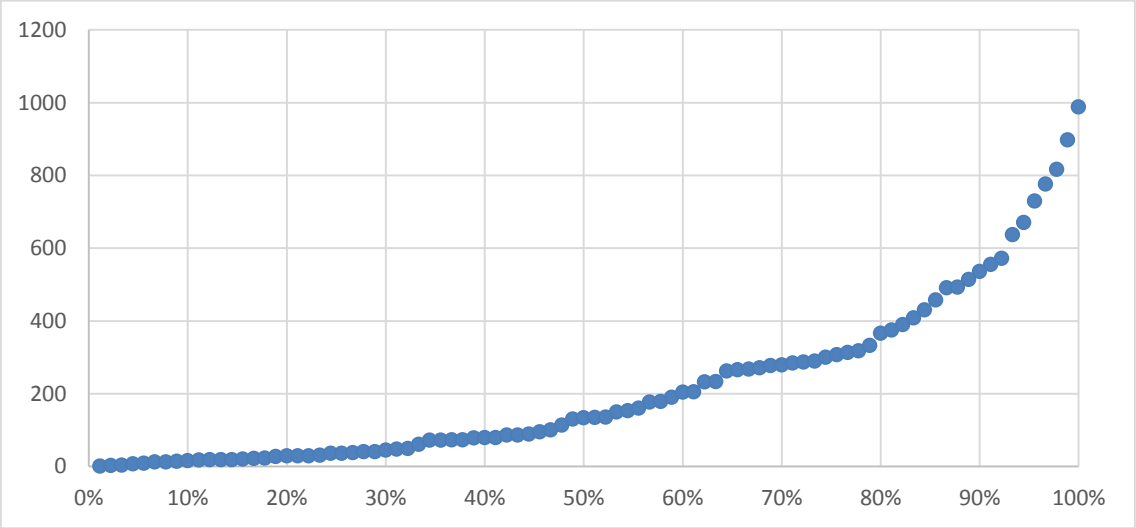
No segundo passo, define-se os percentis de cada valor através da ordenação deles de forma crescente, permitindo identificar o maior valor que representa cada percentual, conforme ilustrado na Tabela 4.

Tabela 4 – Passo 2: definição dos percentis dos valores da métrica

%	LOC	%	LOC	%	LOC	%	LOC	%	LOC
1%	1	23%	31	46%	95	68%	271	90%	536
2%	3	24%	36	47%	100	69%	277	91%	555
3%	4	26%	36	48%	114	70%	279	92%	572
4%	7	27%	38	49%	130	71%	284	93%	637
6%	9	28%	41	50%	134	72%	287	94%	671
7%	12	29%	41	51%	135	73%	290	96%	730
8%	12	30%	45	52%	136	74%	300	97%	776
9%	14	31%	48	53%	150	76%	307	98%	817
10%	16	32%	49	54%	153	77%	313	99%	898
11%	18	33%	61	56%	160	78%	318	100%	988
12%	19	34%	72	57%	177	79%	333		
13%	19	36%	72	58%	179	80%	366		
14%	19	37%	73	59%	190	81%	375		
16%	20	38%	73	60%	204	82%	390		
17%	22	39%	78	61%	205	83%	408		
18%	23	40%	79	62%	232	84%	430		
19%	27	41%	79	63%	233	86%	458		
20%	29	42%	86	64%	262	87%	491		
21%	29	43%	86	66%	266	88%	493		
22%	29	41%	89	62%	268	82%	514		

O terceiro passo é a plotagem dos valores num gráfico de dispersão, no qual o eixo X apresenta os valores percentuais, e o eixo Y apresenta os valores da métrica. Esse gráfico permite observar a variação dos valores da métrica, conforme ilustrado na Figura 7.

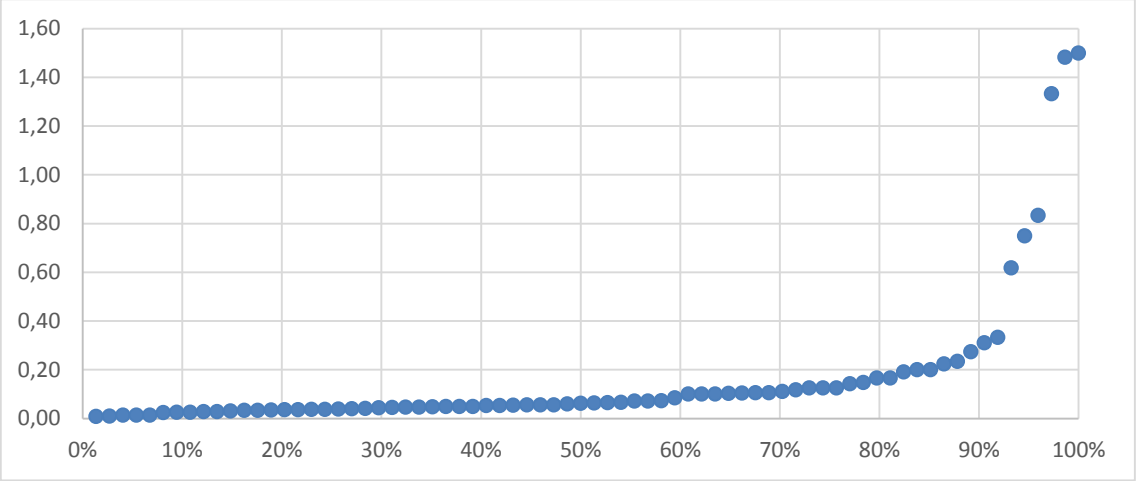
Figura 7 – Passo 3: plotagem dos dados num gráfico de dispersão



O quarto e último passo é observar e compreender como os valores variaram ao longo do tempo, e finalmente, definir com base nessa variação, a quantidade de valores de referência desejada. No gráfico de dispersão ilustrado na Figura 7, é possível perceber que a variação ocorre ao longo de quase toda a distribuição, acentuando-se um pouco mais a partir de 30%. A proposta desse exemplo é definir três valores de referência para essa métrica, deste modo poderiam ser definidos os seguintes valores de referência: 30% (45), 60% (204) e 90% (536).

Com esses três valores de referência, o gerente de software poderia considerar que a produção: (i) de menos de 45 linhas de código durante o período é um péssimo resultado; (ii) entre 45 e 204 seria um resultado ruim; (iii) entre 204 e 536 seria considerado um bom resultado; e (iv) acima de 536 seria encarado como um excelente resultado. Vale salientar que essa interpretação vale apenas para esse atributo específico de avaliação, ou seja, produção de linhas de código-fonte.

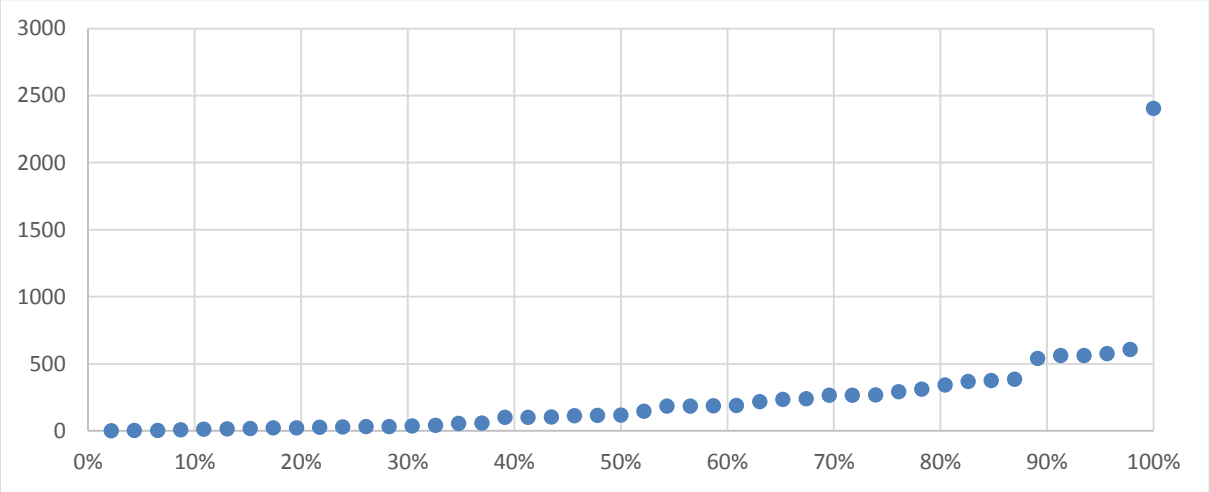
Figura 8 – Gráfico de dispersão com distribuição de cauda pesada



O gráfico de dispersão ilustrado na Figura 8 apresenta dados de uma observação realizada num período específico, com uma equipe de desenvolvimento, da métrica de complexidade média em métodos modificados. É possível observar que a distribuição está mais próxima do formato de uma distribuição de cauda pesada, quase não havendo variação até 50% dos valores. Entre 50% e 60% há uma pequena variação, e a partir de 60% os valores passam a variar bem mais. Ou seja, pode-se afirmar que a variação ocorre a partir de 60%. Nesse cenário, um gerente de software poderia derivar os valores de referência em 60% (0,09), 90% (0,29) e 95% (0,75).

É interessante notar também que a técnica para definição de valores de referência também pode ajudar na identificação de *outliers*. Portanto, mesmo que o gerente de software não tenha detectado os *outliers* na etapa anterior, eles ficam evidentes quando os valores da métrica são plotados num gráfico de dispersão, conforme ilustrado na Figura 9, que mostra valores reais colhidos do SIPAC em agosto de 2014.

Figura 9 - Gráfico de dispersão evidenciando um possível *outlier*



Cada ponto no gráfico é a soma de linhas de código adicionadas, modificadas e removidas por um desenvolvedor durante uma semana. O gráfico também mostra que 98% dos valores apresentados são menores ou iguais a 608 linhas de código. É possível observar também o valor de 2404 linhas de código, que é quase quatro vezes superior ao segundo maior valor apresentado pelo gráfico (608), e que, dada a sua distância em relação ao restante do conjunto de valores da métrica, pode representar um *outlier*. Dessa forma, cabe ao gerente de software analisar a contribuição de código realizada pelo desenvolvedor e verificar se realmente houve a ocorrência de *outliers*.

3.3 Atributos e Métricas de Contribuição de Desenvolvedores

Esta seção apresenta, em detalhes, como funcionam as minerações da abordagem avaliada por esta dissertação, bem como as métricas geradas, a interpretação de seus dados e as ameaças conhecidas para cada uma delas. A abordagem de análise da contribuição individual dos desenvolvedores avalia a contribuição dos desenvolvedores sob a ótica de quatro atributos: (i) volume de contribuição, (ii) complexidade média por método, (iii) introdução de *bugs* e (iv) contribuição em correção de *bugs*.

3.3.1 Volume da Contribuição

O volume da contribuição é um atributo que quantifica a contribuição individual em termos de desenvolvimento de código. Esse atributo é avaliado através de seis métricas distintas:

(i) **Quantidade de linhas de código adicionadas**: quantifica o número de linhas de código adicionadas em artefatos Java por cada desenvolvedor, desconsiderando linhas em branco e comentários;

(ii) **Quantidade de linhas de código modificadas**: idêntica à anterior, mas quantifica linhas modificadas;

(iii) **Quantidade de linhas de código removidas**: idêntica à anterior, mas quantifica linhas removidas;

(iv) **Quantidade de métodos adicionados**: quantifica o número de métodos adicionados em artefatos Java por cada desenvolvedor;

(v) **Quantidade de métodos modificados**: idêntica à anterior, mas quantifica métodos modificados;

(vi) **Quantidade de métodos removidos**: idêntica à anterior, mas quantifica métodos removidos.

3.3.1.1 Interpretação das métricas

Usando os valores de referência para as referidas métricas, o gerente de software pode identificar valores baixos, intermediários e altos para as métricas de volume da contribuição. Altos valores para as métricas de volume de contribuição indicam que o desenvolvedor adicionou, modificou e/ou removeu grande quantidade de código-fonte. Naturalmente, baixos valores indicam baixa contribuição em código-fonte no período.

3.3.1.2 Estratégia de mineração

A estratégia de mineração dessas métricas consiste em oito passos:

1) **Consulta dos *commits* no período**: a mineração realiza uma consulta ao repositório de código em busca dos *commits* realizados durante o período desejado, utilizando o período de interesse fornecido através dos parâmetros de entrada;

2) **Filtro dos desenvolvedores**: cada *commit* é verificado para checar se ele foi realizado por um dos desenvolvedores contidos na lista, utilizando a lista de desenvolvedores de interesse fornecida através dos parâmetros de entrada. Os *commits* realizados por desenvolvedores fora da lista são desconsiderados;

3) **Coleta de artefatos manipulados**: para cada *commit* identificado no passo anterior, a mineração captura os artefatos manipulados, sejam adicionados, modificados ou removidos;
4) **Filtro de caminhos ignorados**: a mineração aplica mais um filtro, mas dessa vez sobre os artefatos identificados no passo anterior. O filtro descarta os artefatos que estão nos caminhos (*paths*) fornecidos através dos parâmetros de entrada;
5) **Filtro de artefatos**: um outro filtro é aplicado sobre os artefatos, desconsiderando aqueles que foram deletados e aqueles que não possuem a extensão ".java". Esse último filtro cria uma lista de artefatos de interesse da mineração;
6) **Coletando a lista de revisões**: para cada artefato dentro da lista de interesse criada no passo anterior, a mineração realiza uma consulta no sistema de controle de versões em busca das revisões de cada artefato;
7) **Calculando as métricas**: nesse passo existem duas estratégias diferentes para lidar com artefatos modificados e artefatos adicionados.
    a) **Artefatos modificados**: identifica-se a revisão atual do artefato (HEAD) e a revisão anterior (antes de modificação) através da lista de revisões do artefato, elaborada no passo anterior. Com essas duas revisões é possível realizar um "diff" para determinar exatamente quais linhas[8] foram adicionadas, modificadas e removidas pelo desenvolvedor. Através da identificação de linhas modificadas, é possível determinar quais métodos foram modificados. A métrica de métodos adicionados é feita verificando quais métodos não existiam antes da modificação e passaram a existir depois. A métrica de métodos removidos é feita usando o inverso dessa estratégia;
    b) **Artefatos adicionados**: para os artefatos adicionados existem apenas linhas e métodos adicionados. Portanto, a mineração precisa apenas calcular quantas linhas[8] de código e quantos métodos tem o artefato.

[8] Desconsideram-se linhas em branco e comentários.

8) **Identificação das tarefas**: o último passo da mineração é identificar quais tarefas demandaram os *commits* realizados pelos desenvolvedores. Isso é realizado fazendo uma consulta ao repositório de tarefas (*issues*) do projeto, usando os números de revisões dos *commits*.

3.3.1.3 Ameaças Conhecidas

Uma ameaça previamente conhecida para as métricas que quantificam linhas de código decorre do fato de que elas só levam em consideração linhas de código produzidas, modificadas e removidas em artefatos com a extensão ".java". Essa ameaça pode ocorrer quando um desenvolvedor contribuir em código em outros artefatos, como páginas JSP, *scripts* SQL, arquivos de configuração XML, etc. Essa contribuição não será quantificada pelas métricas que avaliam a contribuição em LOC no presente trabalho, mas pode ser considerada em trabalhos futuros que envolvam projetos com grande diversidade de artefatos.

3.3.2 Complexidade Média por Método

A complexidade média por método é uma medida de como ficou dispersa, ou concentrada, a complexidade adicionada pelo desenvolvedor em suas contribuições durante o período de interesse. A complexidade que se refere esse estudo é a complexidade ciclomática de (MCCABE, 1976). Esse atributo de contribuição é avaliado através de duas métricas:

(i) **Complexidade média em métodos adicionados**: somatório da complexidade dos métodos adicionados pelo desenvolvedor no período, dividido pela quantidade de métodos adicionados pelo desenvolvedor no período;

(ii) **Complexidade média em métodos modificados**: somatório dos deltas[9] da complexidade dos métodos modificados pelo desenvolvedor no período, dividido pela quantidade de métodos modificados pelo desenvolvedor no período.

[9] O delta da complexidade é calculado através da subtração entre o valor da complexidade antes da modificação e o valor depois da modificação.

3.3.2.1 Interpretação das métricas

A complexidade ciclomática é um importante atributo a ser medido, pois ela está diretamente associada com a facilidade de manutenção do software, conforme explica (SOMMERVILLE et al., 2007):

> Há diversos estudos de diferentes tipos de complexidade em um sistema (MCCABE, 1976; HALSTEAD, 1977) e de relacionamentos entre complexidade e facilidade de manutenção (KAFURA e REDDY, 1987; Banker, et al., 1993). Não é surpresa que esses estudos tenham descoberto que quanto mais complexo for um sistema ou um componente, mais onerosa será sua manutenção. [...] Eles sugerem que, para reduzir os custos de manutenção, você deve substituir, especificamente, componentes complexos do sistema por alternativas mais simples.

Através das métricas de complexidade média por método, o gerente de software pode descobrir se a complexidade adicionada por cada desenvolvedor ficou concentrada em poucos métodos, o que pode dificultar a manutenção posteriormente; ou se ficou dispersa em vários métodos, o que facilitaria a manutenção futura. Além disso, a métrica permite avaliar a complexidade média do que foi produzido pelos desenvolvedores, ou seja, se os métodos desenvolvidos possuem algoritmos complexos ou são simplesmente métodos simples com pouca lógica de processamento de dados.

Valores altos para essas métricas indicam que o desenvolvedor concentrou a complexidade em poucos métodos, ou seja, ele contribuiu com código de alta complexidade, e distribuiu essa complexidade em poucos métodos. Valores baixos para essas métricas podem indicar dois possíveis cenários:

(i) O desenvolvedor teve pouca contribuição em termos de desenvolvimento de código-fonte. Para descobrir se isso ocorreu, basta que o gerente verifique se as métricas de volume da contribuição (Seção 3.3.1) estão com valores baixos;

(ii) O desenvolvedor contribuiu com código de alta complexidade, mas distribuiu essa complexidade em vários métodos. Para descobrir se isso ocorreu, basta que o gerente verifique se os valores isolados das complexidades adicionadas e modificadas estão altos, e se os valores das métricas de métodos adicionados e modificados também estão com valores altos.

3.3.2.2 Estratégia de mineração

A estratégia de mineração dessas métricas consiste em nove passos:

(i) **Consulta dos *commits* no período**: a mineração realiza uma consulta ao repositório de código em busca dos *commits* realizados durante o período desejado, utilizando o período de interesse fornecido através dos parâmetros de entrada;

(ii) **Filtro dos desenvolvedores**: cada *commit* é verificado para checar se ele foi realizado por um dos desenvolvedores contidos na lista, utilizando a lista de desenvolvedores de interesse fornecida através dos parâmetros de entrada. Os *commits* realizados por desenvolvedores fora da lista são desconsiderados;

(iii) **Coleta de artefatos manipulados**: para cada *commit* identificado no passo anterior, a mineração captura os artefatos manipulados, sejam adicionados, modificados ou removidos;

(iv) **Filtro de caminhos ignorados**: a mineração aplica mais um filtro, mas dessa vez sobre os artefatos identificados no passo anterior. O filtro descarta os artefatos que estão nos caminhos (*paths*) fornecidos através dos parâmetros de entrada;

(v) **Filtro de artefatos**: um outro filtro é aplicado sobre os artefatos, desconsiderando aqueles que foram deletados e aqueles que não possuem a extensão ".java". Esse último filtro cria uma lista de artefatos de interesse da mineração;

(vi) **Coletando a lista de revisões**: para cada artefato dentro da lista de interesse criada no passo anterior, a mineração realiza uma consulta no sistema de controle de versões em busca das revisões de cada artefato;

(vii) **Calculando as complexidades**: para cada artefato, seja ele adicionado ou modificado, os métodos são analisados utilizando estratégias distintas para métodos adicionados e métodos modificados:

   a) **Métodos adicionados**: calcula-se e soma-se a complexidade ciclomática de todos os métodos adicionados pelo desenvolvedor;

   b) **Métodos modificados**: calcula-se a complexidade ciclomática do método antes e depois da modificação. Se esses dois valores forem diferentes, é feita uma subtração entre eles para calcular o delta da complexidade. Se o delta for positivo, soma-se o valor do delta a um somatório dos deltas da complexidade do desenvolvedor; se o delta for negativo, desconsidera-se o valor do delta.

(viii) **Calculando a métrica de complexidade média em métodos adicionados**: para calcular a métrica de complexidade média em métodos adicionados, é feita uma divisão entre o somatório da complexidade adicionada pelo desenvolvedor e a quantidade de métodos adicionados pelo desenvolvedor;

(ix) **Calculando a métrica de complexidade média em métodos modificados**: para calcular a métrica de complexidade média em métodos modificados, é feita uma divisão entre o somatório dos deltas de complexidade modificada pelo desenvolvedor e a quantidade de métodos modificados pelo desenvolvedor.

3.3.2.3 Ameaças conhecidas

Conforme explicado anteriormente (Seção 3.3.2.1), manter a complexidade ciclomática dos métodos baixa é considerada uma boa prática de programação. Contudo, manter a complexidade dos métodos sempre reduzida não é, necessariamente, uma boa prática. Por exemplo: um determinado método m() tem complexidade ciclomática 20, e portanto precisa ser modificado para reduzir a sua complexidade. A divisão desse método em quatro métodos de complexidade 5 já seria suficiente, contudo, um desenvolvedor poderia dividí-lo em 10 métodos de complexidade 2, o que não necessariamente seria algo bom, tendo em vista que uma futura mudança impactaria numa maior quantidade de métodos.

Outra ameaça conhecida para essa métrica é o fato dela descartar os deltas de complexidade negativos. Ou seja, simplificações e/ou refatorações de métodos podem não ser contabilizados por ela. A decisão de desconsiderar os deltas negativos de complexidade foi uma decisão realizada neste trabalho porque não há uma forma clara de se lidar com os valores positivos e negativos dessa métrica.

3.3.3 Introdução de *Bugs*

A introdução de *bugs* é um atributo de avaliação da qualidade da contribuição que é medido através da quantidade de *bugs* introduzidos por um desenvolvedor. Esse atributo de contribuição é medido através de duas métricas:

(i) **Quantidade de *commits* que introduziram *bugs***: quantidade de *commits* realizados por um desenvolvedor em que suas alterações precisaram ser corrigidas posteriormente;

(ii) **Falha em testes**: quantidade de vezes que tarefas retornaram para desenvolvimento em virtude de falha em testes.

3.3.3.1 Interpretação da Métrica

A qualidade daquilo que é produzido por cada desenvolvedor é outro importante atributo que deve ser acompanhado pelos gerentes de software. Um dos atributos de qualidade é a quantidade de *bugs* introduzidos por cada desenvolvedor, durante o desenvolvimento de código-fonte para um dado sistema. Naturalmente, quanto maior for o valor dessas métricas, maior é a quantidade de erros introduzidos pelo desenvolvedor. Contudo, vale salientar que, devido a estratégia de mineração utilizada por essas métricas, a introdução de um erro só é contabilizada quando ele é corrigido. Portanto, os erros presentes em um período específico, não significam que eles foram introduzidos nesse período, mas sim foram corrigidos durante ele.

3.3.3.2 Estratégia de Mineração

As duas métricas possuem estratégias de mineração distintas. A estratégia de mineração da primeira métrica, (i) Quantidade de commits que introduziram bugs, é descrita a seguir.

1) **Busca por tarefas de correção de *bugs***: o primeiro passo é realizar uma consulta no gerenciador de tarefas do projeto, em busca de tarefas de correção de *bugs*, que foram concluídas dentro do período de interesse;
2) **Busca pelos *logs* das tarefas de correção**: localizadas as tarefas de correção de *bugs*, o segundo passo é realizar uma nova consulta ao gerenciador de tarefas, em busca dos *logs* de cada uma das tarefas encontradas no passo anterior;
3) **Busca pelos artefatos alterados nos *logs***: uma vez localizados os *logs* das tarefas de correção de erro, o terceiro passo é identificar quais artefatos precisaram ser alterados para que o *bug* fosse corrigido. Artefatos que não tenham a extensão ".java" são desconsiderados nesse passo;
4) **Calcular os *diffs***: o quarto passo é calcular os *diffs* para determinar quais linhas de código precisaram ser alteradas para que a tarefa fosse concluída;
5) **Executar os *blames***: uma vez identificadas as linhas de código que precisaram ser alteradas em cada artefato, o quinto passo é executar *blames* nas linhas que

precisaram ser modificadas, a fim de descobrir quem foi o último desenvolvedor a manipular tais linhas que precisaram ser corrigidas;

6) **Calcular a métrica**: a contagem da métrica ocorre por revisão, e não por linha de código. Portanto, supondo que um desenvolvedor introduziu trinta linhas de código que precisaram ser corrigidas posteriormente, mas todas essas linhas foram introduzidas num único *commit*, a métrica contará que ele introduziu apenas um *bug*, e não trinta.

A estratégia de mineração da segunda métrica, (ii) Falha em testes, é descrita a seguir.

1) **Buscar por *logs* que representem falha em testes**: o primeiro passo é consultar, no gerenciador de tarefas do projeto, por *logs* que representem falha em testes, e as tarefas em que esses *logs* ocorreram;
2) **Buscar por *logs* de desenvolvimento[10]**: o segundo passo é consultar, no gerenciador de tarefas do projeto, por *logs* de desenvolvimento posteriores ao *log* de erro de testes;
3) **Busca pelos artefatos alterados nos *logs***: o terceiro passo é identificar quais artefatos precisaram ser alterados para que o teste passasse. Artefatos que não tenham a extensão ".java" são desconsiderados nesse passo;
4) **Calcular os *diffs***: o quarto passo é calcular os diffs para determinar quais linhas de código precisaram ser alteradas para que o teste passasse;
5) **Executar os *blames***: uma vez identificadas as linhas de código que precisaram ser alteradas em cada artefato, o quinto passo é executar *blames* nas linhas que precisaram ser modificadas, a fim de descobrir quem foi o último desenvolvedor a manipular tais linhas que precisaram ser corrigidas;
6) **Calcular a métrica**: assim como na métrica anterior, a contagem da métrica ocorre por revisão, e não por linha de código.

Apesar dessas métricas apresentarem apenas a quantidade de erros introduzidos pelo desenvolvedor, a mineração registra várias outras informações, tais como: as linhas que

[10] Um *log* é um registro de esforço realizado para conclusão de uma tarefa. Um *log* de desenvolvimento representa um esforço de codificação visando a conclusão da tarefa.

precisaram ser corrigidas, o número de cada linha, o artefato onde elas se encontram, a revisão, a data que foram manipuladas, etc. Vale salientar também que essas duas métricas não filtram suas consultas buscando apenas por *logs* que foram realizados pelos desenvolvedores da lista de interesse (parâmetro de entrada). Tal filtro é realizado pelo formatador da mineração, que formata os valores das métricas apenas para os desenvolvedores constantes na lista.

3.3.3.3 Ameaças Conhecidas

Uma ameaça conhecida das métricas de introdução de *bugs* se deve a forma como elas identificam o desenvolvedor que introduziu o *bug*. As duas métricas discutidas na seção anterior consideram que o último desenvolvedor a manipular o código-fonte antes da correção é aquele que introduziu o *bug*. Contudo, é possível que isso nem sempre seja verdade, pois tudo vai depender de como e onde a correção foi realizada. Portanto essas métricas podem resultar em alguns falso-positivos.

3.3.4 Contribuição em Correção de *Bugs*

Outro atributo que pode ser avaliado pelo gerente de software é a contribuição de cada desenvolvedor na correção de *bugs*. Esse atributo de contribuição é medido através de uma métrica:

(i) **Quantidade de *logs* de desenvolvimento em correção de *bugs***: quantidade de *logs* de desenvolvimento em tarefas de correção de *bugs* de cada desenvolvedor, durante o período de interesse, dividido pela quantidade de *logs* de desenvolvimento em tarefas de correção de erro dos desenvolvedores de sua equipe, durante o período de interesse.

3.3.4.1 Interpretação da Métrica

A referida métrica é calculada através de uma razão entre a contribuição do desenvolvedor e a contribuição de sua equipe em tarefas de correção de erro. Multiplicando o resultado dessa divisão por cem, tem-se medida percentual da participação do desenvolvedor nesse atributo de contribuição. Quanto maior for o valor percentual, maior foi a participação do desenvolvedor na correção de *bugs*, em relação aos outros membros de sua equipe.

3.3.4.2 Estratégia de mineração

A estratégia de mineração dessa métrica consiste em uma consulta no gerenciador de tarefas do projeto, em busca de *logs* de desenvolvimento em tarefas de correção de erros. A consulta inclui um filtro para buscar apenas por *logs* realizados durante o período de interesse. Vale salientar que a mineração em si não filtra os *logs* pela lista de desenvolvedores fornecida através dos parâmetros de entrada. Tal filtro é realizado pelo formatador da mineração, que formata os valores da métrica apenas para desenvolvedores desejados.

3.3.4.3 Ameaças Conhecidas

Dependendo da dinâmica de trabalho de cada desenvolvedor, a estratégia de mineração adotada pode favorecer desenvolvedores que cadastram *logs* de desenvolvimento a cada pequeno incremento da tarefa, e desfavorecer aqueles que deixam para cadastrar um único *log* que conclui a tarefa inteiramente.

3.4 Sumário

Este capítulo apresentou a abordagem proposta por esta dissertação para análise da contribuição individual dos desenvolvedores. Foi apresentada uma visão geral da abordagem, os seus principais componentes, tais como as métricas e as minerações, a técnica de definição de valores de referência e a técnica de identificação e descarte de *outliers*. Foram apresentados também os detalhes de implementação de cada mineração, bem como orientações para interpretação de cada métrica e as ameaças previamente conhecidas.

4 ESTUDO DE AVALIAÇÃO DA ABORDAGEM DE ACOMPANHAMENTO INDIVIDUAL

Este capítulo apresenta o estudo de avaliação da utilidade e aplicabilidade da abordagem apresentada no capítulo anterior, dentro de um contexto industrial de desenvolvimento de software.

4.1 Contexto do Estudo

O estudo de avaliação foi realizado na Superintendência de Informática (SINFO, 2014a), órgão da Universidade Federal do Rio Grande do Norte[11] (UFRN), que é responsável por coordenar atividades de administração, projeto e desenvolvimento de sistemas da universidade, bem como pelo gerenciamento de toda a infraestrutura de rede da UFRN.

Entre os sistemas desenvolvidos e mantidos pela SINFO, pode-se destacar alguns: (i) Sistema Integrado de Patrimônio, Administração e Contratos (SIPAC); (ii) Sistema Integrado de Gestão de Atividades Acadêmicas (SIGAA); (iii) Sistema Integrado de Gestão de Planejamento e de Recursos Humanos (SIGRH); (iv) Sistema de Administração dos Sistemas (SIGAdmin); (v) Sistema Integrado de Gestão Eletrônica de Documentos (SIGED); (vi) Sistema Integrado de Gerência de Projetos (iProject). Além desses sistemas, a SINFO ainda possui uma equipe que desenvolve um projeto chamado Sucupira (SINFO, 2014b). Essa plataforma coleta informações, realiza análises e avaliações para formar a base de referência do Sistema Nacional de Pós-Graduação (SNPG) da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior[12] (CAPES).

Os sistemas desenvolvidos e mantidos pela SINFO são usados por vinte universidades, entre federais e particulares, bem como por cinco institutos federais de educação. Os sistemas também são usados por outros nove órgãos federais: (i) Agência Brasileira de Inteligência (ABIN); (ii) Controladoria Geral da União (CGU); (iii) Departamento da Polícia Federal (DPF); (iv) Departamento da Polícia Rodoviária Federal (DPRF); (v) Fundo Nacional de Desenvolvimento da Educação (FNDE); (vi) Instituto Nacional de Seguridade Social (INSS);

[11] http://www.ufrn.br
[12] http://www.capes.gov.br/

(vii) Ministério da Cultura (MinC); (viii) Ministério da Justiça (MJ); (ix) Ministério do Planejamento, Orçamento e Gestão (MPOG).

Os sistemas desenvolvidos pela SINFO são gerenciados através de uma ferramenta de coordenação de tarefas, denominada iProject[13]. Ela é uma ferramenta de gerenciamento de processo de desenvolvimento bastante completa, que contempla atividades de desenvolvimento, testes, validação, suporte e integração.

Nosso estudo de avaliação da abordagem utilizou dados de cinco sistemas: SIPAC, SIGAA, SIGRH, o Sucupira e o iProject. Ele foi conduzido com a participação de alguns gerentes de software da SINFO. Entre as diversas atribuições desses gerentes, uma delas é determinar, mensalmente, a produtividade de cada membro de sua equipe. A definição dessa produtividade é importante não só para o gerente acompanhar a sua equipe, mas também porque ela define um bônus financeiro no salário do funcionário. Dessa forma, o gerente de software de cada equipe tem a responsabilidade de definir a produtividade dos membros de sua equipe de forma justa, estando atento às diversas formas de participação que seus funcionários possam exercer no projeto.

4.2 Planejamento

O planejamento desse estudo de avaliação começou cerca de 45 dias antes de sua execução. Contudo, a adaptação das minerações para se adequar à dinâmica de trabalho da SINFO foi uma atividade realizada meses antes desse planejamento. Cerca de oito meses antes do planejamento desse estudo começar ser realizado, houve alguns esforços para que esse estudo pudesse ser executado, tais como: (i) compreender a dinâmica de trabalho das equipes da SINFO; (ii) compreender como os dados foram estruturados no iProject para que as minerações pudessem consultá-los; e (iii) adequar a implementação das minerações para atender a esse contexto específico.

A compreensão da dinâmica de trabalho das equipes da SINFO envolveu algumas conversas com desenvolvedores de algumas equipes, e a observação do iProject para compreensão sobre como as tarefas evoluíam. A compreensão dos dados fornecidos pelo

[13] http://www.iproject.ufrn.br/

iProject envolveu a análise da estrutura de tabelas e relacionamentos do banco de dados desse sistema. A adequação da implementação das minerações para atender à dinâmica específica das equipes de desenvolvimento da SINFO demandou algum tempo de codificação e muitos testes realizados para a checagem dos resultados apresentados pelas métricas.

Durante o planejamento propriamente dito do estudo de avaliação, inicialmente definiram-se os objetivos a serem atingidos, as questões de pesquisa a serem respondidas e o formato do estudo a ser conduzido. Numa reunião periódica com os gestores da SINFO, comunicamos o interesse de realizar, em breve, um estudo de caso com a participação dos gerentes de software. Em resposta, os gestores disseram estar disponíveis e poder contar com a participação de suas equipes.

O formato do estudo escolhido foi uma avaliação da abordagem através de entrevistas semi-estruturadas com os gerentes de software da SINFO. Em seguida iniciou-se a elaboração do roteiro das entrevistas, que foi tema de discussão de algumas reuniões entre o autor desta dissertação, prof. Dr. Uirá Kulesza (orientador), prof. Dr. Fernando Figueira (co-orientador) e Christoph Treude, PhD[14]. As perguntas do roteiro da entrevista foram elaboradas visando responder as questões de pesquisa do estudo e, portanto, buscando atingir os objetivos definidos para o estudo.

No dia 05/Maio/2014, alguns dias antes da execução do estudo, foi realizada uma reunião de abertura com a participação de vários gerentes da SINFO, do autor desta dissertação e do prof. Dr. Fernando Figueira. O encontro ocorreu na sala de reuniões do prédio sede da SINFO. Durante essa reunião foram apresentados os pesquisadores que faziam parte do estudo a ser conduzido e os objetivos do projeto. Por fim, os gerentes foram convidados para participar do estudo sob a garantia de que todos os dados colhidos pelo estudo tinham apenas propósitos acadêmicos, e de que a identidade de cada participante seria protegida. Ao todo, sete gerentes concordaram em participar do estudo. Todas as entrevistas foram realizadas em horário e local conforme apresentados na Tabela 5.

[14] http://ctreude.ca/

Tabela 5 - Data, hora, local, cargo e codinomes dos entrevistados

Data e hora	Local	Cargo	Codinomes
08/05/2014 as 08h30	Prédio anexo	Gerente de desenvolvimento	GD4
08/05/2014 as 10h30	Prédio anexo	Gerente de desenvolvimento	GD1
08/05/2014 as 14h30	Prédio anexo	Gerente de testes	GT
08/05/2014 as 15h30	Prédio anexo	Gerente de suporte	GS
09/05/2014 as 14h30	Prédio sede	Gerente de desenvolvimento	GD3
09/05/2014 as 15h30	Prédio sede	Gerente de requisitos	GR
16/05/2014 as 08h30	Prédio anexo	Gerente de desenvolvimento	GD2

4.3 Participantes

O estudo de avaliação contou com a participação de sete gerentes de equipes distintas da SINFO, sendo quatro deles gerentes de desenvolvimento do SIPAC, SIGAA, SIGRH e Sucupira; bem como um gerente de requisitos, um gerente de suporte e um gerente de testes. Com o objetivo de enriquecer a diversidade dos dados colhidos pelo presente estudo, foram convidados gerentes de outras áreas, como testes, requisitos e suporte. Essa estratégia possibilitou agregar diferentes visões aos resultados.

Conforme definido com os gerentes durante a reunião de abertura do estudo, todas as respostas às questões das entrevistas foram publicadas de forma anônima, a fim de garantir a confidencialidade da identidade dos participantes. Assim, para se referir aos gerentes de software entrevistados, são usados os codinomes apresentados na Tabela 5.

4.4 Objetivos

O objetivo geral do nosso estudo foi avaliar a aplicabilidade e utilidade da abordagem num contexto de produção industrial de software, para analisar se ela oferece meios para que os gerentes de software acompanhem suas respectivas equipes, bem como dar suporte à tomada de decisão gerencial. O estudo foi conduzido através de entrevistas individuais semi-estruturadas com os gerentes de cada equipe e buscando atingir os seguintes objetivos:

(i) Compreender como a avaliação da contribuição dos desenvolvedores é realizada atualmente;

(ii) Compreender como os gerentes acreditam que a avaliação da contribuição dos desenvolvedores pode melhorar;

(iii) Avaliar a utilidade e aplicabilidade da abordagem discutida anteriormente para analisar a contribuição individual dos desenvolvedores.

4.5 Questões de Pesquisa

As entrevistas realizadas com os gerentes objetivaram responder três questões de pesquisa, que estão alinhadas com os objetivos do estudo.

QP1) Como a avaliação da contribuição dos desenvolvedores é feita atualmente? Essa questão de pesquisa busca compreender como os gerentes avaliam atualmente seus desenvolvedores para determinar como a abordagem pode contribuir para essa avaliação;

QP2) Como a avaliação da contribuição dos desenvolvedores poderia ser melhorada? Essa questão de pesquisa visa compreender se haveria outras formas de avaliação e acompanhamento que os gerentes julgam interessantes. Esta compreensão é importante para determinar se a abordagem poderia contribuir para essas novas perspectivas de avaliação;

QP3) Quão útil é a abordagem para avaliar a contribuição individual dos desenvolvedores? Essa questão visa compreender como e o quão útil é a abordagem para os gerentes de software acompanharem e avaliarem seus desenvolvedores.

4.6 Execução da Abordagem de Acompanhamento

Os passos da abordagem de análise de contribuição de desenvolvedores, ilustrados na Figura 5, foram executados antes da realização das entrevistas, com a finalidade de coletar e processar os dados que produzam as métricas que foram usadas durante as entrevistas com os gerentes de software.

Contudo, antes de se executar os passos de abordagem, foi necessário definir como as métricas seriam utilizadas e apresentadas para os gerentes entrevistados. As métricas de linhas de código adicionadas, modificadas e removidas foram somadas para a produção de um único indicador de volume de contribuição. As duas métricas relativas a Introdução de Bugs também foram somadas para a produção de um único indicador que avalia esse atributo. As outras métricas foram usadas conforme apresentadas na Seção 3.3. Conforme definido com a equipe de gerência da SINFO, quatro gerentes de equipes de desenvolvimento participaram do

estudo. Houve, portanto, a necessidade de mineração dos dados de quatro projetos distintos: SIGAA, SIPAC, SIGRH e Sucupira.

As minerações foram executadas colhendo dados entre 03 de fevereiro de 2014 até 25 de abril de 2014, totalizando 12 semanas para serem analisadas. Formatadores foram executados para exportar os dados colhidos para arquivos de texto no formato CSV, que posteriormente foram processados para a produção de planilhas do Microsoft Excel, com os dados compilados e organizados. Em seguida, iniciou-se a etapa de identificação e descarte de *outliers*, através dos passos discutidos na Seção 3.2.2, e concluiu-se que, para o período analisado, nenhum *outlier* foi identificado.

O passo seguinte foi a definição dos valores de referência das métricas, através da técnica que compõe a abordagem proposta nesta dissertação. A técnica permite ao usuário escolher a quantidade de valores de referência a serem definidos, a fim de atender às necessidades específicas de cada projeto. Para o presente estudo de avaliação, escolheu-se definir três valores de referência para cada métrica, permitindo que os dados sejam classificados em quatro grupos. A definição dos valores de referência foi realizada exclusivamente pelo autor desta dissertação, não havendo, portanto, a participação dos gerentes de software nessa etapa.

Os passos discutidos na Seção 3.2.3 foram executados, a fim de produzir os gráficos de dispersão que permitem a definição dos valores de referência. Tais gráficos de dispersão são apresentados no Apêndice B. A Tabela 6, Tabela 7, Tabela 8 e Tabela 9 apresenta os valores de referência definidos para a equipe do SIGAA, SIGRH, SIPAC e Sucupira, respectivamente. Conforme apresentado nas tabelas a seguir, foram definidos três valores de referência para cada métrica, que foram determinados pelo autor desta dissertação sem a participação dos gerentes de software. Conforme discutido nas seções de interpretação das métricas da Seção 3.3, os valores de referência são usados de diferentes formas entre as métricas. Nas métricas de Introdução de *Bugs* e Complexidade Média por Método, quanto menores os valores, melhor. Nas métricas de Correção de Erros e Volume de Contribuição, quanto maior o valor, melhor.

Tabela 6 - Valores de referência para a equipe do SIGAA

Métricas	**Valores de referência**		
	VR1	**VR2**	**VR3**

Introdução de *bugs*		23	84	229
Complexidade média por método	**Adicionados**	1,34	2,00	2,75
	Modificados	0,59	1,21	2,33
Correção de erros		10%	25%	43%
Volume de contribuição		110	459	937

Tabela 7 - Valores de referência da equipe do SIGRH

Métricas		**Valores de referência**		
		VR1	**VR2**	**VR3**
Introdução de *bugs*		20	52	125
Complexidade média por método	**Adicionados**	1,40	1,83	3,19
	Modificados	0,66	1,88	3,67
Correção de erros		6%	12%	23%
Volume de contribuição		100	221	584

Tabela 8 - Valores de referência da equipe do SIPAC

Métricas		**Valores de referência**		
		VR1	**VR2**	**VR3**
Introdução de *bugs*		12	45	137
Complexidade média por método	**Adicionados**	1,21	2,00	3,74
	Modificados	0,09	0,17	0,75
Correção de erros		3%	7%	18%
Volume de contribuição		152	491	991

Tabela 9 - Valores de referência da equipe do Sucupira

Métricas		**Valores de referência**		
		VR1	**VR2**	**VR3**
Introdução de *bugs*		5	14	27
Complexidade média por método	**Adicionados**	1,11	2,00	3,75
	Modificados	0,84	1,63	2,57
Correção de erros		9%	14%	49%
Volume de contribuição		93	242	523

4.7 Coleta de Dados

As entrevistas foram realizadas individualmente e em local reservado, sem a interferência de outras pessoas. O tempo de duração das entrevistas variou razoavelmente, mas em média duraram uma hora, conforme consta na Tabela 10.

Tabela 10 - Tempo de duração das entrevistas

Participante	**Duração das entrevistas**
GD1	01h 10min 46s
GS	00h 57min 26s
GT	00h 50min 22s
GD2	00h 55min 34s
GD3	00h 48min 42s
GD4	01h 17min 12s
GR	01h 05min 51s
Tempo total	**07h 05min 53s**
Média	**01h 00min 50s**

Todas as entrevistas foram devidamente gravadas utilizando um software de gravação no *smartphone*, posicionado entre o entrevistador e o entrevistado para capturar o áudio claramente. Além disso, o Camtasia Studio 8[15] também foi utilizado para gravar a conversa durante a entrevista através do microfone do notebook. Essa segunda forma de gravação foi utilizada para garantir que a entrevista não fosse perdida caso houvesse algum problema com a gravação no *smartphone*.

O Camtasia Studio 8 também foi utilizado para gravar as ações realizadas no notebook em que os dados do estudo (valores das métricas, valores de referência, etc.) estavam sendo apresentados para os entrevistados. Isso é importante porque se fosse realizada a gravação apenas do áudio, a entrevista poderia ficar confusa quando o entrevistador ou o entrevistado apontasse para alguma informação apresentada na tela do computador.

[15] http://www.techsmith.com/camtasia.html

4.7.1 Roteiro da Entrevista

Inicialmente foi esclarecido para o entrevistado que as informações colhidas através das entrevistas têm apenas propósitos acadêmicos e, portanto, os dados nela revelados não são usados para avaliá-los profissionalmente, ou avaliar os seus respectivos desenvolvedores. Também assegura-se que tudo será publicado de forma anônima. As questões da entrevista são apresentadas no Apêndice A e estão divididas em cinco partes que serão explicadas a seguir.

A primeira parte da entrevista era composta por perguntas que visaram a identificação do perfil do participante. Tais questões objetivaram, principalmente, determinar a experiência do gerente no exercício de tal cargo de gestão. A segunda parte era composta por questões que buscavam compreender como a avaliação da contribuição dos desenvolvedores era feita atualmente. A terceira parte era composta por questões que objetivaram compreender como a avaliação poderia ser melhorada.

A quarta parte da entrevista visava determinar a utilidade e aplicabilidade de cada uma das métricas da abordagem proposta na análise da contribuição individual dos desenvolvedores. Nesse ponto da entrevista, foi apresentado para os entrevistados uma planilha contendo dados das métricas de uma equipe específica (SIGAA, SIPAC, SIGRH ou Sucupira). Quando o entrevistado era um gerente de desenvolvimento, a planilha de sua equipe era apresentada. Quando o entrevistado era um gerente de requisitos, suporte ou testes, ele era questionado sobre qual equipe gostaria de ver os dados.

Cada planilha contém treze abas. Na primeira aba constam três valores de referência para cada métrica, definidos conforme apresentado na Seção 4.6. Tais valores foram usados pela planilha para colorir cada valor de métrica nas outras abas, usando o seguinte padrão de cores: azul quando o valor for ótimo, amarelo quando for bom, laranja quando for ruim e vermelho quando for péssimo. As outras doze abas contêm dados das métricas referentes a uma semana específica: (i) 03 a 07 de fevereiro de 2014; (ii) 10 a 14 de fevereiro de 2014; (iii) 17 a 21 de fevereiro de 2014; (iv) 24 a 28 de fevereiro de 2014; (v) 03 a 07 de março de 2014; (vi) 10 a 14 de março de 2014; (vii) 17 a 21 de março de 2014; (viii) 24 a 28 de março de 2014; (ix) 31 de março a 04 de abril de 2014; (x) 07 a 11 de abril de 2014; (xi) 14 a 18 de abril de 2014; (xii) 21 a 25 de abril de 2014.

Figura 10 – Amostra de dados apresentada aos entrevistados

Developer	Bug Commits	Complexity		Bug Fix	Volume
		Added	Changed		
A	0	1,00	2,71	0%	54
B	27	5,00	2,33	65%	62
C	85	0,00	0,00	0%	0
D	0	0,00	0,00	0%	0
E	56	1,88	2,40	18%	81
E	27	0,00	0,00	0%	0
F	7	0,00	1,13	12%	32
G	75	1,59	0,40	6%	459
H	0	0,00	0,00	0%	0
I	0	0,00	0,00	0%	0
J	0	0,00	0,00	0%	0
K	0	1,62	0,93	0%	313
L	0	0,00	0,00	0%	0

A Figura 10 ilustra os dados apresentados numa aba, com valores de métricas de uma semana específica. Os dados estão organizados da seguinte forma: na primeira coluna constam os nomes dos desenvolvedores da equipe, que foram apresentados para os gerentes durante as entrevistas, contudo, na referida figura, foram omitidos para garantir a confidencialidade. A partir da segunda coluna constam os valores das métricas para cada desenvolvedor.

A coluna *Bug Commits* apresenta dados que são resultado da soma entre as métricas de quantidade de *commits* que introduziram *bugs* e a falha em testes, ambas métricas do atributo de avaliação de introdução de bugs, conforme apresentado na Seção 3.3.3. As duas colunas seguintes apresentam dados referentes ao atributo de complexidade média em métodos adicionados e modificados, apresentadas na Seção 3.3.3. A coluna *Bug Fix* apresenta os dados da métrica de contribuições em correção de *bugs*, apresentada na Seção 3.3.4. A última coluna apresenta dados que são resultado da soma entre as três métricas de linhas de código (adicionado, modificado e removido) de volume de contribuição, apresentadas na Seção 3.3.1. Antes de questionar o gerente com as perguntas da quarta parte da entrevista, foi feita uma breve explicação sobre como cada métrica foi calculada. A quinta e última parte da entrevista buscava determinar a utilidade e aplicabilidade do conjunto das métricas, e não mais elas individualmente, conforme foi feito na quarta parte.

4.8 Análise dos Dados

Esta seção visa apresentar como foi realizada a análise qualitativa dos dados colhidos durante as entrevistas. Essa análise consistiu nos seguintes passos: (i) transcrição das entrevistas, (ii) codificação dos dados e (iii) escrita dos memorandos. Todas as etapas de análise desse estudo foram executadas pelo autor deste trabalho, sob a supervisão e revisão dos orientadores, a fim de manter a confidencialidade dos dados colhidos através das entrevistas, e para resguardar detalhes sensíveis do processo de trabalho da própria SINFO.

Nesse estudo de avaliação utilizou-se uma abordagem baseada na Teoria Fundamentada em Dados (*Grounded Theory*), que é uma metodologia de análise qualitativa para derivar teorias através de dados. Essa metodologia, que foi criada por Barley Glaser e Anselm Strauss, é largamente utilizada em ciências sociais e tem como objetivo gerar ideias teóricas novas ou hipóteses a partir da emergência de conceitos e categorias advindos dos dados (CORBIN; STRAUSS, 2008).

4.8.1 Transcrição das Entrevistas

O primeiro passo para análise dos dados foi realizar a transcrição das entrevistas gravadas. Para realizar tal tarefa, utilizou-se um software específico de transcrição, chamado Express Scribe[16] da NCH Software. Este software possui vários recursos que facilitam o processo de transcrição, mas apenas alguns deles foram utilizados nesse estudo de avaliação: (i) o controle de reprodução do áudio através do teclado, permitindo que o pesquisador possa pausar e reiniciar o áudio sem a necessidade de alternar para a janela do Express Scribe; (ii) reduzir ou aumentar a velocidade de reprodução do áudio; e (iii) realizar o pré-processamento do áudio para reduzir o ruído externo.

As escritas das transcrições foram feitas no Microsoft Word, e foram realizadas de forma fiel, ou seja, palavra por palavra. Em cada fala, seja do entrevistado ou do entrevistador, foi registrado o nome daquele que falava no momento, seguido do seu discurso. Além disso foi registrado o tempo (horas, minutos e segundos) da fala do entrevistador. Durante as transcrições, o autor desse trabalho realizou algumas anotações que poderiam ser

[16] http://www.nch.com.au/scribe/

úteis no momento da análise dos dados. Essas anotações foram feitas quando o pesquisador via a correlação entre informações dadas entre diferentes gerentes, ou outros tipos de ocorrências importantes durante as entrevistas.

Uma vez concluída cada transcrição, foi realizada uma revisão do texto, através da reprodução de todo o áudio da entrevista e acompanhando o que havia sido transcrito em documento, a fim de garantir que a transcrição havia sido realizada de forma fiel. Mesmo usando um software específico para transcrição das entrevistas, essa tarefa levou cerca de trinta dias para ser concluída. Ao final de todas as transcrições, cada documento tinha, em média, 17 páginas, e um total de 120 páginas de transcrições.

4.8.2 Codificação dos Dados

O segundo passo realizado para a análise nesse estudo de avaliação foi a codificação das entrevistas transcritas. A codificação consiste em nomear segmentos de dados com uma classificação que categoriza, resume e representa o segmento. Esse é um dos passos necessários para passarmos dos enunciados reais dados pelos entrevistados à elaboração de interpretações analíticas sobre eles.

(CORBIN; STRAUSS, 2008) dividem a codificação em três etapas:

(i) **Codificação aberta**: o texto é lido pelo pesquisador a fim de identificar categorias relevantes;
(ii) **Codificação axial**: organiza as categorias de forma hierárquica, relacionando as categorias às subcategorias. Refina as categorias, define propriedades e reagrupa os segmentos de dados codificados;
(iii) **Codificação seletiva**: organiza as categorias em torno de uma categoria principal (*core category*), que representa o fenômeno central a ser estudado.

A codificação dos dados foi realizada utilizando um software específico para análise qualitativa de dados, o MAXQDA[17] 11. Os recursos presentes nesse software facilitam bastante a execução das três etapas de codificação: (i) organização dos códigos criados através do *Code System*; (ii) visualização dos documentos a serem codificados no próprio programa;

[17] http://www.maxqda.com/

(iii) codificação de segmentos de texto; (iv) reorganização dos códigos no *Code System*; e (v) visualização dos segmentos de códigos em diferentes documentos; entre muitas outras funcionalidades.

O processo de codificação gerou a seguinte hierarquia de 38 códigos, que inclui categorias principais e subcategorias. Essa estrutura de códigos serviu para classificar 342 segmentos de dados.

- **Identificação de perfil**
  - Tempo de serviço na SINFO
  - Tempo de serviço como gerente
  - Experiência de gerência em outras empresas
- **QP1) Como a avaliação da contribuição dos desenvolvedores é feita atualmente?**
  - Características e atuação do gerente
  - Características ruins de um desenvolvedor
  - Características do bom desenvolvedor
  - Acompanhamento/avaliação
    - Critérios de avaliação/acompanhamento
    - Suporte ferramental
    - Divisão de tarefas
    - Problemas e dificuldades
    - Exemplo de definição da produtividade mensal
    - Definindo a complexidade da tarefa
- **QP2) Como a avaliação da contribuição dos desenvolvedores poderia ser melhorada?**
  - Uso de métricas
- **QP3) Quão útil é a abordagem para avaliar a contribuição individual dos desenvolvedores?**
  - Introdução de *bugs*
    - Utilidade
    - Ameaças e limitações
  - Complexidade média por método
    - Exemplos
    - Ameaças e limitações

        - Utilidade
    - Correção de erros
        - Ameaças e limitações
        - Utilidade
    - Volume de contribuição
        - Exemplos
        - Ameaças e limitações
        - Utilidade
    - Métricas gerais
        - Limitações
        - Benefícios
        - Ameaças
        - Acompanhamento
- **Notas do entrevistador**

A primeira categoria visou agrupar os códigos e segmentos relativos a identificação de perfil do gerente que estava sendo entrevistado. As três categorias conceituais seguintes são relativas às questões de pesquisa do estudo, em que seus códigos e segmentos podem contribuir para respondê-las. Por fim, um último código foi criado (Notas do entrevistador), visando agrupar as notas que foram feitas pelo entrevistador durante e após as entrevistas, a fim de facilitar a posterior análise dos códigos para a escrita dos memorandos.

4.8.3 Escrita dos Memorandos

A escrita dos memorandos é uma etapa posterior à codificação, que exige que o pesquisador analise os segmentos codificados, compreenda o que está sendo posto por eles e escreva uma análise sobre os segmentos.

Para facilitar a análise dos segmentos codificados e a escrita dos memorandos, utilizou-se uma abordagem *bottom-up*, em que se analisou primeiro os segmentos dos códigos-filho, e uma vez analisados todos eles, analisou-se os códigos-pais considerando não apenas os segmentos presentes neles, mas também os memorandos dos seus respectivos códigos-filho. Essa análise *bottom-up* foi realizada até chegar nas quatro categorias conceituais.

4.9 Resultados

A presente seção mostra os memorandos escritos durante a análise dos códigos e de seus respectivos segmentos, conforme explicado na Seção 4.8.3.

4.9.1 Perfil dos Participantes

Os códigos referentes a identificação do perfil permitiram definir os dados demográficos dos participantes da pesquisa, conforme apresentados na Tabela 11.

Tabela 11 - Dados demográficos dos participantes

Participante	Tempo de serviço na SINFO	Tempo de serviço como gerente	Experiência de gerência em outras empresas
GD1	6 anos	3 anos	-
GS	14 anos	7 anos	2 anos
GT	19 anos	6 anos	1 ano e 9 meses
GD2	5 anos	1 ano	1 ano e 6 meses
GD3	8 anos	2 anos	-
GD4	8 anos	2 anos	-
GR	5 anos	4 anos	3 anos
Total	**65 anos**	**25 anos**	**8,2 anos**
Média	**9,28 anos**	**3,57 anos**	**1,17 ano**

Nem todos os gerentes participantes tinham experiência de gerência em outras empresas ou instituições, portanto não constam seus tempos de experiência na quarta coluna.

4.9.2 QP1) Como a avaliação da contribuição dos desenvolvedores é feita atualmente?

Antes da discussão sobre como os gerentes de software avaliam seus desenvolvedores, é importante discutir sobre o que esses gerentes consideram um bom desenvolvedor, ou seja, quais são suas características, suas qualidades e como eles atuam no projeto. A compreensão sobre essas características é importante para avaliar se e como a abordagem pode auxiliar o gerente de software no acompanhamento dos desenvolvedores.

Qualidades do desenvolvedor de software

A Tabela 12 apresenta, de forma resumida, as qualidades desejáveis que os gerentes indicaram, ordenada pela quantidade de menções recebidas por cada qualidade. A primeira qualidade, comentada por todos os gerentes entrevistados, se refere à capacidade de comunicação do desenvolvedor, e não apenas à capacidade de se expressar através da fala,

mas também através da escrita. Os gerentes de desenvolvimento mencionaram essa qualidade quando disseram que é importante um desenvolvedor pedir ajuda de outros quando tiver alguma dúvida: "*Comunicação é importante, até porque a pessoa pode não ter conhecimento técnico, ou precisar de uma ajuda, eu considero um ponto negativo ela gastar muito tempo, solicitar uma ajuda talvez ajudasse mais de alguém que tenha mais experiência, ou do coordenador.*" (GD1).

Tabela 12 - Qualidades dos desenvolvedores

Qualidade	**Quantidade de menções**
Comunicação	7
Conhecimento técnico	5
Proatividade	4
Conhecimento negocial	4
Compromisso com os prazos	3
Análise da tarefa antes de implementá-la	3
Experiência	2
Auxiliar outros desenvolvedores	2
Capacitar-se	1
Código com baixa taxa de erros	1
Cumprimento dos horários	1
Realiza testes	1

O gerente de testes mencionou essa característica quando relatou que um bom desenvolvedor deve discriminar em detalhes o que foi realizado na tarefa, para que a equipe de testes possa realizar testes mais precisos e adequados: *"[...] o desenvolvedor é bom quando? Quando ele solicita teste, coloca o log dizendo as informações sobre como testar e o que testar. Então, pra a gente no caso, o bom desenvolvedor é aquele que coloca tudo direitinho... [...] Discrimina tudo direitinho, diz como é que é para testar, o que está faltando, utiliza isso aqui, coloca todos os caminhos, os casos de uso, tá entendendo? Ele coloca também o changelog, que é a mudança que está sendo feita*" (GT).

O gerente de requisitos relatou que um bom desenvolvedor deve ter a capacidade de descrever a implementação, através de uma documentação ou através de comentários no próprio código, para dar apoio à futura manutenção do código: "*Então você fazer uma boa documentação, você ter uma descrição [...] Mas aí se você faz uma documentação, mesmo em código ali, daquilo que você está trabalhando, para deixar claro também pra um próximo que venha a pegar aquele código, trabalhar nele sem problema*" (GR).

A segunda qualidade mais comentada é o conhecimento técnico. Cinco dos sete gerentes entrevistados citaram essa característica como sendo desejável para um bom desenvolvedor de software. Esse conhecimento compreende não apenas a linguagem de programação utilizada, mas também os padrões utilizados no projeto, os testes, a arquitetura do sistema, etc. "*O cara tem que ter conhecimento técnico da tecnologia que é usada aqui, da arquitetura, de como os casos de uso funcionam.*" (GD1).

A terceira característica desejável num bom desenvolvedor, comentada por quatro dos sete gerentes, é a proatividade. Essa característica diz respeito ao empenho empregado pelo desenvolvedor para resolver as demandas sob sua responsabilidade, sem a necessidade de cobrança por parte do gerente da equipe ou de fatores externos. "*E assim, isso é uma característica que, pelo menos, eu tento fazer com que o pessoal tenha, que é a questão da pró-atividade, então você vê que vai dar um problema em determinado lugar, você chega, tenta resolver, ou tenta conversar com alguém que possa auxiliar você a tentar resolver aquele problema para que não aconteça nenhum problema no sistema, ou na funcionalidade que você está desenvolvendo.*" (GD3).

Não menos importante do que a proatividade está o conhecimento negocial, que foi mencionado entre quatro dos sete gerentes entrevistados. Segundo eles, essa característica diz respeito ao conhecimento dos conceitos específicos do contexto do sistema, que é adquirido com a experiência.

Mencionado por três entre os sete gerentes está o compromisso do desenvolvedor em cumprir os prazos estabelecidos nas tarefas. É possível perceber, nos discursos dos gerentes, que essa qualidade está ligada a proatividade, pois um desenvolvedor precisa ter empenho em buscar o que é necessário para resolver as suas tarefas e não precisar ser cobrado pelo gerente. Caso ele não seja capaz de cumprir seus prazos, que apresente argumentos para justificar o atraso. "*Eu acho que um bom desenvolvedor, ele além de ser um cara pontual e comprometido, ele tem que ser um cara que ele tenha responsabilidade com suas tarefas. O que eu quero dizer com você? Eu não preciso passar uma tarefa pra você com um deadline e ter que ficar lhe cobrando. O bom desenvolvedor tem que ter a característica de se cobrar, não tem o deadline lá?*" (GD4).

Analisar e compreender bem a solicitação da tarefa antes de implementá-la também foi dada como uma importante qualidade aos desenvolvedores (GS, GD3 e GR). Essa característica envolve a leitura e compreensão da solicitação, analisar o impacto que a

implementação da tarefa poderá causar, analisar a melhor forma de implementar, etc. "*Eu acho que um bom desenvolvedor é aquele que pensa, que analisa um problema, então se você recebe uma tarefa, e você analisa aquilo, sabe se vai impactar em outros lugares, não só também o tempo de execução daquela tarefa, não só isso é importante, pelo menos na minha visão, porque as vezes não adianta você fazer uma coisa muito rápido e acabar impactando em outras coisas, e deixar de funcionar alguma coisa que estava funcionando anteriormente.*" (GD3).

Outras características também foram dadas como desejáveis, como por exemplo experiência (GD1 e GD2), auxiliar outros desenvolvedores da equipe (GD1 e GD4), capacitar-se (GD2), produzir código com baixa incidência de erros (GD1), cumprir os horários (GD4) e realizar testes antes de enviar a tarefa para a equipe de testes (GT).

Deficiências do desenvolvedor de software

Quando questionados sobre a qualidades de seus desenvolvedores, alguns gerentes de software apresentaram algumas deficiências. Entre elas, pode-se destacar: descumprimento dos horários (GD1), ocorrência de algum problema em produção ocasionado por erro do desenvolvedor (GD1), problemas de relacionamento entre os membros da equipe (GD4), atraso injustificado de tarefas (GD4), não realizar os mais triviais testes de software (GT), e novamente a falta de comunicação foi mencionada: "*uma falha que acontece muito nos desenvolvedores é a falta de comunicação, muitos nem vão atrás caso tenham dúvidas, seria muito mais rápido ele falar com alguém, mas ele prefere pesquisar, ir atrás, então acho que falta de comunicação é um problema.*" (GD2).

Quantidade de tarefas resolvidas *versus* complexidade subjetiva das tarefas

Para todos os quatro gerentes de desenvolvimento entrevistados, a avaliação dos desenvolvedores é realizada, principalmente, através do quantitativo de tarefas resolvidas por cada desenvolvedor, mas levando-se em consideração a complexidade dessas tarefas. A complexidade da tarefa é definida pelo gerente da equipe, normalmente antes de se atribuir a tarefa a um desenvolvedor, e é realizada de forma subjetiva. Algumas informações da tarefa servem como indicadores para os gerentes avaliarem a complexidade dela, tais como:

(i) **Título e descrição da tarefa**: pela experiência adquirida ao longo dos anos, os gerentes alegam que, através da leitura da descrição da tarefa, eles são capazes

de determinar se a implementação da tarefa irá impactar em várias classes, ou se irá alterar procedimentos de cálculos complexos;

(ii) **Tipo da tarefa**: alguns tipos de tarefas específicos, como integração de *builds* e ajustes ou correções em cálculos orçamentários, são considerados mais complexos do que outros;

(iii) **Natureza do módulo**: o módulo onde a tarefa será implementada também auxilia o gerente de software a definir a complexidade da tarefa.

Além disso, alguns gerentes (GD1, GD2 e GD4) alegaram realizar a avaliação da complexidade da tarefa após a conclusão da mesma, a fim de auxiliar no julgamento da produtividade mensal do desenvolvedor. Os critérios estabelecidos por eles para essa definição: (i) quantidade de vezes que a tarefa voltou de testes; (ii) tempo para conclusão da tarefa; (iii) registro de *logs* do desenvolvedor na tarefa relatando o que foi realizado; (iv) quantidade de *commits* realizados; (v) quantidade de artefatos alterados nos *commits*.

Outras considerações

Todos os gerentes também relataram que avaliam outras atividades e levam em consideração outros aspectos no momento da definição da produtividade de cada um, tais como: (i) cumprimento da carga-horária e pontualidade; (ii) cumprimento dos prazos das tarefas; (iii) atividades de integração; (iv) participação em reuniões, videoconferências e treinamentos; (v) ter bom relacionamento com os outros membros da equipe; (vi) se o desenvolvedor provocou algum problema em produção; (vii) suporte a outros desenvolvedores da equipe.

Um dos gerentes de software (GD4) tomou a iniciativa de levar a planilha com a definição da produtividade dos desenvolvedores de um mês específico, e explicou como ele definia os valores através de exemplos. Além dos critérios já explicados anteriormente, ele enfatizou que, ao definir a produtividade de um desenvolvedor, ele compara as tarefas desse desenvolvedor com as tarefas de outros desenvolvedores do mesmo nível (Programador, Analista I, Analista II, etc.). Ele explicou que os desenvolvedores do mesmo nível precisam estar no mesmo patamar de produção, levando em consideração não apenas a quantidade de tarefas resolvidas, mas também a complexidade de cada uma.

Suporte ferramental

Todos os gerentes entrevistados realizam o acompanhamento e avaliação dos seus desenvolvedores através do iProject. Nele, os gerentes conseguem emitir relatórios de tarefas trabalhadas por desenvolvedor. Além disso, dentro do iProject há um recurso chamado Cronograma. Cada desenvolvedor possui o seu cronograma, que é um espaço onde ele próprio ou o gerente da equipe podem alocar tarefas para serem realizadas pelo desenvolvedor. O cronograma pode ser semanal, quinzenal ou mensal.

Contudo, algumas outras ferramentas foram mencionadas, como o Google Drive, onde eles mantêm uma planilha, chamada Planilha de Demandas, compartilhada entre as equipes de desenvolvimento e a equipe de requisitos.

Outro gerente (GD3) relatou que usa o Microsoft Project para fazer uma organização das entregas a serem feitas, bem como os subsistemas do projeto e seus respectivos casos de uso.

Um gerente (GD4) informou que utiliza o quantitativo de *commits* do sistema de controle de versão para avaliar a complexidade da tarefa, e dessa forma, usar isso como indicador para definir a produtividade do desenvolvedor. Um outro gerente (GD2) disse realizar consultas SQL diretamente no banco de dados para obter alguns indicadores de quantitativo de tarefas que não eram fornecidos pelo iProject. Fora esses dois casos, os outros gerentes informaram não utilizar nenhum tipo de métrica com base em dados do sistema de controle de versão ou do iProject.

Problemas e dificuldades

Os gerentes participantes também foram questionados sobre problemas e dificuldades que eles passam ao usar a atual abordagem de avaliação. Dentre as dificuldades apresentadas pelos gerentes, pode-se destacar:

(i) Melhorar a forma como o acompanhamento das tarefas é realizado através do iProject, permitindo que o gerente associe subtarefas e tarefas macro (GD2);
(ii) Implementar um quadro Kanban no iProject para melhorar o acompanhamento das tarefas (GD1 e GD2);
(iii) Reduzir a quantidade de *status* de tarefas, pois muitas delas não são usadas e isso pode dificultar o acompanhamento delas (GD2);
(iv) A abordagem atual para definição da produtividade de cada desenvolvedor toma muito tempo do gerente da equipe (GD2 e GD3);

(v) Organizar melhor o recebimento de novas demandas através da definição de um calendário para tal fim. Atualmente as demandas aparecem a qualquer momento (GD2);

(vi) Indefinição dos critérios de avaliação dos desenvolvedores. Dessa forma, o desenvolvedor não sabe, e o gerente também não consegue esclarecer, o que o desenvolvedor precisa fazer para ganhar uma promoção (GD1 e GD4);

(vii) Dificuldade de comunicação entre o gerente e cada desenvolvedor, para que o gerente possa esclarecer sobre porque o desenvolvedor irá receber determinado valor de produtividade, e para que o desenvolvedor também possa justificar uma possível redução de seu desempenho (GD1 e GD4);

(viii) A avaliação dos desenvolvedores através do quantitativo de tarefas resolvidas no mês não considera aspectos sociais e psicológicos dos desenvolvedores (GD4);

(ix) Falta de reuniões periódicas para discussão de problemas e soluções (GR).

4.9.3 QP2) Como a avaliação da contribuição dos desenvolvedores poderia ser melhorada?

A segunda questão de pesquisa visa compreender como os gerentes acreditam que o processo de avaliação dos seus desenvolvedores pode melhorar. Essa compreensão é importante para analisar se e como a abordagem desse estudo pode contribuir para essa avaliação e acompanhamento do gerente de software junto aos seus desenvolvedores.

Questionados sobre possíveis mudanças na abordagem de acompanhamento e avaliação de seus desenvolvedores, os gerentes elencaram algumas melhorias que poderiam ser realizadas nesse processo. Dois gerentes (GD1 e GD4) alegaram que é importante definir e comunicar aos desenvolvedores os critérios de avaliação de produtividade, pois essa indefinição pode ser desestimulante para os desenvolvedores, uma vez que eles não vão saber em que precisam melhorar para pleitear uma promoção. "*Eu acho que é esclarecer realmente pra o desenvolvedor, quem está recebendo a produtividade, o que é que ele precisa pra melhorar, se ele está recebendo menos.*" (GD1).

Uma outra sugestão de melhoria seria permitir que o gerente e o desenvolvedor possam estipular e registrar na própria tarefa a previsão de conclusão dela, bem como a sua complexidade (GD2). Isso remete a uma outra observação, feita por GR, que é permitir que o desenvolvedor realize uma análise completa da tarefa e de seus impactos antes de

implementar a tarefa. Esse mesmo gerente afirma ainda que, devido a quantidade de demandas simultâneas que surgem, é comum os desenvolvedores não terem tempo hábil para realizar essa análise de forma adequada.

A implementação de testes funcionais automatizados também foi uma outra sugestão de melhoria que surgiu, e que já começou a ser desenvolvida, contudo ainda não foi implantada devido a quantidade de demandas existentes (GT). Dois gerentes entrevistados, apesar de reconhecerem que a abordagem de avaliação sempre pode melhorar, não conseguiram imaginar quais seriam essas melhorias.

Métricas

Os gerentes entrevistados foram questionados sobre métricas que seriam úteis para o acompanhamento e a avaliação de produtividade de seus desenvolvedores. Dois gerentes (GS e GD2) disseram que métricas temporais, relativas às tarefas, que permitam o melhor acompanhamento delas por parte dos gerentes de software, poderiam ser úteis. A ideia é que essas métricas quantifiquem o tempo que cada tarefa passa em cada fase do processo.

Outra sugestão dada (GD1 e GD2) foi de considerar, no julgamento da produtividade dos desenvolvedores, a quantidade de erros de teste causados por eles. Alguns gerentes informaram que algumas tarefas são mal testadas pelos desenvolvedores, e isso muitas vezes causa atraso na conclusão da tarefa, tendo em vista que, uma vez que ela não foi aprovada pelos testes realizados pela equipe de testes, ela precisa voltar para desenvolvimento. Um maior rigor nos testes dos desenvolvedores poderia evitar esse problema. *"[...] tem um desenvolvedor que entrou agora que está dando conta do recado, e assim está tendo baixo nível de percentual de erros nas tarefas que ele manda pra teste. Então assim poderia considerar também no histórico da tarefa... [...] quantos retornos de erro foram contabilizados pra poder determinar a produtividade de alguém.*" (GD1).

Um dos gerentes de desenvolvimento (GD3) fez duas interessantes observações: a primeira delas é que ele gostaria de utilizar pontos de função para definir o esforço para realização de cada tarefa. "*Gostaria muito de fazer uma análise por ponto de função, que eu acho que é uma métrica, assim, não é a ideal mas é uma métrica que é usada no mundo todo, e principalmente no sistema brasileiro.*" (GD3). Independente da análise sugerida pelo gerente, é importante notar seu interesse por uma métrica que quantifique esforço de desenvolvimento.

A segunda observação feita por esse gerente de desenvolvimento foi informar o uso de uma ferramenta chamada Sonar[18], que é uma plataforma livre que levanta alguns indicadores sobre o sistema em desenvolvimento. Entre esses indicadores, está um que quantifica a complexidade ciclomática dos componentes, inclusive criando um *ranking* daqueles que tem maior complexidade. *"[...] a gente usa aqui uma ferramenta chamada Sonar, que faz uma análise de código. Uma das análises que ela faz é justamente a complexidade ciclomática, e ele dá o valor da complexidade ciclomática e onde estão sendo causados essas complexidades... [...] Ela dá um ranking lá, só que ela dá isso em forma numérica. Eu não sei quem causou aquilo muitas vezes.*" (GD3).

O uso dessa ferramenta indica não apenas o interesse dos gerentes de software em utilizar métricas de software, mas também indica especificamente o interesse deles em utilizar métricas relativas a complexidade ciclomática. Apesar de possuírem tais indicadores, o Sonar não é capaz de identificar os desenvolvedores que introduziram a complexidade ciclomática a determinadas classes do sistema.

Todos os outros gerentes alegaram achar interessante o uso de métricas para avaliação e acompanhamento de seus desenvolvedores. Contudo, três deles afirmaram que elas devem ser relativas ao número de tarefas resolvidas por desenvolvedor. Outros dois gerentes (GD4 e GR) relataram que, apesar de acharem interessante o uso de métricas para acompanhamento, não sabiam informar, no momento, quais seriam elas.

4.9.4 QP3) Quão útil é a abordagem para avaliar a contribuição individual dos desenvolvedores?

A terceira questão de pesquisa visa compreender se e como a abordagem apresentada, em especial suas métricas, poderiam ser úteis para a avaliação e o acompanhamento da contribuição individual dos desenvolvedores. Inicialmente a análise foi realizada considerando cada métrica individualmente, e, em seguida, considerou-se o conjunto das métricas.

[18] http://www.sonarqube.org/

4.9.4.1 Volume de Contribuição

A presente seção visa apresentar os memorandos sobre as métricas relativas a volume de contribuição. Incialmente, os memorandos discutem sobre a utilidade e a aplicabilidade das métricas, e em seguida, sobre as possíveis ameaças e limitações associadas a elas.

Utilidade e Aplicabilidade

Todos os gerentes de software entrevistados concordaram que a métrica de volume de contribuição é interessante para avaliar e acompanhar o desenvolvimento individual, contudo, três deles (GD1, GD2 e GR) ressaltaram que as métricas de volume devem estar acompanhadas de outras informações, como as tarefas realizadas pelo desenvolvedor, ou as métricas relativas à complexidade. "*Ela pode ser útil casada com outras informações, tipo junto com complexidade, que a gente sabe que ele colocou método de complexidade cinco, mas com volume de 62 linhas, então ele poderia ter feito isso melhor. [...] Não olhar só para aí, quer dizer, ver com as tarefas. Se foi um caso de uso novo, um novo MBean, um novo DAO, uma nova classe de domínio, então vai acrescentar bastante, então teria que casar com as duas coisas, e aqui talvez depois um quantitativo de tipo de tarefa.*" (GD2).

"[...] eu acho que pode combinar o volume com a complexidade" (GD1).

"*É um número isolado que você vê, que assim ajuda, mas aí você tem que sempre associar algo... mesmo assim né? A esse número. Por que é que um cara que produz mil e tantas numa semana, na outra ele cai pra 64? Pra 62? Então não é simples... será que o recurso... eu enquanto desenvolvedor estou sendo bem utilizado pela minha equipe? Eu trabalho por demanda. Depende muito disso também [...]*" (GR).

Essa ideia é reforçada por dois episódios específicos que ocorreram durante as entrevistas: ao serem questionados sobre se tinham alguma surpresa ao verem os valores das métricas de volume, dois gerentes de software (GD3 e GD4) se surpreenderam, cada um com um membro de sua respectiva equipe, pois não souberam explicar porque aqueles desenvolvedores tinham altos valores de contribuição em código.

Um dos gerentes de software (GR) acrescentou que as métricas de volume de contribuição poderiam ser úteis para provocar o gerente de software da equipe a investigar porque houve uma queda de desempenho de determinado desenvolvedor. "*Por quê uma*

pessoa sai de um nível tão alto de geração de código pra outra tão baixa? O que levou a isso?" (GR).

Um outro gerente entrevistado (GD3) também concordou que as métricas de volume são úteis, contudo, seria mais interessante que elas quantificassem contribuições em outros artefatos, e não apenas em artefatos com a extensão ".java", tendo em vista que algumas tarefas são concluídas com codificação em páginas JSP ou arquivos de configuração, que não são contabilizados pelas métricas de volume.

"*[...] talvez fosse interessante separar essa métrica, por exemplo, código Java ou código de programação executável, e de linguagem de marcação, seria CSS, JavaScript, HTML, talvez separando... pra a gente também saber quanto... por exemplo, porque muitas vezes o que acontece? Porque muitas vezes resolve alguns erros que é mais coisa de página, alguma coisa do tipo, alguma página está exibindo alguma informação incorreta, do tipo as vezes um layout que tem que ser alterado, coisas do tipo, então talvez fosse interessante...*" (GD3).

Ao ser questionado sobre as ameaças ou limitações associadas às métricas de volume, um gerente de software (GS) percebeu que as métricas poderiam ser usadas para formalizar a conjunto de critérios de avaliação dos desenvolvedores. A falta dessa formalização foi citada como uma das deficiências da abordagem de acompanhamento atual, além de ter sido como sugestão de melhoria no processo. "*É como eu disse, os riscos se tornam até positivos porque, talvez algumas pessoas olhando a sua produção diante dessas métricas, ele já tenha o instinto de melhorar. Então se torna positivo. [...] Com essa informação você sabe como o cara está produzindo dentro da sua equipe. Isso é positivo, eu acho positivo.*" (GS).

Um outro gerente (GR) questionou se as métricas relativas ao volume de contribuição poderiam ser utilizadas para avaliar a quantidade de demandas passadas para o desenvolvedor. "*Eu não sei se esse quantitativo pode estar influenciando nisso também. Naquela semana ali, eu fui demandado tanto, que eu fiz tanto. Ou essa semana eu não fiz tanto porque eu não fui demandado tanto assim. Não sei se poderia fazer uma análise por exemplo disso também.*" (GR).

Ameaças e Limitações

Uma limitação colocada por um dos gerentes (GD3) foi o fato de que as métricas de volume não contabilizam contribuições em artefatos não-Java. Dessa forma, essas métricas

podem penalizar um desenvolvedor que tem muitas contribuições em artefatos *front-end*, como páginas HTML, JSP, scripts CSS e JavaScript, pois elas não levam em consideração tais tipos de artefatos.

Uma ameaça apontada por outro gerente (GD4) é que as métricas de volume podem penalizar desenvolvedores que usem tecnologias mais modernas, ou técnicas mais sofisticadas para determinadas situações. *"[...] tem um cara que quebra mais a classe, já outro cara tenta fazer tudo numa linha só. Por exemplo, tem um cara que conhece Generics e faz as coisas com Generics, tem outro que faz com Iterator, então gera mais linhas de código, mas na realidade se você for olhar o mais correto, o que usou Generics tem um ponto maior aí, por conhecer a tecnologia.*" (GD4).

Um outro exemplo da ameaça discutida no parágrafo anterior é o uso de expressões regulares. Em determinadas situações, trechos de códigos inteiros podem ser substituídos por uma única linha usando expressões regulares. Dessa forma, as métricas de volume poderiam penalizar os desenvolvedores que usem expressões regulares, e, de certa forma, beneficiar aqueles que escrevem trechos de código.

Uma outra ameaça relatada (GT) foi que, uma vez que os desenvolvedores saibam que serão avaliados através da quantidade de linhas de código produzidas, modificadas e excluídas, eles podem tentar manipular uma maior quantidade de LOC apenas para dar a falsa impressão de que estão produzindo mais.

Dois gerentes entrevistados (GD2 e GR) alegaram que as métricas relativas ao volume de contribuição não devem ser utilizadas de forma isolada, pois dessa forma poderiam causar interpretações equivocadas. Eles afirmam que essas métricas devem estar acompanhadas de outras informações, como as tarefas realizadas pelos desenvolvedores, bem como outras métricas, como por exemplo, as métricas relativas a complexidade média por método.

Os outros gerentes de software (GD1 e GS) não visualizaram nenhuma ameaça em se utilizar as métricas de volume na avaliação da contribuição e acompanhamento dos desenvolvedores.

Sumário

Os principais resultados encontrados relativos as métricas de volume de contribuição são destacados a seguir.

(i) As métricas relativas a volume de contribuição são úteis para avaliação da contribuição do desenvolvedor, contudo, precisam ser complementadas com outras informações, como, por exemplo, a relação de tarefas desempenhadas pelo desenvolvedor no período analisado;

(ii) Permitem que o gerente de projeto de software avalie quedas acentuadas de produção de código;

(iii) Contribuem para formar o conjunto de critérios de avaliação da produtividade dos desenvolvedores. Por outro lado, é um conjunto de métricas facilmente manipuláveis para dar a falsa impressão de boa produtividade;

(iv) Deveriam também quantificar contribuições em artefatos não-Java;

(v) Penaliza desenvolvedores que utilizam técnicas mais modernas de programação, como Generics e expressões regulares, que permitem reduzir a quantidade de linhas necessárias.

4.9.4.2 Complexidade Média por Método

A presente seção visa apresentar os memorandos sobre as métricas relativas a complexidade média por método.

Utilidade e Aplicabilidade

As métricas relativas à complexidade média por método da contribuição também foram positivas na visão dos gerentes entrevistados, pois apenas um deles não concorda que essas métricas seriam úteis para o acompanhamento e avaliação dos desenvolvedores.

Entre as colocações positivas feitas pelos gerentes em relação a essas métricas, está o fato de que elas permitem o gerente avaliar como a complexidade ficou distribuída, e cobrar de seus desenvolvedores que produzam código com menor complexidade para facilitar futuras manutenções. [Entrevistador]: "*Então você acha que essa métrica, ela ajuda você a acompanhar o desenvolvedor?*"; [GD2]: "*Ajuda. Essa ajuda. A gente até saber... e saber como é depois pra... ou reduzir essa complexidade em um método só, em quebrar... fazer algo diferente que ajude na manutenção futura.*".

[Entrevistador]: *"Então você acha que essa métrica faz sentido?*"; [GD3]: "*Faz, para fazer uma análise técnica dos desenvolvedores. Porque assim, quanto maior a complexidade, mais difícil e mais acoplado o sistema... não sei se mais acoplado, mas assim mais complexo*

ele fica de se fazer algum tipo de ajuste, de alteração, de uma possível evolução no sistema, então eu acredito que dê para fazer uma boa métrica sim.".

Ao ser questionado sobre as ameaças ou riscos dessas métricas, um dos gerentes (GD3) sugeriu que, para se fazer uma análise mais detalhada do código que está sendo produzido pelo desenvolvedor, as métricas de complexidade média por método devem ser acompanhadas pelos valores base, como a quantidade de métodos adicionados e modificados pelo desenvolvedor, bem como a complexidade ciclomática de código adicionado e modificado por ele. Dessa forma, o gerente da equipe poderia compreender melhor o porquê dos valores das métricas de complexidade média por método.

[Entrevistador]: "*Certo. Bom, mas essa métrica você vem algum problema em se usar ela? Você vê algum tipo de ameaça ou risco?*"; [GD3]: "*Não, eu vejo como uma boa ferramenta de análise, pelo menos, e talvez não só a métrica em si mas a métrica associada àquela outra planilha lá. Que eu acho que, você só vendo a métrica, talvez você não consiga ter essa análise propriamente dita, mas por exemplo: 'ah, essa métrica deu tanto, mas quantos métodos realmente ele ajustou? E quanto de complexidade ciclomática ele introduziu?'"*; [Entrevistador]: "*Saber o porque desse valor...*" [GD3]: "*Exatamente. Talvez também fosse interessante, então você ter... aí serem três colunas: o adicionado, mas ser métodos inseridos, complexidade ciclomática dos métodos inseridos, e a média... a média não, a divisão. A divisão dos dois.*".

Um outro gerente de software (GD1) sugeriu que, para compreender melhor sobre porque um desenvolvedor está apresentando tais valores de complexidade média por método, eles devam estar acompanhados da lista de tarefas realizadas pelo desenvolvedor, pois isso permitirá fazer um cruzamento entre os valores de complexidade e as tarefas realizadas pelo desenvolvedor, associadas a um julgamento subjetivo das complexidades das tarefas desempenhadas.

[GD1]: "*[...] a gente pode fazer uma análise aqui agora, mas era bom pegar as tarefas que foram trabalhadas naquela semana e tentar... oh porque é que ele está adicionando tanta complexidade numa tarefa que é tão simples? Então fazer esse cruzamento com... talvez isso aqui ao invés de combinar com essa talvez seja com a lista de tarefas.*" [Entrevistador]: "*A complexidade associada com a lista de tarefas.*"; [GD1]: "*Isso. Fosse um negócio mais útil...*".

Pequenas variações para essas duas transcrições foram dadas pelos outros gerentes de software entrevistados. Apenas um gerente (GD4) relata que essas métricas não poderiam ser utilizadas para acompanhamento dos seus desenvolvedores, pois, na visão dele, o fato do desenvolvedor concentrar a complexidade de sua contribuição, não justifica que o gerente solicite que ele refatore seu código.

"*Entendi. Mas... isso é muito íntimo. Por exemplo, desses dois métodos que ele criara realmente tem esses dois fluxos, esses três fluxos a mais, de uma forma ou outra. Não vejo como penalizar ele por um código que ele adicionou que gerou um pouco mais de complexidade. Entendeu? Então assim, não vejo como isso me traria alguma métrica pra avaliação não. [...] Até mesmo por exemplo, para melhorar, se tivesse que chegar pra ele e dizer assim: melhore porque seu código está gerando complexidade. Não tinha nem o que eu dizer pra ele. Complicado eu dizer assim: melhore a complexidade ciclomática do seu código, diminua a quantidade de caminhos. Mas ele chega pra mim e diz: não GD4, mas aqui ou eu vou pra um lado ou eu vou pra o outro. É normal, então. Eu acho que não vai trazer nenhum...*" (GD4).

Apesar de acreditar que as métricas relativas à complexidade ciclomática não serviriam para realizar o acompanhamento de seus desenvolvedores, esse mesmo gerente de software acredita que essas métricas seriam mais úteis se fossem baseadas na notação do Grande-O[19], pois ele acredita que, nesse caso, ele poderia solicitar ao seu desenvolvedor que revisasse seu código para melhorar o desempenho.

"*[...] se a gente não usasse complexidade ciclomática e usasse complexidade em termos das ordens O(n), pudesse trazer alguma coisa a mais.*" (GD4).

Ameaças e Limitações

A única ressalva feita por um gerente de software (GD3) em relação às métricas relativas a complexidade média por método, foi que elas deveriam estar acompanhadas de seus valores base, ou seja, a complexidade adicionada/modificada e a quantidade de métodos adicionados/modificados. O gerente afirma que esses valores base são importantes para

[19] http://pt.wikipedia.org/wiki/Grande-O

compreender melhor porque o desenvolvedor está apresentando tais valores de complexidade média por método.

Sumário

Os principais resultados encontrados para as métricas relativas à complexidade média por método são destacados a seguir.

(i) Permite que o gerente de projeto de software realize uma análise técnica da qualidade do código produzido, sob a ótica da complexidade ciclomática de McCabe;
(ii) Devem ser acompanhadas de seus valores base, para permitir explicar porque determinador desenvolvedor está apresentando tais valores para as métricas;
(iii) Devem ser acompanhadas da relação de tarefas desempenhadas pelos desenvolvedores.

4.9.4.3 Introdução de *Bugs*

Esta seção apresenta os memorandos sobre a métrica relativa a introdução de *bugs*.

Utilidade e Aplicabilidade

As considerações a respeito da métrica de introdução de *bugs* foram heterogêneas. Alguns gerentes acreditam que elas sejam interessantes, outros dizem que ela não pode ser utilizada de forma isolada, e outros chegam a alegar que elas carregam muitas ameaças que podem inviabilizar a sua aplicabilidade e utilidade.

Através da leitura dos valores apresentados pela métrica de introdução de *bugs*, um gerente de software (GD1) constatou que um de seus desenvolvedores, que ele acreditava ter baixos índices de erros (baseado nos erros de teste das tarefas realizadas pelo desenvolvedor), de fato apresentava baixos valores para essa métrica. "*Então ela já trabalhou em módulos completos que ela desenvolveu. Módulos novos. Então é realmente uma constatação do que eu falei, que o que ela manda de código vem com erros mas numa quantidade baixa em relação aos outros.*" (GD1).

Outro gerente de software (GS) viu a métrica como positiva e útil para avaliação dos desenvolvedores. [Entrevistador]: *"[...] você acha que essa métrica faz sentido pra o gerente de software acompanhar os seus desenvolvedores?*"; [GS]: "*Faz, faz sim. Por que? Porque*

você vai identificar quem da sua equipe tá gerando mais bug, quem commita mais bug.". Ele ainda acrescenta que essa métrica pode ajudar o gerente a avaliar a quantidade de demandas passadas para um determinado desenvolvedor. Na visão dele, erros de software podem ser causados pelos desenvolvedores pela falta de tempo para testar seu código de forma apropriada. Portanto essa métrica poderia ser uma ferramenta útil para identificar esses casos. "*Baseado nessa informação talvez ele faça isso por que? Porque o prazo está muito curto, ele precisa fazer uma coisa com rapidez, com urgência, e aí a qualidade cai. Ele não tem o tempo de fazer a revisão naquele código*" (GS).

Outra opinião interessante dada por outro gerente de software (GT), é que essa métrica é capaz de identificar casos de desenvolvedores que realizam o *commit* de seu código-fonte sem realizar testes triviais. "*Acho que pode identificar... porque as vezes tem casos críticos de desenvolvedores que, não é que errem muito, mas que commitam e não tem aquela atenção, então dá para detectar muito isso aí. Pessoas que commitam de todo jeito... aqueles casos que eu falei, que tem gente que commita, não testa, nem nada e já manda para teste. E aí você detecta logo aqueles erros mais rápido, e sabe que é aquela pessoa mesmo.*" (GT). Esse problema também foi apontado por esse mesmo gerente de software como uma das características negativas dos desenvolvedores. Portanto essa métrica pode oferecer suporte adequado ao gerente de software, no que concerne a avaliação do desenvolvedor na perspectiva da quantidade de *bugs* introduzidos por ele.

Na opinião de outro gerente de software (GR), essa métrica não pode ser utilizada de forma isolada, pois isso poderia induzir o gerente da equipe a interpretações equivocadas. "*[...] é um analista nosso por exemplo, que ele é pleno, é sênior nosso aqui. Ele tem um alto... o desenvolvimento que ele pega é de alta complexidade, é um cara que desenvolve muito, que contribui muito, que gera muito código. Então 85 pra mim ali não representa nada. Não quer dizer que o cara... não, de forma alguma. Tá entendendo? Então ela isolada não me ajuda, eu precisaria ver ela associada ao que ele está produzindo.*" (GR).

O valor 85 citado pelo gerente se refere a quantidade de *bugs* introduzidos pelo desenvolvedor numa determinada semana. Na visão dele, apesar do desenvolvedor ter introduzido grande quantidade de *bugs* no período (baseado nos valores de referência), não justifica penalizá-lo de nenhuma forma, por se tratar de um excelente profissional e que trabalha com tarefas consideradas de alta complexidade.

Ameaças e Limitações

A principal ameaça identificada para essa métrica, constatada por seis dos sete gerentes entrevistados, através da observação dos valores, foi a de que os desenvolvedores que possuíam os maiores valores para essa métrica, ou seja, introduziram mais *bugs*, eram aqueles que tinham mais experiência e estavam a mais tempo no projeto. Portanto, na visão dos gerentes, essa métrica tende a penalizar os desenvolvedores mais experientes da equipe, pois eles têm mais código produzido se comparado a desenvolvedores mais novos, e portanto, têm mais chance de serem apontados como culpados pela métrica.

[GD4]: "*O que tem mais contribuição, mais commit, você vai ver que quanto mais contribuição, mais bug commit.*".

[GD2]: "*Porque [omitido] está aqui desde quando começou, então ele tem mais código do que [omitido] que entrou a pouco tempo, do que [omitido], que começou agora, então a gente vê os mais antigos têm mais código, por isso...*".

Um gerente de software (GD2) relata que, no contexto da SINFO, as tarefas são abertas, em sua grande maioria, pela equipe de requisitos, que pode cadastrá-la com uma classificação incorreta, causando falsos positivos na métrica de introdução de *bugs*. "*[...] talvez a tarefa entrou como um erro, mas era alteração de regra de negócio, então não é só erro, então teria essa eventualidade também. [...] E outra, a tarefa não é aberta por alguém que conhece de tipo de código e regra. Ela é aberta por requisitos, em sua maioria. Então a gente tem alguns erros de abertura de tarefas, tarefas de erro entra como administrativo, administrativos entram como erros, então tem esse problema também.*" (GD2).

Outro problema destacado por dois gerentes entrevistados (GD2 e GR), é que os valores apresentados pela métrica de introdução de *bugs* não se limitam a um intervalo de tempo específico, pois o *bug* pode ter sido introduzido pelo desenvolvedor num momento muito anterior ao período em que o gerente de software está interessado em analisar. Dessa forma, o gerente poderia penalizar o desenvolvedor por causa de um *bug* introduzido há muito tempo atrás.

Outro gerente de software (GD3) afirma que essa métrica não deve ser utilizada de forma isolada, uma vez que poderia causar interpretações equivocadas. Em vez disso, o gerente de software deve utilizá-la combinada com outras informações complementares.

Um dos gerentes (GR) afirma que é importante observar sobre quais circunstâncias o desenvolvedor estava ao ser avaliado sobre o quantitativo de *bugs* introduzidos por ele, pois é

preciso considerar não apenas o valor da métrica, mas também quais pressões o desenvolvedor sofreu para cumprimento dos prazos, quantas demandas ele tinha durante o período, entre outros aspectos.

Sumário

Esta seção apresenta um sumário dos resultados sobre a métrica relativa a contribuição em correção de *bugs*.

(i) Permite identificar casos de desenvolvedores que fazem *commit* de código sem realizar testes triviais;

(ii) Não pode ser usada de forma isolada, sob o risco de permitir interpretações equivocadas;

(iii) Essas métricas tendem a penalizar os desenvolvedores mais experientes e que estão há mais tempo na equipe, pois, dada a forma como a mineração encontra o desenvolvedor que introduziu o *bug*, a chance deles serem apontados como culpados é maior;

(iv) É preciso observar em quais circunstâncias o desenvolvedor estava ao introduzir aquele *bug*, como por exemplo a quantidade de demandas atribuídas a ele, a pressão sofrida para cumprimento dos prazos, entre outros aspectos.

4.9.4.4 Contribuição em Correção de *Bugs*

Utilidade e Aplicabilidade

No contexto específico da SINFO, as equipes de desenvolvimento possuem células de correção de erros, que são desenvolvedores que concentram a maior parte das tarefas relativas à correção de *bugs*, e normalmente é composta por um ou dois desenvolvedores. Através da leitura dos valores ao longo do tempo, e pela opinião dos próprios gerentes de software, foi possível constatar que a métrica de contribuição em correção de *bugs* foi bem sucedida em indicar quem são desenvolvedores que compõem as células de correção em cada equipe.

Segundo dois gerentes de software (GD1 e GD4), em determinados períodos há um aumento na ocorrência de tarefas de erro, e portanto eles tendem a distribuir mais tarefas desse tipo entre os membros da equipe, ao invés de concentrá-las na célula de correção. Nesse tipo de situação, um dos gerentes (GD1) comentou que, quando houvesse uma maior distribuição das tarefas entre os outros desenvolvedores, ele poderia cobrar mais dos membros

da célula pela realização de outros tipos de tarefas, como por exemplo, o desenvolvimento de novas funcionalidades. "*[...] aquela semana eu distribuí mais os erros, é mais uma coisa pra reforçar que ele poderia ter produzido mais. Que ele teve ajuda de outras.*" (GD1).

Contudo, um dos gerentes de software (GD4) viu essa questão sob uma ótica diferente: apesar de também reconhecer que, em alguns períodos, as tarefas de correção de erros podem ser alocadas a desenvolvedores que não fazem parte da célula de correção, a sua opinião é de que a métrica não teria utilidade, uma vez que apenas indicaria quem foram os desenvolvedores que mais receberam esse tipo de tarefa, informação essa que o gerente já sabe.

"*A atribuição dele, junto a de [omitido], é corrigir erros. Ou [omitido] ou outro cara aqui olha, [...], ele tem 25 erros aqui... 33 erros. Ou seja, Bug Fixing talvez não vá... vamos ver o histórico, vá passando pra a gente ver. Olha, 21%, 10%, ou seja, é o cara que... 19%, então assim não vai trazer muita... 30%... ele sempre vai estar com a quantidade maior. Ele é o cara do [omitido] que corrige erros. [...] É sempre vai aparecer um cara. Talvez outro e outro em momentos que, por exemplo, quando ele está muito atolado, aí eu jogo outras pessoas também para corrigir erros. Quando, por exemplo, a gente acaba de lançar uma build, aí tem um fluxo maior de erros, aí eu pego e jogo... ah bota [omitido], bota [omitido] pra corrigir erros, boto [omitido]. Distribuo na equipe para dar uma ajuda. Mas essa métrica aí vai trazer só... na realidade vai confirmar o que a gente já sabe mesmo.*" (GD4).

Ameaças e Limitações

Conforme relatado por um gerente (GD2) e discutido anteriormente, algumas tarefas podem ser classificadas incorretamente, pois elas são abertas, em sua maioria, por pessoas que não possuem tanto conhecimento técnico. A classificação errônea das tarefas quanto ao seu tipo pode provocar a ocorrência de falsos positivos para essa métrica, pois uma tarefa que demanda, por exemplo, o desenvolvimento de uma nova funcionalidade poderia ser classificada como correção de erro. Esse problema de classificação das tarefas também foi relatado por outro gerente (GT). Ele afirma que, por exemplo, são abertas tarefas de erro para solicitações de mudanças na interface gráfica, mesmo que ela não esteja apresentando nenhum problema.

Segundo o gerente de software (GD3) de uma das equipes de desenvolvimento, no caso específico do sistema que eles desenvolvem, essa métrica irá apresentar muitos falso-

positivos. Isso ocorre porque, no caso específico dessa equipe, um funcionário que está lotado num órgão em Brasília (DF) é o responsável por averiguar cada tarefa e fechá-la. Contudo, antes de fechar a tarefa, ele cadastra um *log* de desenvolvimento. Portanto, a métrica irá quantificar os *logs* cadastrados por esse funcionário mesmo sem ele ter contribuído na correção do *bug*. "*Todo mundo vai desenvolvendo as tarefas, e no final ele fica de se reportar aos gestores lá da [omitido] para fazer o fechamento das tarefas. Então pode ser que ele sempre é o que tem mais, porque ele faz justamente o último log de desenvolvimento nas tarefas.*" (GD3).

Um outro gerente de software (GD1) comentou que é comum o uso de *logs* de desenvolvimento para a realização de comentários, apesar de se ter outros tipos de *logs* mais adequados para tal propósito. No caso das tarefas de correção de erros, esses *logs* também serão contabilizados, mesmo que o desenvolvedor não tenha contribuído em desenvolvimento para a solução daquele problema. "*[...] só a título de informação pra você: que a gente usa log de desenvolvimento pra fazer comentários.*" (GD1).

Esse mesmo gerente (GD1) informou que existem algumas tarefas que são classificadas como bloqueantes, pois elas impedem a conclusão do trabalho do usuário, e portanto têm maior urgência na sua resolução. Ele alegou que tais tarefas não possuem *logs* de desenvolvimento, pois o desenvolvedor cadastra *logs* de solicitação de *update* em banco de dados ou solicitação de *update* em produção, para que a correção seja aplicada em produção mais rapidamente, e permita que o usuário conclua a operação que estava realizando. Como a métrica só contabiliza *logs* de desenvolvimento, essas contribuições não seriam consideradas.

Outra situação que pode ocorrer, relatada por este mesmo gerente de software (GD1), é que algumas tarefas de correção de *bugs* não demandam a produção ou alteração de código-fonte, portanto não possuem *logs* de desenvolvimento. *"[...] porque tem tarefa que é só código, só atualização de base de dados, não tem código... pode acontecer de eu abrir uma tarefa e dizer que é pra fazer um update ou atualizar a tabela de percentuais do INSS, alguma coisa assim. Então ele gera o script e o log não é de desenvolvimento, é de solicitação de update de banco. Quer dizer a pessoa trabalhou, dependendo da complexidade do update, pode ter trabalhado muito tempo [...]*" (GD1). Nesses casos, a métrica não levaria em consideração a contribuição do desenvolvedor nessas tarefas.

Além disso, esse gerente de software (GD1) ainda ponderou que é preciso analisar melhor as tarefas de correção de *bugs* realizadas pelo desenvolvedor, e não apenas considerar

a quantidade de tarefas resolvidas, porque é preciso considerar também complexidade delas para a avaliação do gerente. [Entrevistador]: "*Então quando o cara coloca um log numa tarefa de correção de erro você já conta aquela tarefa como ele... contribuiu pra aquela tarefa?*"; [GD1]: "*Contribuiu mas aí eu abro a tarefa e vejo... porque as vezes o cara escreveu alguma coisa lá e não fez muita coisa. Gastou 5 minutos pra analisar uma coisa, boto logo... posso até considerar baixa a complexidade.*".

Dois gerentes de software (GD1 e GR) acrescentaram que o valor percentual da métrica pode não ajudar muito, pois isso promove uma comparação entre os desenvolvedores da mesma equipe, e essa comparação pode não ser justa porque os desenvolvedores que compõem a célula de correção de erros são responsáveis por esse tipo de atividade, portanto possuem menor quantidade de tarefas deste tipo. Segundo esses gerentes, a comparação deve ser feita usando valores absolutos e entre as células de correção de *bugs* de cada equipe de desenvolvimento.

"*E aí o comparativo dele [se referindo ao desenvolvedor que compõe a célula de correção de erros da sua equipe] é em relação a [o desenvolvedor da célula de erros da outra equipe], que ele é a célula de correção de erros do SIPAC. Então assim, vai ser um... por que é que está tão distante a quantidade de tarefas que [omitido] está resolvendo do que [omitido] está?*" (GD1).

"*É, aí a gente tem que analisar, por exemplo, da mesma forma, ele isolado não funciona, por que? Porque dentro das equipes de desenvolvimento tem equipes de correção de bugs, então eu tenho que analisar entre eles, e não entre... por exemplo, uma pessoa que está trabalhando com correção de bugs e outra que está trabalhando com funcionalidade nova.*" (GR).

Um dos gerentes de software (GT) apresentou dois pontos de vista diferentes e interessantes a respeito dessa métrica. O primeiro ponto de vista foi colocado por ele, logo após a explicação sobre como a métrica funcionava, argumentou que alguns desenvolvedores cadastram um *log* de desenvolvimento a cada pequeno incremento na tarefa, enquanto outros desenvolvedores cadastram um único *log* de desenvolvimento ao concluí-la. Dessa forma a métrica tende a beneficiar aqueles que cadastram vários *logs* de desenvolvimento a cada incremento, e penalizar aqueles que cadastram um único *log* ao concluir a tarefa.

Ao ser questionado se ele acreditava que a métrica seria mais adequada se contabilizasse a quantidade de tarefas de correção de erros, em vez de contar o número de *logs* de desenvolvimento em tarefas desse tipo, o referido gerente (GT) colocou seu segundo ponto de vista: "*[...] o caso em que ele corrigiu, aqui ele colocou a revisão, aí quando ele foi fazer um teste, alguma coisa, encontrou outra coisa. Aí corrigiu de novo, colocou de novo outro log. Com a mesma classe [...]*" (GT). Nesse segundo ponto de vista, a preocupação dele se volta para os casos em que a tarefa volta de testes algumas vezes, gerando um esforço maior para a conclusão da tarefa.

Ele (GT) conclui o seu raciocínio com o seguinte questionamento: *"[...] ele é bom porque ele com um log resolveu tudo, ou às vezes com quatro logs para resolver errando e colocando e perdendo tempo. Está entendendo? Assim, eu estou perguntando: qual é o bom? É que o Bug Fixing seja maior ou menor?*" (GT). O questionamento feito pelo gerente de software é uma evidência de que a forma como as métricas são interpretadas devem ser analisadas com cuidado para não gerar a conclusões equivocadas.

Essa mesma preocupação foi apresentada por um outro gerente de software (GD1), quando afirmou que talvez o quantitativo de tarefas resolvidas fosse mais interessante do que a contagem dos *logs* produzidos pelo desenvolvedor. "*Mas aí o quantitativo dos logs é que eu não sei se é tão útil, em relação... pra mim eu acho que a quantidade de tarefas em si.*" (GD1).

Sumário

Os principais resultados encontrados para as métricas relativas à contribuição em correção de *bugs* são destacados a seguir.

(i) Teve êxito em identificar os desenvolvedores que representam a célula de correção de erros nas equipes de desenvolvimento;
(ii) Útil para que o gerente identifique como se deu a distribuição de tarefas de correção de erros entre os desenvolvedores da equipe;
(iii) Tarefas de alta prioridade, classificadas como bloqueantes, não possuem *logs* de desenvolvimento, portanto a métrica não levou em consideração contribuições nesses tipos de tarefas;

(iv) Alguns tipos de tarefas não demandam correção de código-fonte, e sim ajustes no banco de dados, portanto a métrica também não levou em consideração as contribuições nessas tarefas;

(v) O valor absoluto da métrica pode ser mais útil do que o valor percentual, pois esse oferece uma comparação entre os membros da equipe, e aquele permitiria uma comparação entre as células de correção de erros das equipes;

(vi) O quantitativo de tarefas de correção de erros pode ser mais útil do que o número de *logs* de desenvolvimento produzidos pelo desenvolvedor.

4.9.4.5 Visão Geral sobre a Aplicabilidade e Utilidade das Métricas

Essa seção apresenta os memorandos que discutem sobre os benefícios e limitações das métricas na visão dos gerentes de software. Apesar de serem questionados a respeito das métricas como um todo, e não mais cada uma delas individualmente, alguns gerentes preferiram não generalizar, e deram a sua opinião apontando os benefícios e limitações de cada uma das métricas.

Benefícios

Dois gerentes (GD2 e GR) afirmaram que as métricas relativas ao volume de contribuição são úteis quando combinadas com outras informações, como as tarefas realizadas pelos desenvolvedores no período, bem como outras métricas.

As métricas de complexidade média por método e contribuição em correção de erros foram apontadas, por um dos gerentes (GD2), como sendo um retrato do período analisado. Tal opinião foi reforçada pela constatação que o gerente fez através dos valores alterados das métricas durante a semana de carnaval. "*Então a gente viu até na semana do carnaval, que eu achei estranho os valores, mas porque explicou-se que era uma semana de carnaval e teve apenas dois dias, por isso que os valores foram diferentes.*" (GD2).

Esse gerente ainda acrescenta que as métricas podem auxiliá-lo no acompanhamento sob uma ótica mais técnica, permitindo observar os desenvolvedores mais produtivos e analisar a necessidade de treinamento daqueles que apresentam indicadores negativos. "*A gente pode ver a produtividade em código do desenvolvedor, eu sei a produtividade eu consigo mensurar por tarefa e por complexidade, mas por código, por técnica, isso iria ajudar bastante. E até ver qual desenvolvedor precisaria de algum treinamento a mais, de*

melhoria de padrões. Então a gente consegue saber os dois lados, quem tá desenvolvendo bastante e quem precisa de um auxílio ou de uma ajuda de parte técnica." (GD2).

Tal opinião também foi dada por um outro gerente de software (GD3), ao dizer que as métricas permitiram ele identificar os desenvolvedores que necessitam de treinamento para melhorar a qualidade de sua produção. [Entrevistador]: "*Você acha que essas métricas, o gerente de posse delas, consegue compreender melhor, usá-las como indicadores para compreender melhor o que se passou durante a semana?*".

[GD3]: "*Sim, inclusive até mesmo pra que o próprio gerente consiga realizar estratégias para tentar melhorar, inclusive, tecnicamente, talvez cursos de nivelamento, de melhores práticas, coisas do tipo acho que é bem interessante pra que a equipe fique mais homogênea. Para que você consiga números mais... como é que eu posso dizer? Mais pertos um do outro. Porque a gente pode ver aqui que está bem heterogêneo, tem semana que um faz muita coisa, o outro não faz. Então tentar balancear isso.*".

Ao ser questionado sobre as dificuldades que ele tinha em realizar a avaliação de produtividade mensal de seus desenvolvedores, esse mesmo gerente de software (GD3) alegou que a abordagem de acompanhamento atual toma muito do seu tempo. Após conhecer as métricas oferecidas, ele afirma que elas podem contribuir na avaliação da produtividade mensal de seus desenvolvedores. "*E até também em termos de produtividade também, de saber quem produz melhor, quem causa mais erros, que não causa. Talvez isso aí seja um bom indicador.*" (GD3).

Um outro benefício vislumbrado por um outro gerente de software (GD1), foi que os valores das métricas poderiam ser utilizados como base para justificar ao desenvolvedor o porquê dele estar recebendo aquele bônus de produtividade mensal. Segundo o gerente, usando apenas a quantidade de tarefas resolvidas pelo desenvolvedor, é possível que ele não consiga dar uma justificativa satisfatória ao desenvolvedor. "*Porque mesmo eu falando uma conversa pessoal e dizendo: poderia ter se esforçado mais, talvez só a quantidade de tarefas não seja suficiente. Mas mostrando esses números aqui, isso sirva para esclarecer*" (GD1).

Segundo um dos gerentes de software (GR) as métricas podem complementar a percepção subjetiva que os gerentes têm diante de seus desenvolvedores. Através da leitura do histórico dos valores das métricas, o gerente pode perceber uma mudança acentuada nos valores, e isso pode indicar algum problema externo, de ordem pessoal ou profissional do

desenvolvedor. Dessa forma, o gerente pode conversar com o desenvolvedor para tentar identificar os problemas e, se possível, ajudá-lo na solução.

[Entrevistador]: "*Mas trazem benefícios [se referindo às métricas]?*"; [GR]: "*Trazem sim, com certeza. Eu acho que até pra você entender o outro lado. Pra você poder entender que, é a percepção, e as vezes nem todo mundo tem isso, de você conseguir perceber o outro, e isso daqui ajuda isso. A minha visão é que ele vai ajudar ao gerente a ter essa visão. Dizer 'opa! Aconteceu alguma coisa. O que foi?', aí eu vou atrás disso.*"; [Entrevistador]: "*Sim, entendi. Tentar descobrir porque.*"; [GR]: "*Tentar descobrir, saber o porquê que está acontecendo aquilo dali, porque até então está na minha subjetividade. Você tem uma ferramenta que me ajuda até nisso, é fantástico.*".

Um dos gerentes de software (GD4) acredita que as métricas relativas ao volume de contribuição irão estar em sintonia com a avaliação baseada em tarefas que eles fazem mensalmente, para definição do bônus de produtividade mensal. "*Volume sim, vai ajudar. Até porque isso no final do mês vai me dar um total, e eu desconfio que o total vai bater com esse daqui. Então isso é interessante, esse Volume aí eu gostei muito.*" (GD4).

Outro gerente de software (GS) notou a importância da interpretação dos valores das métricas associado ao contexto atual em que sua equipe se encontra, ou seja, usando também informações sobre as tarefas demandadas para a equipe, possíveis atividades externas que podem reduzir a produção durante o período, etc.

[GS]: "*O gerente vai ter que saber fazer a leitura disso aí com o momento que a equipe vive. Que de repente você está fazendo um novo ajuste no SIGAA, por exemplo, um novo ajuste no documento da graduação, então você sabe que tem a pressão da Pró-Reitoria de Ensino o tempo todo porque você tem prazos. Então a equipe traduz isso aqui, aí você sabe que dentro daquela pressão existente houveram... aconteceram várias demandas dos vários módulos, mas que a equipe conduziu isso aqui naquela semana. Na próxima semana pode... de acordo com o que for planejado, você pode identificar que a produção vai ser menor, ou maior. E aí o gerente tem que saber identificar e fazer a leitura desses dados aí.*".

Outro benefício que vale ser destacado é que, apesar da importância da avaliação subjetiva feita pelos gerentes de software, eles anseiam por ferramentas que os permitam complementar essa avaliação, tornando elas menos baseadas em suas próprias opiniões e fazendo-as mais baseadas em dados e opiniões de outros. [GR]: "*Eu tenho estudado algumas*

ideias, algumas ferramentas que possam contribuir de alguma forma pra que isso fosse menos 'minha opinião'. E é por isso que eu..."; [Entrevistador]: "*Menos subjetivo.*"; [GR]: "*Menos subjetivo, exato.*".

Limitações

Dois gerentes (GD2 e GR) afirmaram que as métricas não podem ser usadas de forma isolada, mas sim combinadas com outras métricas ou outras informações provenientes das tarefas desempenhadas pelo desenvolvedor. "*É, as limitações é porque o número isolado é complicado, quando você começa a ver várias métricas, vários indicadores, aí você começa a complementar as informações. Mas mesmo assim eu acho que falta essa questão do pessoal. Que aí não tem como você fugir da análise qualitativa mesmo.*" (GR).

Além disso, um gerente (GR) destacou, em sua fala, que apenas as informações provenientes das métricas não são suficientes para realizar uma avaliação completa a respeito dos seus desenvolvedores, tendo em vista que essas informações não consideram questões de ordem pessoal e, portanto, o gerente de software não pode abandonar a sua avaliação subjetiva.

Uma limitação destacada por dois gerentes de software (GD1 e GD2) foi em relação às métricas relativas a introdução de *bugs*, por não apresentar o cenário exato do período em que o gerente está analisando, e pelas ameaças associadas à técnica que a mineração utilizada para identificar o desenvolvedor que introduziu o *bug*, que podem penalizar desenvolvedores mais experientes.

Uma outra limitação destacada por outros dois gerentes (GT e GD3) foi relativa à métrica de contribuição em correção de *bugs*, que pode penalizar ou beneficiar um desenvolvedor dependendo da forma como ele registra os *logs* nas suas tarefas. Outro importante destaque feito por um dos gerentes (GR) foi a limitação que essas métricas possuem relativas à comparação entre desenvolvedores. Esse gerente alega que, numa mesma equipe, existem desenvolvedores que ocupam diferentes cargos no projeto dependendo de sua experiência e tempo de serviço, portanto a comparação das métricas deve ser feita entre desenvolvedores pertencentes ao mesmo cargo, para não ser injusto com aqueles menos experientes.

[GR]: "*Porque é complicado eu analisar um grupo que é tão diferente. Se eu tenho um Analista I, um programador e um Analista II, eu teria que comparar dentro desse grupo, e*

não no todo. Porque eu não posso comparar um programador com um Analista II, com um Sênior, é desonesto, é injusto. É injusto demais. Então eu acho que se a gente conseguisse fazer dessa maneira seria mais interessante.".

4.10 Discussões

Esta seção tem como objetivo apresentar algumas discussões sobre o estudo de avaliação da abordagem que foi conduzido na SINFO/UFRN. A riqueza do estudo qualitativo é permitir não apenas responder às questões de pesquisa iniciais, mas também oferecer uma compreensão mais ampliada sobre várias outras questões, conforme discutidas nesse capítulo.

A discussão se inicia com o cruzamento das respostas obtidas em diferentes questões de pesquisa, e, em seguida, discute sobre mudanças que podem ser implementadas nas minerações e nas métricas a fim de melhorar a abordagem proposta.

4.10.1 Como as métricas podem contribuir para a abordagem atual de definição do bônus de produtividade dos desenvolvedores?

O presente questionamento visa analisar como as métricas podem contribuir para a atual abordagem de acompanhamento e avaliação realizada pelos gerentes de SINFO. Ao responderem às perguntas da primeira questão de pesquisa, os gerentes deixaram claro que a avaliação da produtividade dos desenvolvedores é feita através de uma análise das tarefas realizadas por eles, associadas a uma complexidade das tarefas que é definida de forma subjetiva pelos gerentes de software, levando-se em consideração informações sobre a própria demanda que a tarefa apresenta, bem como a sua experiência.

As métricas associadas a dois atributos de avaliação, volume de contribuição e complexidade média por método, analisam a contribuição individual do desenvolvedor num nível mais baixo de abstração, ou seja, linhas de código e complexidade ciclomática de McCabe, respectivamente. Portanto as métricas associadas a esses dois atributos de avaliação não são capazes de automatizar ou facilitar a produção de indicadores utilizados pela atual abordagem de avaliação (tarefas e complexidade das tarefas), contudo, elas são capazes de complementar essa avaliação com outras informações julgadas relevantes pelos gerentes de software.

Essa percepção foi dada por vários gerentes de software após a apresentação das métricas: as métricas oferecidas pela abordagem de acompanhamento não podem ser

utilizadas de forma isolada, mas devem complementar a avaliação subjetiva que o gerente de software realiza, pois elas não quantificam aspectos considerados pelos gerentes, como aspectos humanos e outras atividades não contabilizadas pelas métricas, tais como, participação em reuniões, treinamentos, suporte a outros desenvolvedores, e tantos outros tipos de atividades. As informações complementares oferecidas pelas métricas poderiam auxiliar o gerente de software a realizar uma avaliação da produtividade de seus desenvolvedores com base em mais dados, e de forma menos subjetiva, conforme desejo apresentados por eles próprios.

É interessante notar também que as métricas podem ser produzidas de forma automática e periódica, podendo, dessa forma, contribuir para melhorar a agilidade na atual abordagem de avaliação do bônus de produtividade dos desenvolvedores, em que alguns gerentes relatam que tomam muito do seu tempo. Ou seja, as métricas podem sim contribuir de forma complementar, e não substitutiva, para a atual abordagem de avaliação dos desenvolvedores da SINFO/UFRN, conforme conclui um dos gerentes de software entrevistados (GS).

[GS]: "*Eu estou vendo que essas métricas que estão sendo utilizadas elas são bem focadas para gestão mesmo. Então ajuda sim. O gestor vai ter a consciência tanto da contribuição do desenvolvedor, do acompanhamento da tarefa do desenvolvedor, de identificar as contribuições dele, de identificar se ele está se envolvendo, se aquele de repente está fazendo um pouquinho a mais ou a menos daquilo ali, para aumentar até a produção pra dele e da equipe, dele como desenvolvedor e da equipe. Eu enxergo dessa forma.*".

4.10.2 Dadas as melhorias propostas pelos gerentes para a avaliação da produtividade dos desenvolvedores, como as métricas podem contribuir para essas melhorias?

Esse questionamento visa realizar uma análise cruzada entre as métricas oferecidas pela abordagem de acompanhamento do presente estudo, com aquilo que os gerentes apontam como melhorias para a atual abordagem de avaliação da produtividade dos desenvolvedores. A abordagem proposta nesta dissertação e avaliada por nesse estudo, bem como suas métricas, não vão contribuir em algumas sugestões de melhorias apontadas pelos gerentes de software, como por exemplo (i) melhorias no iProject; (ii) concessão de mais tempo para o desenvolvedor realizar uma análise completa da demanda da tarefa, permitindo que ele possa compreender melhor o impacto dela; (iii) implementação de testes automatizados.

Um dos problemas da abordagem atual de avaliação da produtividade, apontado pelos gerentes de software, é a indefinição dos critérios de avaliação sob os quais desenvolvedores passam para a definição do bônus de produtividade. O uso das métricas oferecidas pela abordagem avaliada por esse estudo poderiam contribuir positivamente nesse sentido, pois, dessa forma, os desenvolvedores estariam cientes sobre quais atributos eles estariam sendo avaliados. O efeito negativo disso é que os desenvolvedores poderiam tentar adotar medidas para influenciar os valores das métricas em benefício próprio. Contudo, vale salientar que, conforme discutido anteriormente, a avaliação através das métricas deve estar acompanhada daquela realizada subjetivamente pelo gerente de software.

Na opinião dos gerentes entrevistados, as métricas poderiam oferecer importantes indicadores para o acompanhamento e avaliação de seus desenvolvedores. Alguns deles sugeriram mudanças nas métricas da abordagem avaliada nesse estudo, bem como a implementação de novas, contudo, essa discussão será feita mais a frente. A utilidade das métricas da abordagem são atestadas pelo interesse dos gerentes de software em utilizá-las em seus respectivos projetos. A maioria dos gerentes disse que gostaria de utilizar a maioria das métricas apresentadas. Apenas um gerente disse que usaria apenas as métricas relativas a volume de contribuição.

Ao ser questionado sobre possíveis mudanças na abordagem de avaliação de seus desenvolvedores, um dos gerentes (GD3) relatou que gostaria de utilizar pontos de função como medida de esforço de desenvolvimento. Esse interesse por métricas também pôde ser observado pelo uso do Sonar, que permite analisar o sistema desenvolvido sob a ótica de várias métricas de software, inclusive a complexidade ciclomática de McCabe. A ferramenta permite a identificação dos módulos do sistema que apresentam maiores valores de complexidade, contudo, não é capaz de indicar quem as introduziu, informação essa que pode ser obtida através das métricas avaliadas nesse estudo.

4.10.3 Na visão dos gerentes, de que forma a abordagem e as métricas podem ser melhoradas?

Ao conhecerem as métricas que compõem a abordagem de acompanhamento avaliada, os gerentes entrevistados fizeram várias sugestões de melhorias. Grande parte dessas sugestões visavam aperfeiçoar os resultados produzidos pelas métricas, principalmente através da redução da quantidade de falsos positivos. Outras sugestões foram dadas para melhor adequar as métricas à dinâmica de trabalho da SINFO.

As sugestões de melhorias dadas pelos gerentes serão apresentadas a seguir por métrica. Contudo, uma das sugestões não está associada a nenhuma métrica específica, e sim ao seu uso de forma geral. A sugestão dada pelo gerente (GR) é que as comparações dos valores das métricas apresentados pelos desenvolvedores sejam feitas considerando o cargo/nível de cada um. Ou seja, comparar desenvolvedores que pertençam ao mesmo cargo/nível, para que essa comparação seja realizada de forma justa. Essa é uma importante observação, contudo, o cruzamento das métricas de desenvolvedores da mesma equipe, mas pertencentes a diferentes cargos, pode ajudar na identificação de desenvolvedores de cargos inferiores com desempenho semelhante aos desenvolvedores de cargos superiores, e vice-versa. Isso pode dar o suporte adequado ao gerente de software no momento da decisão de promover algum de seus desenvolvedores.

Uma outra sugestão relativa a abordagem, e não as métricas em si, é a forma como os valores das métricas são apresentados. Conforme ilustrado na Figura 10, os valores das métricas são apresentados com cores de fundo para indicar a sua posição em relação aos valores de referência. Segundo um dos gerentes (GR), isso pode representar um problema para pessoas que possuem daltonismo, como por exemplo um dos gerentes de software entrevistados. Para resolver esse problema, o gerente sugeriu que os valores das métricas sejam acompanhados de símbolos, em vez de utilizar cores para tal fim.

4.10.3.1 Introdução de *Bugs*

Sem dúvida a métrica que, na visão dos gerentes, mais apresentou ameaças foi aquela relativa à introdução de *bugs*. A grande maioria dos gerentes acredita que a abordagem utilizada pela mineração para identificar o desenvolvedor que introduziu o *bug* pode apresentar muitos falsos positivos, pois eles acreditam que o desenvolvedor que de fato introduziu o *bug* nem sempre será o último a ter modificado o código que precisou de correção. Portanto vários gerentes alegaram que essa métrica iria apenas apontar os desenvolvedores que mais contribuem ou aqueles mais experientes na equipe. Dessa forma, a ameaça associada a essa métrica se torna maior do que seus benefícios, reduzindo a sua utilidade de forma significativa.

Para solucionar esse problema, alguns gerentes sugeriram que essa métrica utilizasse uma abordagem mais simplificada: contar quantas vezes cada tarefa voltou da equipe de testes com falhas, e associar esse número ao desenvolvedor responsável pela tarefa. Dessa forma, mesmo não havendo uma medida de introdução de *bugs*, ainda se teria uma medida de

qualidade do que foi produzido na tarefa. Segundo os gerentes, utilizando essa abordagem mais simplificada, a métrica apresentaria valores mais confiáveis, e ainda seria capaz de identificar casos em que os desenvolvedores não testam suas tarefas de forma apropriada, conforme relatou um dos gerentes entrevistados (GT). Essa mudança ainda contribuiria para resolver outras ameaças e limitações associadas a essa métrica, como o problema da classificação inadequada das tarefas, e o problema da temporalidade dos valores apresentados pela métrica, que podem se referir a *bugs* introduzidos há muito tempo atrás.

4.10.3.2 Contribuição em Correção de Erros

Durante a entrevista com um dos gerentes (GT), surgiu uma discussão interessante a respeito da métrica relativa à contribuição em correção de *bugs*. A métrica contabiliza, para cada desenvolvedor, a quantidade de *logs* de desenvolvimento em tarefas de correção de erro. Em sua resposta, o gerente de software apresentou dois cenários diferentes: um em que o desenvolvedor cadastra um *log* de desenvolvimento a cada pequeno incremento da tarefa; e outro cenário em que o desenvolvedor cadastra um único *log* ao concluir a tarefa.

Dessa forma, a métrica tende a beneficiar os desenvolvedores que cadastram um *log* a cada pequeno incremento, em detrimento aos desenvolvedores que trabalham da outra forma. Ao ser questionado se ele achava que seria mais justo fazer uma contagem por tarefas, conforme sugerido por outros gerentes, ele relatou um terceiro cenário: um desenvolvedor está trabalhando numa tarefa para a correção de um *bug* num módulo bastante complexo, e, ao concluí-la, cadastra um *log* de desenvolvimento e a envia para testes. Posteriormente, essa tarefa volta com erros de teste, e o desenvolvedor precisa realizar novas modificações, cadastrando mais um *log* de desenvolvimento, e enviando para testes novamente. Mais uma vez a tarefa retorna com erros, que força o programador a modificar o código e cadastrar outro *log* de desenvolvimento, e assim por diante, até que a tarefa passe pelos testes realizados pela equipe de qualidade.

Segundo o gerente de software, nesse cenário específico, é justo que a contagem seja realizada por *logs* de desenvolvimento, e não por tarefa, uma vez que essa tarefa, devido a sua complexidade, demandou uma contribuição muito mais significativa do que ele faria numa tarefa mais simples. Os três cenários apresentados por esse gerente de software (GT) mostram que sempre existem ameaças associadas ao se produzir métricas baseadas na contagem de *logs* ou de tarefas, e portanto cabe ao gerente de software, através de sua avaliação subjetiva, fazer uma análise mais completa e identificar se tais ameaças estão ocorrendo.

Outros dois gerentes de software (GD1 e GD3) também não concordaram com a abordagem de contar os *logs* de desenvolvimento em tarefas de erro. Segundo eles, essa métrica seria mais precisa e confiável se, em vez de contar os *logs* de desenvolvimento, contabilizasse os *logs* de dois tipos: solicitação de *update* em produção e atualização de banco de dados. O primeiro é usado quando a tarefa é concluída e o código deve ser colocado em produção. O segundo é usado quando a tarefa está concluída e precisam ser realizadas modificações no banco de dados.

Segundo esses gerentes, essa abordagem seria mais confiável, tendo em vista que apenas os desenvolvedores responsáveis pela tarefa cadastram tais *logs* no iProject, e também resolveria o problema de tarefas de correção urgentes que não possuem *logs* de desenvolvimento, e portanto não são contabilizadas pela atual abordagem da métrica.

Atualmente, os valores da métrica de contribuição em correção de *bugs* são apresentados de forma percentual, a fim de mostrar a contribuição de cada desenvolvedor em relação aos outros de sua equipe. Como, na SINFO, cada equipe possui uma célula de correção de erros, os desenvolvedores dessa célula tendem a apresentar maiores valores para essa métrica, mas isso, segundo os gerentes, apenas indica algo que eles já sabem. Portanto, eles sugerem que os valores dessa métrica sejam apresentados de forma absoluta, para permitir comparações entre células de correção de erros de diferentes equipes.

4.10.3.3 Volume de Contribuição

Apesar das métricas relativas a volume de contribuição terem apresentado bastante aceitação por parte dos gerentes entrevistados, elas ainda foram alvo de algumas sugestões de melhorias. A primeira sugestão foi que as métricas relativas ao volume de contribuição devam levar em consideração artefatos não-Java, como páginas JSP, *scripts* SQL, arquivos XML, etc. O fato das métricas não considerarem outros tipos de artefatos pode até ser considerado uma ameaça dessas métricas, tendo em vista que, durante o período analisado pelo gerente de software, um determinado desenvolvedor poderia ter contribuído apenas em artefatos que não são considerados pela métrica, e portanto passaria a falsa impressão de que ele teria baixa contribuição em volume de código.

Uma outra sugestão interessante e complementar a aquela discutida no parágrafo anterior, é apresentar os valores de volume de contribuição divididos por tipo de artefato. Ou seja, para cada desenvolvedor, haveria um valor para artefatos Java, outro para páginas JSP,

outro para *scripts* SQL, e assim por diante. Isso permitiria ao gerente de software analisar o esforço da contribuição do desenvolvedor através do tipo de artefato em que ele trabalhou.

4.10.3.4 Complexidade Média por Método

Outro conjunto de métricas que foram bastante aceitas pelos gerentes entrevistados foram aquelas relativas à complexidade média por método. Contudo, um gerente de software sugeriu que, em vez de as métricas considerarem a complexidade ciclomática de McCabe, deveriam considerar a notação do Grande-O. Segundo ele, o fato do desenvolvedor produzir código com alta complexidade ciclomática não seria motivo suficiente para que o gerente solicitasse que ele reajustasse a sua implementação.

Em contraponto, todos os outros gerentes reconheceram que as métricas de complexidade média por método podem contribuir para avaliação da qualidade do código produzido pelo desenvolvedor. Uma vez que, conforme discutido anteriormente e argumentado pelos próprios gerentes, a manutenção num código de complexidade alta custa mais do que num código de complexidade baixa.

Apesar de não ter sido apontado por nenhum gerente, uma limitação das métricas relativas à complexidade média por método já era conhecida: o fato da mineração desconsiderar os deltas de complexidade negativos pode representar uma limitação dessas métricas, tendo em vista que deixa de levar em conta refatorações e ajustes no código-fonte que podem ter impacto positivo no sistema.

Contudo, para que os valores de delta negativos de complexidade sejam considerados pelas métricas, é preciso investigar e testar como gerentes de software devem lidar com valores positivos e negativos de mesma métrica, a fim de que não seja realizada apenas uma soma de tais valores, gerando um valor que pode não representar a realidade.

4.10.4 Na visão dos gerentes, quais novas métricas poderiam ser implementadas?

Um gerente (GS) sugeriu a implementação de uma nova métrica que indicasse o tempo que cada tarefa passou em cada equipe, ou seja, suporte, requisitos, desenvolvimento e testes. Segundo ele, isso permitiria analisar as tarefas de forma temporal, permitindo identificar gargalos no processo de desenvolvimento, e analisar a necessidade de possíveis mudanças para melhorar o tempo de conclusão das tarefas. Contudo, tal métrica estaria um

pouco fora do contexto desta dissertação, porque ela estaria associada às tarefas, e não relativa à contribuição individual dos desenvolvedores, foco de estudo dessa dissertação de mestrado.

4.11 Ameaças à Validade do Estudo

Qualquer estudo empírico carrega consigo ameaças à validade de seus resultados e também limitações, contudo, reconhecer tais fatores é importante para a construção da pesquisa científica, e pode resultar em novos trabalhos futuros. O estudo de avaliação conduzido e discutido neste capítulo têm as seguintes ameaças e limitações:

(i) O estudo de avaliação da abordagem foi conduzido apenas com a participação de gerentes de projeto de software, pois a ela foi elaborada visando dar suporte a eles. Contudo, a fim de enriquecer os resultados do estudo e confrontar opiniões, é importante considerar também o *feedback* dos desenvolvedores de software, uma vez que eles serão aqueles analisados pela abordagem;

(ii) Apesar de analisar a contribuição individual do desenvolvedor através de métricas que quantificam quatro atributos, várias outras métricas poderiam ser utilizadas para considerar outros tipos de participação no desenvolvimento, como métricas relativas à coesão, acoplamento, quantidade de *features* implementadas, etc.;

(iii) Nenhuma das métricas que compõem a abordagem analisam aspectos humanos ou sociais do desenvolvedor, sendo esse um dos motivos pela qual ela não pode ser utilizada pelos gerentes de software de forma exclusiva;

(iv) As métricas propostas também não avaliam o desenvolvedor no que diz respeito à integração e envolvimento com a sua equipe. Esse tipo de avaliação também é importante pois, entre as qualidades desejáveis em desenvolvedores apontadas pelos gerentes de software, estão algumas relativas a integração com a sua equipe, como por exemplo a capacidade de comunicação e o auxílio entre desenvolvedores.

4.12 Sumário

Este capítulo apresentou o estudo de avaliação da abordagem proposta por esta dissertação, conduzido através de entrevistas com sete gerentes de projeto da SINFO. Foram apresentados o contexto onde o estudo foi aplicado, o planejamento, os participantes entrevistados, os objetivos a serem atingidos, as questões de pesquisa a serem respondidas, a

execução da abordagem que precedeu a coleta de dados, a coleta de dados em si, os resultados, a análise dos resultados, as discussões, as ameaças a validade dos resultados e as suas limitações.

5 TRABALHOS RELACIONADOS

Este capítulo apresenta e discute os trabalhos relacionados a esta dissertação de mestrado. Todos os trabalhos apresentados a seguir também propõem métricas baseadas em mineração de repositórios para analisar a contribuição individual dos desenvolvedores. Logo após a breve apresentação do trabalho, é feita uma comparação entre os resultados alcançados por eles e os resultados desta dissertação.

5.1.1 *Measuring developer contribution from software repository data*

(GOUSIOS; KALLIAMVAKOU; SPINELLIS, 2008) apresentam uma abordagem de medição da contribuição individual dos desenvolvedores que aproveita a disponibilidade de repositórios de software. A proposta da abordagem é levantar indicadores que sejam gerados automaticamente, sem a necessidade de intervenção humana.

O modelo apresentado pelo trabalho leva em consideração diversas ações de contribuição. Ou seja, as minerações buscam por diversos tipos de ações realizadas pelos desenvolvedores nos repositórios, e cada uma dessas ações podem ser contabilizadas positivamente ou negativamente para o cálculo do fator de contribuição final do desenvolvedor. Ações como a produção de código-fonte, correção de um *bug* ou publicação de uma nova página Wiki são contabilizadas como positivas. Ações como concluir uma tarefa para correção de um *bug*, que posteriormente precisou ser reaberta, realizar um *commit* sem comentário ou realizar um *commit* que gerou um bug, são contabilizadas como ações negativas.

Além disso, o modelo permite que pesos sejam considerados para cada ação, tendo em vista que as ações não têm o mesmo grau de importância para o projeto. Os valores dos pesos de cada ação são independentes do modelo, a fim de refletir particularidades de cada projeto de software. O modelo sugere realizar um somatório entre valores de diferentes métricas, com a finalidade de se obter um valor que represente a contribuição individual do desenvolvedor. Contudo, somar valores de métricas heterogêneas pode não ser adequado, tendo em vista que elas quantificam diferentes atributos de contribuição.

O estudo não aborda como tais métricas podem ser usadas pelos gerentes de software na avaliação da contribuição individual de seus desenvolvedores, ficando a cargo do gerente a

interpretação dos valores apresentados pelas métricas, pois o modelo não oferece valores de referência e nem meios para a definição de tais limiares.

Esta dissertação de mestrado ofereceu não apenas métricas que permitem a avaliação da contribuição individual, mas também meios para a definição de valores de referência, que auxiliam na interpretação das métricas, dando suporte à tomada de decisão. A abordagem também oferece uma técnica para identificação e descarte de *outliers*, a fim de manter a coerência e a qualidade dos dados a serem analisados.

O estudo de avaliação do modelo proposto por (GOUSIOS; KALLIAMVAKOU; SPINELLIS, 2008) aplicou as métricas em 48 sub-projetos do projeto GNOME. As métricas extraíram dados de 17 tipos de ações, sendo 14 tipos consideradas positivas e 3 tipos consideradas negativas, em repositórios de código e de e-mails, desde o início do projeto até janeiro de 2009. Portanto, apesar das métricas oferecidas pelo modelo serem voltadas para avaliação da contribuição individual, o estudo de avaliação conduzido não se concentrou na análise do *feedback* de gerentes de projeto a respeito da utilidade de tais métricas em seus respectivos ambientes de trabalho.

5.1.2 Avaliação da contribuição de desenvolvedores para projetos de software usando mineração de repositórios de software e mineração de processos

(COSTA et al., 2014) utilizaram técnicas de mineração de repositórios e mineração de processos para avaliação da contribuição de desenvolvedores de software. As métricas propostas no estudo avaliam a contribuição do desenvolvedor sob três perspectivas diferentes: (i) *commits* que introduziram *bugs*; (ii) contribuição em código; e (iii) correção de *bugs* prioritários.

O estudo de avaliação foi conduzido através de quatro atividades: (i) seleção dos projetos; (ii) classificação dos desenvolvedores através da mineração de processos; (iii) mineração de repositórios de software; e (iv) análise dos resultados obtidos. A primeira atividade, seleção dos projetos, visou definir em quais contextos o estudo de avaliação seria aplicado. O resultado foi a escolha de um projeto *open-source*, o ArgoUML, e uma empresa privada de desenvolvimento de sistemas.

A segunda atividade da avaliação foi a classificação dos desenvolvedores em papéis de acordo com as ações que eles desempenham. Tal classificação é realizada através de uma técnica de mineração de processos baseada no trabalho proposto por (PONCIN;

SEREBRENIK; VAN DEN BRAND, 2011), utilizando papéis estabelecidos em (NAKAKOJI et al., 2002), ou seja, desenvolvedor *core*, periférico ou ativo. A classificação é realizada considerando que cada desenvolvedor é um processo, e a classificação em papéis é realizada conforme seu padrão de ações durante o projeto.

A terceira atividade da avaliação foi minerar, através do Backhoe, os repositórios de software em busca de eventos que reflitam as três perspectivas de contribuição citadas anteriormente. Ao todo, 17.490 *commits* e 10.308 *bugs* foram analisados nos dois contextos. O quarto e último passo visou analisar os dados com o objetivo de responder três questões de pesquisa: (i) Que tipo de desenvolvedor (*core*, periférico ou ativo) produz introduz menos *bugs*? (ii) Que tipo de desenvolvedor (*core*, periférico ou ativo) tem mais contribuição em código? (iii) Que tipo de desenvolvedor (*core*, periférico ou ativo) corrige mais *bugs* prioritários?

Os resultados do estudo de avaliação concluem que: (i) os grupos de desenvolvedores (*core*, periférico e ativo) apresentam diferenças estatisticamente significativas em relação aos valores apresentados pelas métricas; (ii) no projeto *open-source*, os desenvolvedores *core* tem maior contribuição em código e introduzem proporcionalmente menos *bugs* no sistema; (iii) um novo grupo de desenvolvedores foi encontrado, que apresentam uma menor proporção de introdução de *bugs* em comparação com os desenvolvedores periféricos, mas também apresentam uma proporção maior quando comparados com os desenvolvedores *core*.

É interessante notar que, apesar das métricas que compõem a abordagem proposta nesta dissertação se basearem nas métricas propostas por (COSTA et al., 2014), o estudo de avaliação conduzido por eles buscou identificar quais grupos de desenvolvedores apresentam melhores resultados para as métricas, enquanto que o estudo de avaliação conduzido nesta dissertação visa compreender a utilidade e aplicabilidade das métricas, acompanhadas das técnicas de definição de valores de referência e identificação de *outliers*, num contexto de gerência de projetos de software.

Portanto, o estudo de avaliação conduzido nesta dissertação difere daquele apresentado por (COSTA et al., 2014), uma vez que a avaliação feita nesta dissertação analisa se as métricas poderiam ser usadas da forma como foram usadas no estudo conduzido por eles. Enquanto os resultados apresentados por eles indicam que desenvolvedores *core* apresentam, relativamente, menos *bug commits* em relação aos outros grupos, os resultados do estudo de avaliação desta dissertação apontam que os gerentes de software entrevistados não

usariam essa métrica com confiança, haja vista a quantidade de falsos-positivos que ela poderia produzir.

O estudo de avaliação conduzido por (COSTA et al., 2014) não validou suas métricas enquanto indicadores para suporte à tomada de decisão pelos gerentes de software. O estudo também não tratou sobre como valores de referência desses indicadores poderiam ser produzidos e utilizados. Além disso, as perspectivas de avaliação da contribuição individual não consideram qualquer medida de complexidade da contribuição.

5.1.3 *Samiksha: mining issue tracking system for contribution and performance assessment*

(RASTOGI; GUPTA; SUREKA, 2013) propuseram um *framework*, chamado Samiksha, para avaliação da contribuição e desempenho individual de profissionais de manutenção de software. Esse *framework* possui onze métricas que avaliam diferentes atributos de contribuição de quatro papéis: *bug reporter*, *bug triager*, *bug owner* e *collaborator*. As métricas são produzidas através da extração de dados do *Issue Tracking System* do projeto.

O trabalho elenca três contribuições principais: (i) o *framework* de avaliação da contribuição de profissionais de manutenção de software; (ii) a condução de um *survey* com a participação de dois profissionais de manutenção; e (iii) a análise empírica do *framework* utilizando o *Issue Tracking System* do projeto Google Chromium. O *survey* visa compreender (i) a importância dos atributos capturados pelas métricas propostas no trabalho; e (ii) quão precisas são as técnicas empregadas por esses profissionais para capturar esses atributos atualmente (sem o uso das métricas). Ele foi realizado aplicando um questionário com dois profissionais de manutenção de software com mais de dez anos de experiência. O questionário era dividido em quatro partes, cada uma com questões relativas a um dos quatro papéis citados anteriormente.

De forma geral, os resultados apresentados no *survey* denotam a importância da avaliação dos atributos capturados pelas métricas, e a pouca precisão que os profissionais dispõem atualmente para capturar e avaliar tais atributos. Tal resultado também pôde ser constatado através das entrevistas realizadas nesta dissertação: há um reconhecimento da importância dos atributos avaliados pelas métricas, mas que a abordagem de avaliação atual não consegue captura-los e avalia-los.

O outro estudo de avaliação realizado pelo trabalho foi a condução de um experimento para avaliação de suas métricas, coletando dados de um ano do *Issue Tracker* do Google Chromium[20]. No total, foram coletados 24.743 *bugs*, reportados por 6.765 desenvolvedores. Tais *bugs* passaram por uma triagem, realizada por 500 desenvolvedores. Os *bugs* foram distribuídos para 391 desenvolvedores (*bug owners*). Os *bugs* foram corrigidos com a colaboração de 5.043 desenvolvedores.

Apesar da grande variedade de métricas que compõem *framework*, e dos resultados apresentados pelos estudos de avaliação, as métricas do Samiksha estão centradas nas atividades relativas à manutenção de software, sobretudo na correção de *bugs*, desconsiderando outros atributos de contribuição, como a contribuição em código-fonte ou complexidade ciclomática. O *framework* também não dispõe sobre valores de referência de tais métricas, que poderiam ajudar seus usuários na interpretação, oferecendo suporte adequado à tomada de decisão.

5.1.4 *Rank-me: A Java tool for ranking team members in software bug repositories*

(NAGWANI; VERMA, 2012) apresentam uma técnica para classificação dos desenvolvedores de uma equipe, com base em informações mineradas através de suas contribuições na correção de *bugs*. A classificação é baseada em quatro métricas que são produzidas através de mineração de repositórios de *bugs*. São elas: *fix score*, *comment score*, *reported score* e *CC mailing score*.

Fix score quantifica o número de correções de *bugs* realizadas por um desenvolvedor, levando em consideração o grau de prioridade e severidade de cada *bug*. *Comment score* contabiliza o número de comentários deixados por um desenvolvedor. *Reported score* mede o número de *bugs* reportados por um desenvolvedor. *CC mailing score* quantifica o número de vezes que um desenvolvedor recebeu uma mensagem. Os quatro *scores* de cada desenvolvedor são somados e normalizados para se obter um número que reflita a contribuição do desenvolvedor em correção de *bugs*. Em seguida, um ranking dos desenvolvedores pode ser gerado com base nas pontuações de cada um.

[20] https://code.google.com/p/chromium/issues

Um estudo de avaliação da ferramenta foi conduzido utilizando o repositório de *bugs* do projeto Mozilla, no qual foram extraídos dados de cerca de mil *bugs*, que foram resolvidos por 97 desenvolvedores. Os resultados produzidos pelo estudo foram dois *rankings* com os desenvolvedores que mais pontuaram nas quatro métricas. A análise realizada nesse estudo de avaliação, porém, foi puramente funcional, e portanto carece validação da aplicabilidade e da utilidade das métricas num contexto de gerencia de projetos de software, como foi conduzido por esta dissertação de mestrado.

Além disso, assim como no *framework* apresentado por (RASTOGI; GUPTA; SUREKA, 2013), as métricas propostas por esse trabalho também estão centradas nas contribuições relativas à correção de *bugs*, deixando de considerar outros atributos de contribuição importantes, como volume de código produzido e a complexidade ciclomática. O trabalho também não aborda sobre a definição de valores de referência para suas métricas.

5.2 Sumário

Este capítulo apresentou os trabalhos relacionados com esta dissertação de mestrado. Os trabalhos apresentados têm relação com esta dissertação devido às suas propostas de métricas baseadas em mineração de repositórios, que tem a finalidade de analisar a contribuição individual dos desenvolvedores. Além da breve apresentação dos trabalhos relacionados, foi feita uma comparação entre os resultados alcançados por estes trabalhos e os resultados obtidos por esta dissertação.

6 CONCLUSÃO

Este capítulo apresenta as considerações finais do estudo de avaliação realizado, bem como elenca as principais contribuições realizadas e discute as perspectivas de trabalhos futuros.

6.1 Considerações finais

Esta dissertação de mestrado propôs uma abordagem de análise da contribuição individual de desenvolvedores, a qual é composta por: (i) métricas de contribuição baseadas em mineração de repositórios; (ii) uma técnica de identificação de *outliers*; e (iii) uma técnica de definição de valores de referência. As métricas apresentadas são baseadas em uma suíte proposta por (COSTA et al., 2014), entretanto, duas novas métricas foram elaboradas e incorporadas à abordagem, as quais têm como objetivo avaliar a complexidade ciclomática do código que faz parte da contribuição de um desenvolvedor, e permitem que o gerente de software faça uma avaliação técnica relativa à qualidade do código produzido pelo desenvolvedor.

A dissertação apresentou também um estudo de avaliação da abordagem conduzido através de entrevistas semi-estruturadas em um contexto real de desenvolvimento de software, com a participação de sete gerentes de projeto, que deram o seu depoimento a respeito da abordagem, sobretudo as métricas, que representam o seu cerne. As opiniões dos gerentes variaram em relação às métricas, contudo, foi possível concluir que as métricas avaliadas podem ser usadas para complementar a avaliação subjetiva realizada pelos gerentes de projeto, não podendo ser utilizadas de forma isolada, uma vez que elas não são capazes de avaliar outros aspectos que não são quantificados por elas, como participação em reuniões, treinamentos, suporte a outros desenvolvedores, etc.

De acordo com os resultados do estudo conduzido, as métricas trazem diferentes níveis de benefícios e limitações. As métricas relativas ao volume de contribuição tiveram bastante aceitação dos gerentes de software, que alegaram apenas que elas devam também contabilizar linhas de código em outros artefatos de implementação, que não apenas classes e interfaces Java. A métrica relativa a contribuição em correção de erros também teve relativa aceitação, contudo, o fato dela contabilizar a quantidade de *logs* de desenvolvimento pode trazer algumas ameaças, conforme a opinião dada por alguns gerentes. As métricas relativas à complexidade média por método também tiveram bastante aceitação por parte dos gerentes.

Apenas um deles alegou que ela deveria considerar a análise da complexidade do algoritmo implementado por um dado método em vez da complexidade ciclomática de McCabe.

A métrica mais criticada pelos gerentes entrevistados foi aquela relativa à introdução de *bugs*, devido a forma como a mineração identifica o autor do *bug*. Alguns gerentes sugeriram que ela deveria ser substituída por uma simples contagem de quantas vezes cada tarefa do desenvolvedor voltou da equipe de qualidade com erros de teste.

Houve também opiniões relativas à abordagem, em especial na forma como os valores das métricas devem ser analisados, como por exemplo a inclusão dos cargos dos desenvolvedores para permitir uma comparação mais justa entre eles. A implementação de tal melhoria na abordagem requer apenas que os valores relacionados a desenvolvedores que tenham o mesmo cargo em uma instituição sejam agregados e comparados diretamente, não trazendo, portanto, qualquer dificuldade para a adaptação da abordagem para incorporá-la. Outra consideração importante, que reforça a aceitação que a abordagem teve junto aos gerentes, é que, após a realização das entrevistas, na qual a equipe de gerência da SINFO teve a oportunidade de conhecer melhor a abordagem que estava sendo avaliada, reuniões foram marcadas para discutir como a abordagem pode ser implantada de forma integrada à ferramenta de gerência de tarefas (*issues*) da instituição, denominada iProject. Durante uma dessas reuniões, um gestor informou que um novo desenvolvedor estava sendo contratado para ficar responsável pelas tarefas de melhorias no iProject, e entre elas está a implantação da abordagem, em especial das métricas.

Uma segunda reunião foi também agendada com a participação dos gestores e gerentes de projeto da SINFO, para discutir quais e como eles desejam que as métricas sejam incorporadas ao iProject, tendo em vista os benefícios e limitações de cada uma delas.

6.2 Contribuições

A presente dissertação de mestrado tem como base, entre outros estudos, o trabalho realizado por (COSTA, 2012). Dessa forma é importante deixar claro quais foram as contribuições desta dissertação, que complementaram as contribuições realizadas anteriormente pelo estudo de (COSTA, 2012).

(i) Proposta de uma abordagem para análise da contribuição individual de desenvolvedores, composta por (a) um conjunto de métricas baseadas em

mineração de repositórios; (b) uma técnica de identificação e descarte de *outliers*; e (c) uma técnica de definição de valores de referência;

(ii) Melhorias realizadas nas minerações e nas métricas originais propostas no Backhoe (COSTA, 2012). Conforme apresentado na Seção 3.2.1, as melhorias visavam principalmente a redução da quantidade de falsos positivos apresentados pelas métricas, bem como atenuar suas limitações e melhorar a sua performance;

(iii) Foram elaboradas duas métricas para avaliação da complexidade média por método realizada pelo desenvolvedor. Essas métricas permitem ao gerente de projeto analisar a qualidade da contribuição do desenvolvedor;

(iv) Condução de um estudo de avaliação da abordagem proposta, como ferramenta de acompanhamento da contribuição individual dos desenvolvedores. Tal estudo foi realizado dentro de um contexto real de produção de software, com a participação de sete gerentes de diferentes áreas, através de entrevistas semi-estruturadas, nas quais eles puderam dar suas opiniões a respeito da abordagem e das métricas;

(v) O estudo permitiu identificar os benefícios, limitações, ameaças e desafios associados às métricas, bem como permitiu também compreender sobre como elas poderiam ser melhoradas a fim de oferecer resultados mais úteis e confiáveis à gerência de software. O estudo também permitiu aos gerentes da SINFO, bem como seus gestores, analisar o seu processo de desenvolvimento em busca de problemas e possíveis soluções. Uma das soluções encontradas para melhorar o acompanhamento da contribuição realizada pelos desenvolvedores foi a incorporação da abordagem na ferramenta de gerência de tarefas (*issues*), denominada iProject;

(vi) Entre os resultados alcançados pelo estudo está a compreensão sobre quais são as qualidades desejáveis em bons desenvolvedores, e a constatação de que as métricas propostas na abordagem não são capazes de analisar várias dessas qualidades apontadas pelos gerentes.

6.3 Trabalhos futuros

Esta seção discute sobre perspectivas de trabalhos futuros que podem ser conduzidos a partir do estudo apresentado por essa dissertação.

(i) Analisar como algoritmos de clusterização, como o K-Means, poderia ser utilizado para agregar os valores de métricas distintas, a fim de se agrupar os desenvolvedores que tiveram desempenho similares;

(ii) Conduzir estudos de avaliação sobre como a gamificação pode influenciar o desempenho dos desenvolvedores de software, as métricas poderiam ser utilizadas para a produção automática de *badges*/*achievements*, recompensando os desenvolvedores que, por exemplo, produzissem mais código-fonte, ou corrigissem mais *bugs*, etc;

(iii) Conduzir estudos para determinar como as métricas podem ser interpretadas de forma conjunta e complementar;

(iv) Conduzir um estudo de avaliação para determinar se as métricas relativas ao volume de contribuição seriam mais úteis se considerassem abstrações de módulos utilizados na implementação do sistema. Por exemplo: indicar a quantidade de linhas de código adicionadas, modificadas e removidas em um MBean;

(v) Conduzir estudos para investigar e discutir como a medição através de métricas pode influenciar o comportamento das pessoas.

REFERÊNCIAS

ALVES, T. L.; YPMA, C.; VISSER, J. **Deriving metric thresholds from benchmark data**. Software Maintenance (ICSM), 2010 IEEE International Conference on. 2010

ANSELMO, D.; LEDGARD, H. Measuring productivity in the software industry. **Communications of the ACM**, v. 46, n. 11, p. 121–125, 2003.

BOEHM, B. W. **Improving software productivity**. Computer. 1987

CORBIN, J.; STRAUSS, A. **Basics of qualitative research: Techniques and procedures for developing grounded theory**. [s.l.] Sage, 2008.

COSTA, D. **Avaliação da Contribuição de Desenvolvedores para Projetos de Software usando Mineração de Repositórios de Software e Mineração de Processos**. [s.l.] UFRN, 2012.

D'AMBROS, M. et al. **Analysing software repositories to understand software evolution**. [s.l.] Springer, 2008.

DA COSTA, D. A. et al. **Unveiling Developers Contributions Behind Code Commits: An Exploratory Study**. Proceedings of the 29th Annual ACM Symposium on Applied Computing: SAC'14. New York, NY, USA: ACM, 2014. Disponível em: <http://doi.acm.org/10.1145/2554850.2555030>

DE AQUINO, G. S.; DE LEMOS MEIRA, S. R. **Towards Effective Productivity Measurement in Software Projects**. Software Engineering Advances, 2009. ICSEA'09. Fourth International Conference on. 2009

DE O MELO, C. et al. Interpretative case studies on agile team productivity and management. **Information and Software Technology**, v. 55, n. 2, p. 412–427, 2013.

DUNCAN, A. S. **Software development productivity tools and metrics**. Proceedings of the 10th international conference on Software engineering. 1988

FENTON, N. E.; NEIL, M. **Software metrics: roadmap**. Proceedings of the Conference on the Future of Software Engineering. 2000

FENTON, N. E.; PFLEEGER, S. L. **Software metrics: a rigorous and practical approach**. [s.l.] PWS Publishing Co., 1998.

GOUSIOS, G.; KALLIAMVAKOU, E.; SPINELLIS, D. **Measuring developer contribution from software repository data**. Proceedings of the 2008 international working conference on Mining software repositories. 2008

HAN, J.; KAMBER, M. **Data Mining, Southeast Asia Edition: Concepts and Techniques**. [s.l.] Morgan kaufmann, 2006.

HASSAN, A. E. **Mining software repositories to assist developers and support managers**. Software Maintenance, 2006. ICSM'06. 22nd IEEE International Conference on. 2006

HASSAN, A. E. **The road ahead for mining software repositories**. Frontiers of Software Maintenance, 2008. FoSM 2008. 2008

JONES, T. C. Measuring programming quality and productivity. **IBM Systems Journal**, v. 17, n. 1, p. 39–63, 1978.

KITCHENHAM, B.; MENDES, E. Software productivity measurement using multiple size measures. **Software Engineering, IEEE Transactions on**, v. 30, n. 12, p. 1023–1035, 2004.

MACCORMACK, A. et al. Trade-offs between productivity and quality in selecting software development practices. **IEEE Software**, v. 20, n. 5, p. 78–85, 2003.

MCCABE, T. J. A complexity measure. **Software Engineering, IEEE Transactions on**, n. 4, p. 308–320, 1976.

MENZIES, T.; ZIMMERMANN, T. Software Analytics: So What? **Software, IEEE**, v. 30, n. 4, p. 31–37, 2013.

NAGWANI, N. K.; VERMA, S. Rank-me: A java tool for ranking team members in software bug repositories. **Journal of Software Engineering and Applications**, v. 5, p. 255, 2012.

NAKAKOJI, K. et al. **Evolution patterns of open-source software systems and communities**. Proceedings of the international workshop on Principles of software evolution. 2002

OLIVEIRA, P. et al. **Metrics-based Detection of Similar Software**. INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING AND KNOWLEDGE ENGINEERING. Proceedings. 2013

OLIVEIRA, P.; VALENTE, M. T.; LIMA, F. P. **Extracting relative thresholds for source code metrics**. Software Maintenance, Reengineering and Reverse Engineering (CSMR-WCRE), 2014 Software Evolution Week-IEEE Conference on. 2014

OMG. **BPMN Specification - Business Process Model and Notation**. Disponível em: <http://www.bpmn.org/>. Acesso em: 18 nov. 2014.

PETERSEN, K. Measuring and predicting software productivity: A systematic map and review. **Information and Software Technology**, v. 53, n. 4, p. 317–343, 2011.

PONCIN, W.; SEREBRENIK, A.; VAN DEN BRAND, M. **Process mining software repositories**. Software Maintenance and Reengineering (CSMR), 2011 15th European Conference on. 2011

PREMRAJ, R. et al. **An empirical analysis of software productivity over time**. Software Metrics, 2005. 11th IEEE International Symposium. 2005

RAMASUBBU, N. et al. **Configuring global software teams: a multi-company analysis of project productivity, quality, and profits**. Proceedings of the 33rd International Conference on Software Engineering. 2011

RASTOGI, A.; GUPTA, A.; SUREKA, A. **Samiksha: mining issue tracking system for contribution and performance assessment**. Proceedings of the 6th India Software Engineering Conference. 2013

SINFO. **Superintendência de Informática da UFRN**. Disponível em: <http://www.info.ufrn.br/>. Acesso em: 22 jun. 2014a.

SINFO. **Sucupira**. Disponível em: <http://www.capes.gov.br/avaliacao/plataforma-sucupira>. Acesso em: 22 jun. 2014b.

SOMMERVILLE, I. et al. **Engenharia de software**. [s.l.] Addison Wesley, 2007. v. 8

TRENDOWICZ, A.; MÜNCH, J. Factors Influencing Software Development Productivity—State-of-the-Art and Industrial Experiences. **Advances in computers**, v. 77, p. 185–241, 2009.

YOON, K.-A.; KWON, O.-S.; BAE, D.-H. **An approach to outlier detection of software measurement data using the k-means clustering method**. Empirical Software Engineering and Measurement, 2007. ESEM 2007. First International Symposium on. 2007

APÊNDICE A: QUESTÕES DA ENTREVISTA

As perguntas da entrevista foram divididas em quatro partes, que visaram atingir diferentes objetivos, e também em questões principais e de suporte. As questões principais tem maior importância em relação as questões de suporte, e portanto devem ser feitas aos entrevistados. As questões de suporte têm menor importância e servem para o entrevistador reformular a pergunta, ou refazê-la quando entender que o entrevistado não respondeu a questão principal. As questões principais estão marcadas em negrito.

1ª Parte: Identificação de perfil

1) **Há quanto tempo você trabalha na SINFO?**
2) **Há quanto tempo você é gerente de software na SINFO?**
3) **Você trabalhou como gerente de software em outra empresa? Se sim, por quanto tempo?**

2ª Parte: Como a avaliação da contribuição dos desenvolvedores é feita atualmente?

1) Quem são seus melhores desenvolvedores?
2) **Por que <nome do desenvolvedor> é um dos seus melhores desenvolvedores?**
3) O que faz um desenvolvedor excelente? Você poderia me dar um exemplo da sua equipe?
4) Como você mede o quão bom um desenvolvedor é?
5) **Você poderia me dar um exemplo de como você avalia o desempenho dos seus desenvolvedores? Como você mede o quão bem eles vão?**
6) Que tipo de coisas você observa ou acompanha? Como você tem certeza que seus desenvolvedores estão indo bem?
7) Quais técnicas ou processos são usados para analisar a contribuição individual dos seus desenvolvedores?
8) Quais ferramentas ou dados são usados para acompanhar a contribuição individual dos seus desenvolvedores?
9) **Há algum problema com a abordagem que você mencionou? Se sim, quais?**
10) **Atualmente você usa alguma métrica para avaliar seus desenvolvedores?**

11) Você usa alguma informação ou métrica fornecida através de gerenciadores de tarefas, *issue tracking systems*, ou sistema de controle de versão para acompanhar a contribuição dos seus desenvolvedores? Se sim, quais informações específicas?
12) A SINFO tem uma política de bônus salarial que se baseia no desempenho individual? Como ela funciona?
13) **Em quais situações um desenvolvedor pode receber uma bonificação menor? Em quais casos ele recebe um bônus maior?**
14) **Como você calcula o bônus a ser recebido por cada desenvolvedor? Quais ferramentas você usa para isso?**

3ª Parte: Como a avaliação da contribuição dos desenvolvedores poderia ser melhorada?

15) **Como você gostaria de medir o desempenho dos seus desenvolvedores no futuro?**
16) Quais tipos de dados, ferramentas ou técnicas você gostaria de ter para medir a produtividade dos seus desenvolvedores?
17) **Quais métricas você acredita que seriam úteis para avaliar a contribuição individual de seus desenvolvedores?**

4ª Parte: Quão útil é a abordagem para avaliar a contribuição individual dos desenvolvedores?

18) **Essa métrica faz sentido para você?**
19) **O que essa métrica significa para você?**
20) **Que tipo de ameaças ou perigos existem ao interpretar esses dados?**
21) **Quais desenvolvedores tiveram o desempenho esperado para essa métrica? Por que?**
22) **Quais desenvolvedores não tiveram o desempenho esperado para essa métrica? Por que?**
23) Na sua equipe existem desenvolvedores responsáveis pela correção de erros? Se sim, as métricas relativas à correção de erros apontam melhor desempenho para esses desenvolvedores?

5ª Parte: Análise geral da utilidade e aplicabilidade das métricas

24) **Os resultados das métricas lhe ajudam a compreender o que aconteceu durante a semana?**
25) **Quais benefícios você acredita que essas informações/métricas trazem para aferir a produtividade de seus desenvolvedores?**
26) **Quais limitações você acredita que essas informações/métricas trazem para aferir a produtividade de seus desenvolvedores? Você acha que elas podem ser adaptadas para ajudar a gerenciar seu projeto?**
27) **Você gostaria de usar essas métricas como forma de apoio a medição de produtividade de seus desenvolvedores?**

APÊNDICE B: GRÁFICOS DE DISPERSÃO USADOS NA DEFINIÇÃO DOS VALORES DE REFERÊNCIA

SIGAA

Figura 11 - Gráfico de dispersão usado na definição dos valores de referência para a métrica de Introdução de Bugs

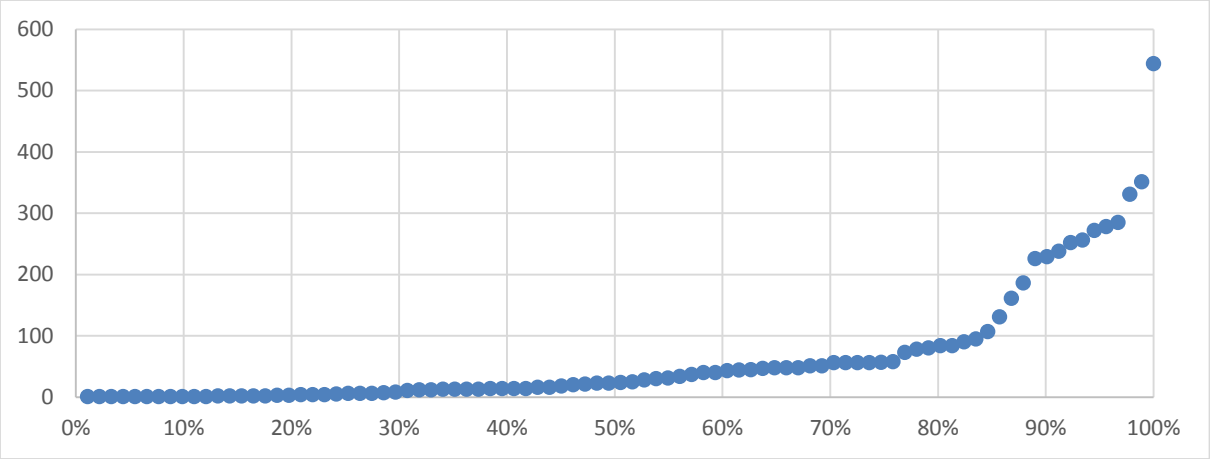


Figura 12 - Gráfico de dispersão usado na definição dos valores de referência para a métrica de Correção de Erros

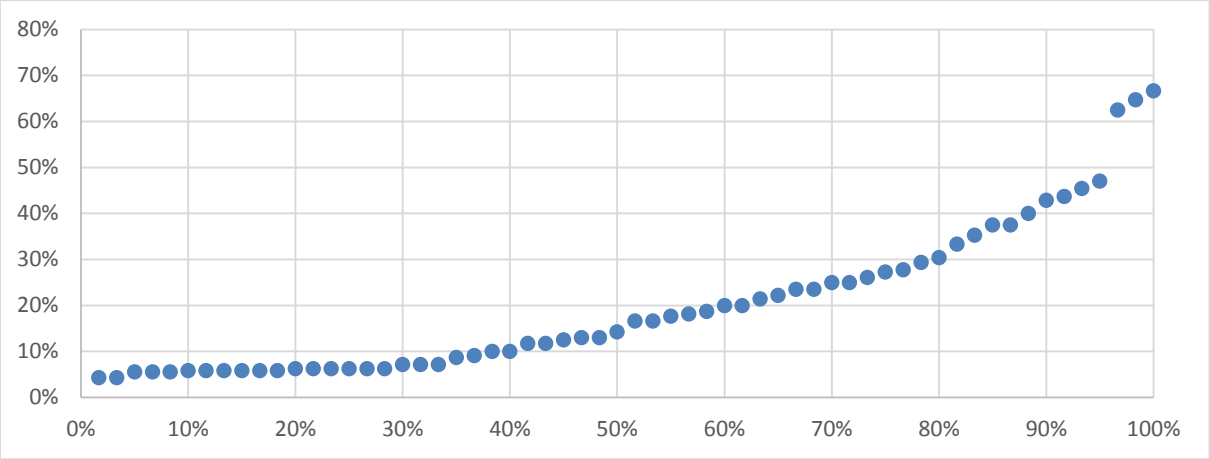


Figura 13 - Gráfico de dispersão usado na definição dos valores de referência da métrica de Complexidade Média em Métodos Adicionados

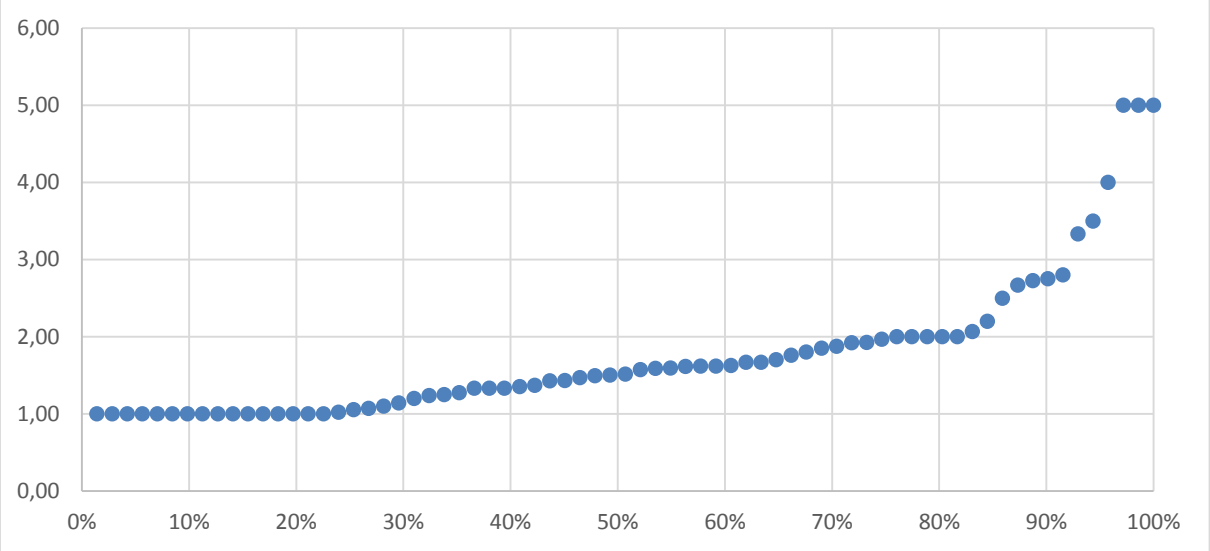


Figura 14 - Gráfico de dispersão usado na definição dos valores de referência para a métrica de Complexidade Média em Métodos Modificados

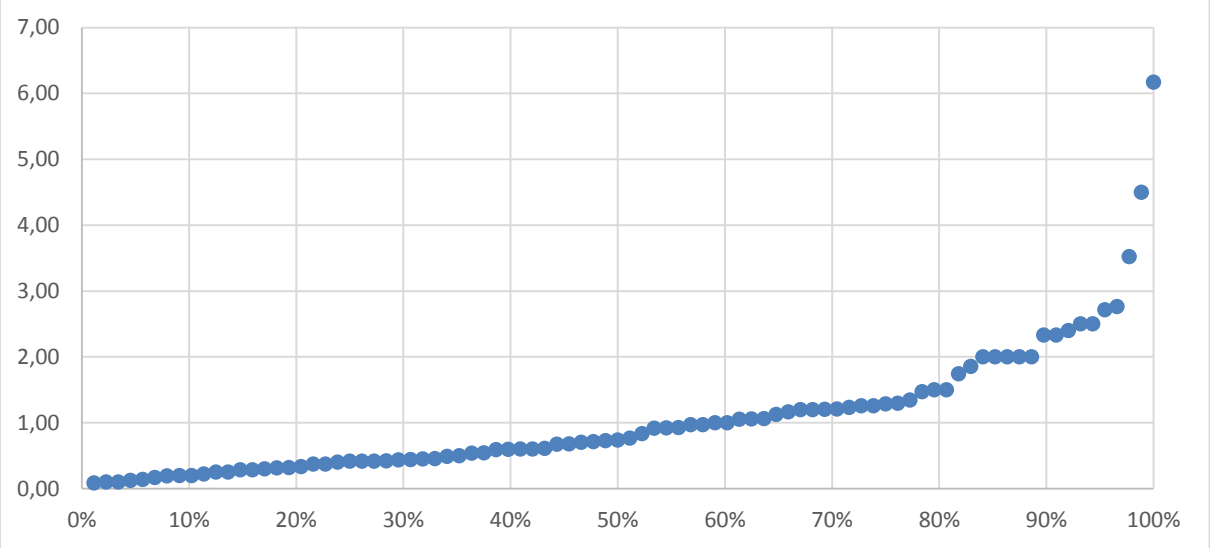
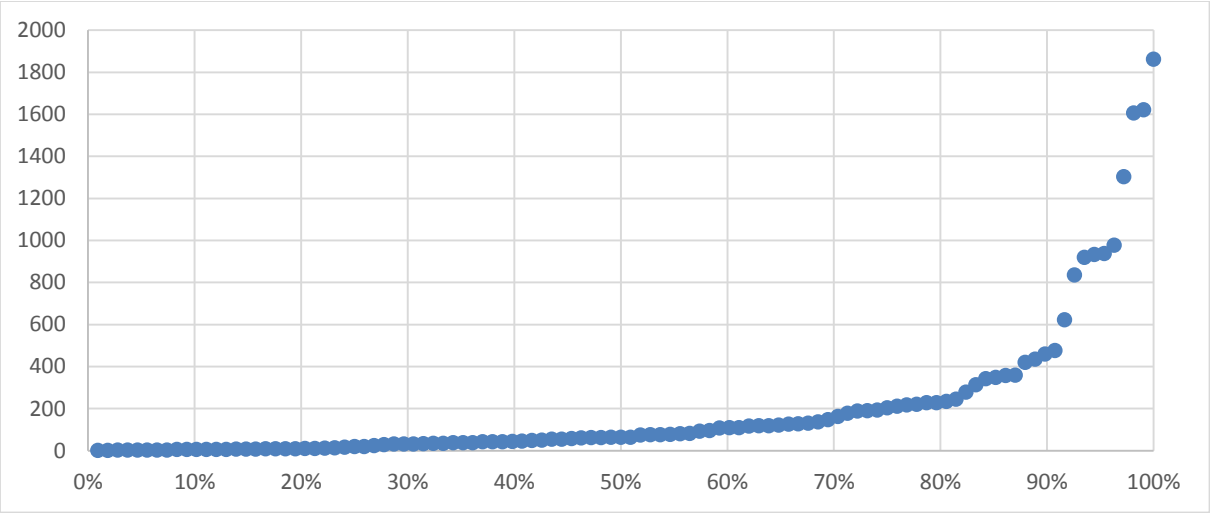


Figura 15 - Gráfico de dispersão usado na definição dos valores de referência da métrica de Volume de Contribuição



SIGRH

Figura 16 – Gráfico de dispersão usado na definição dos valores de referência da métrica de Introdução de *Bugs*

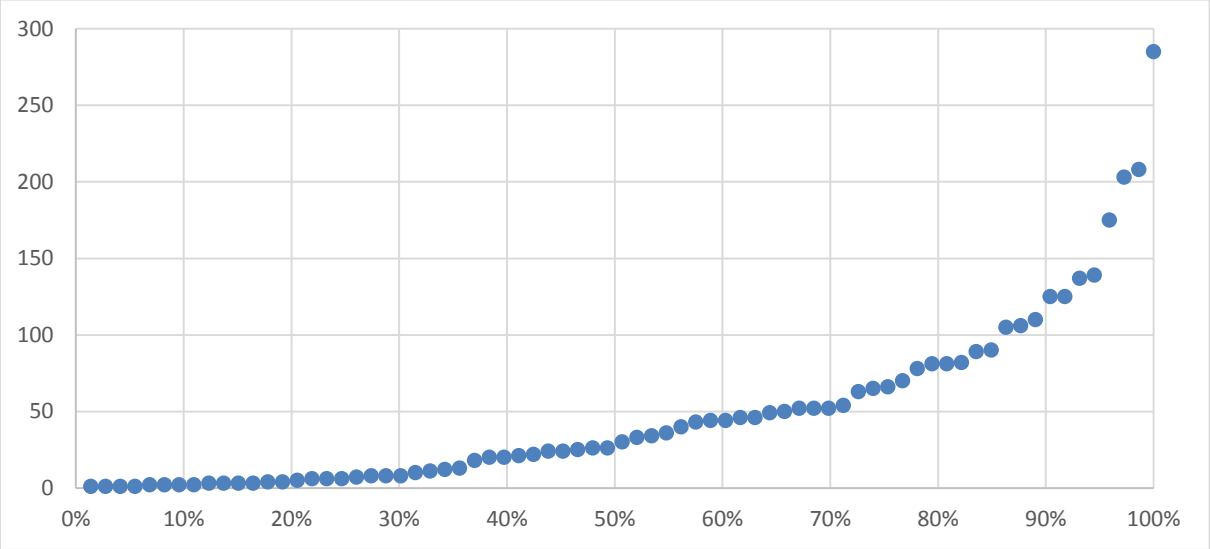


Figura 17 - Gráfico de dispersão usado na definição dos valores de referência de métrica de Correção de Erros

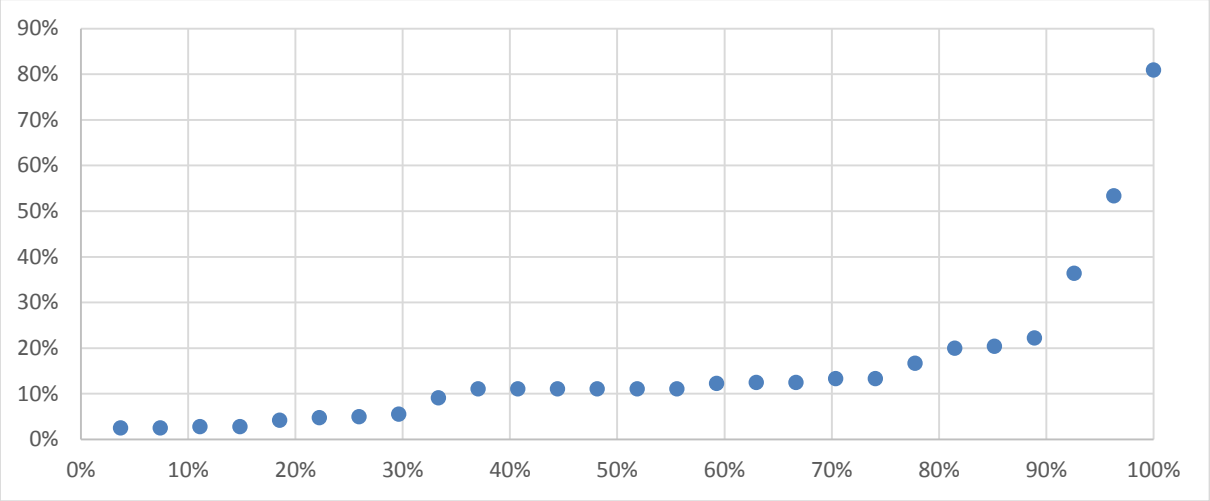


Figura 18 - Gráfico de dispersão usado na definição dos valores de referência da métrica de Complexidade Média em Métodos Adicionados

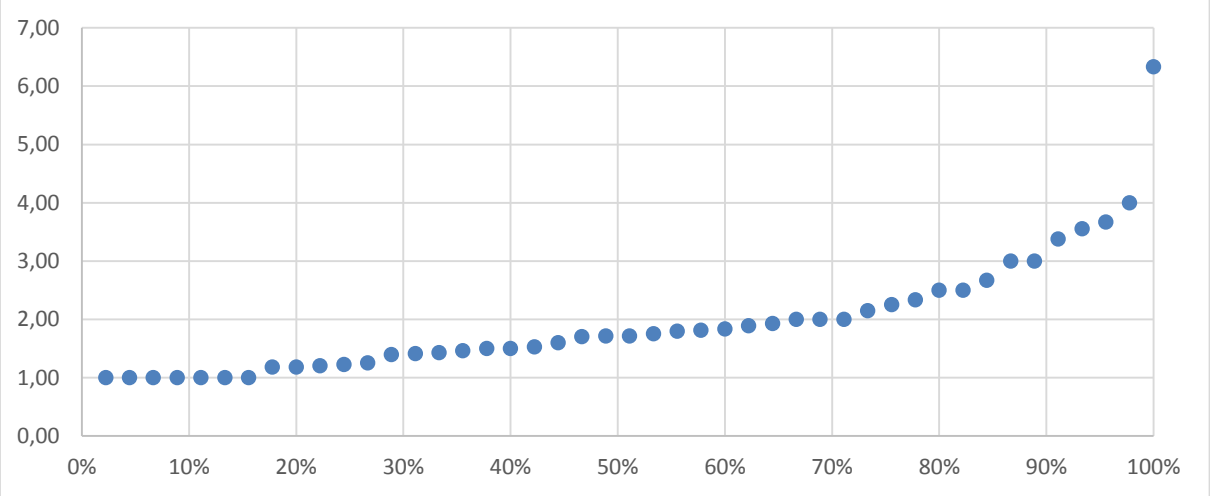


Figura 19 - Gráfico de dispersão usado na definição dos valores de referência da métrica de Complexidade Média em Métodos Modificados

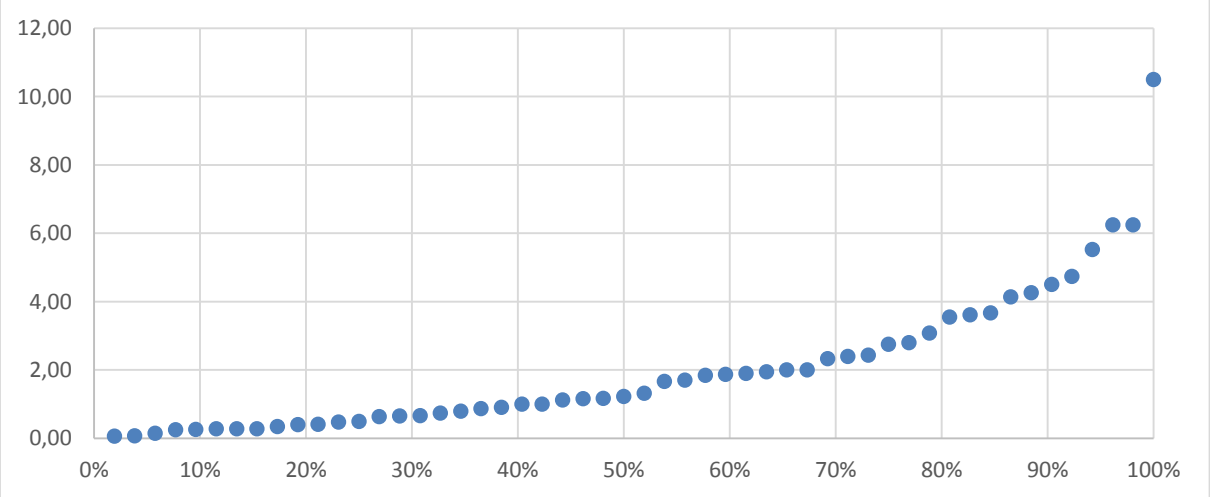
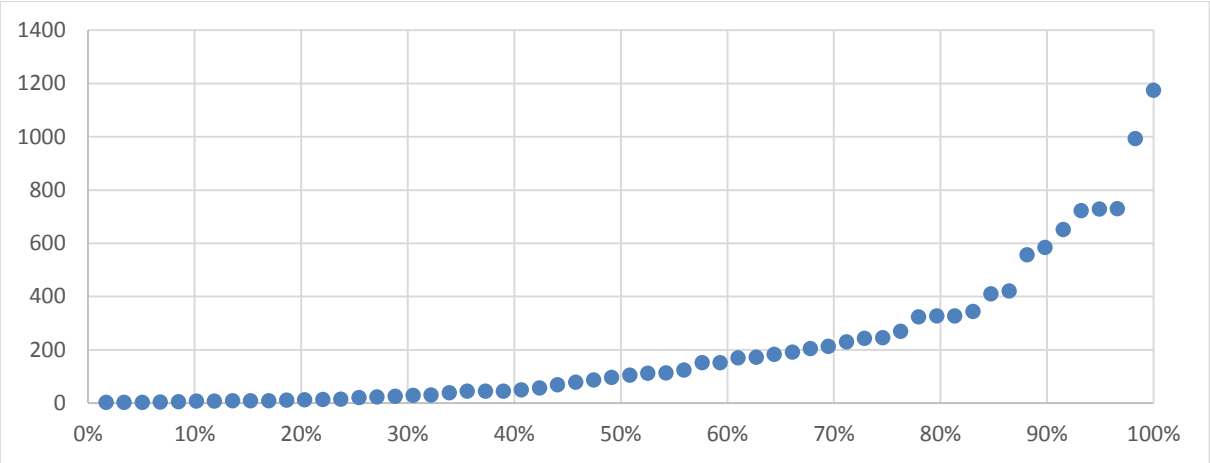


Figura 20 - Gráfico de dispersão usado na definição dos valores de referência da métrica de Volume de Contribuição



SIPAC

Figura 21 – Gráfico de dispersão usado na definição dos valores de referência de métrica de Correção de Erros

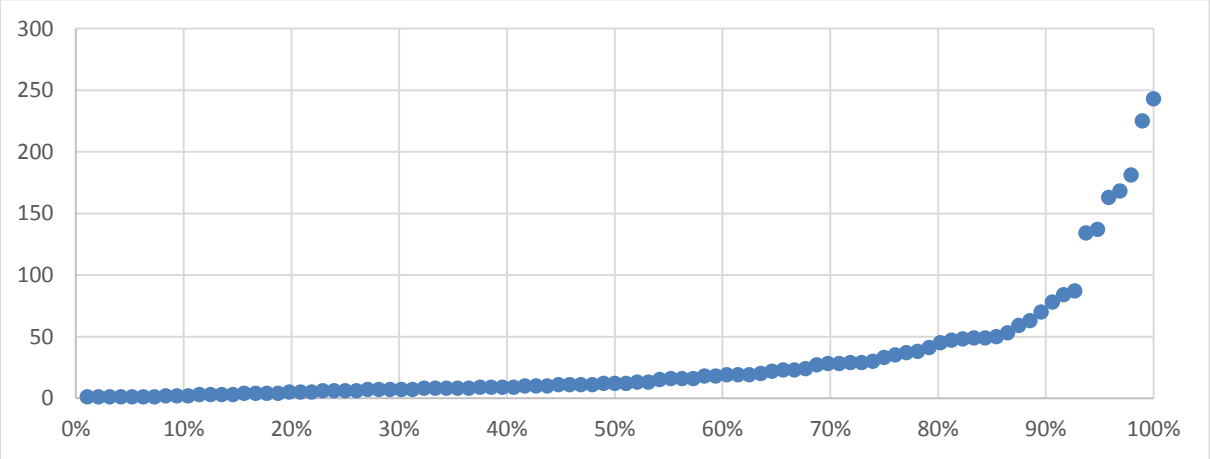


Figura 22 – Gráfico de dispersão usado na definição dos valores de referência da métrica de Correção de Erros

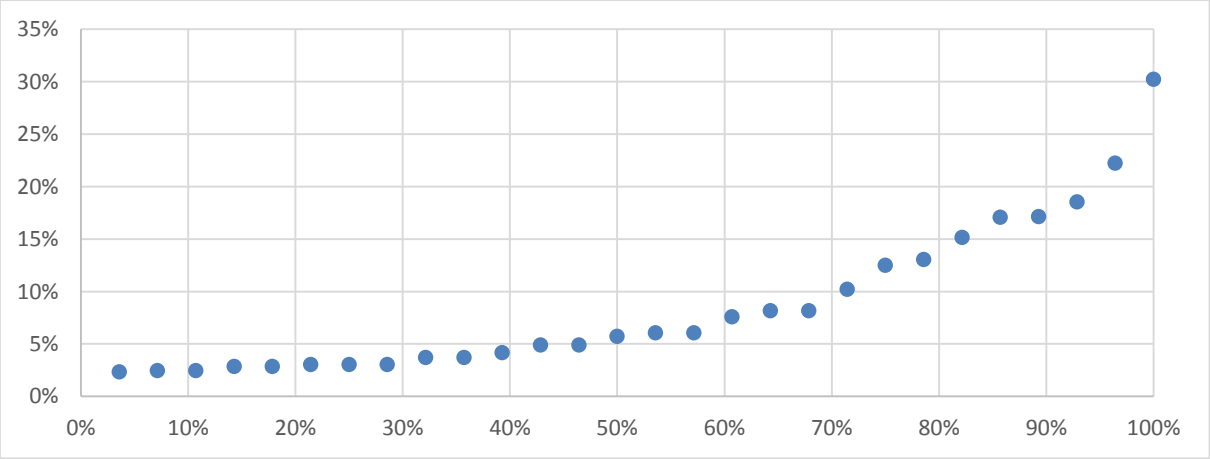


Figura 23 – Gráfico de dispersão usado na definição dos valores de referência da métrica de Complexidade Média em Métodos Adicionados

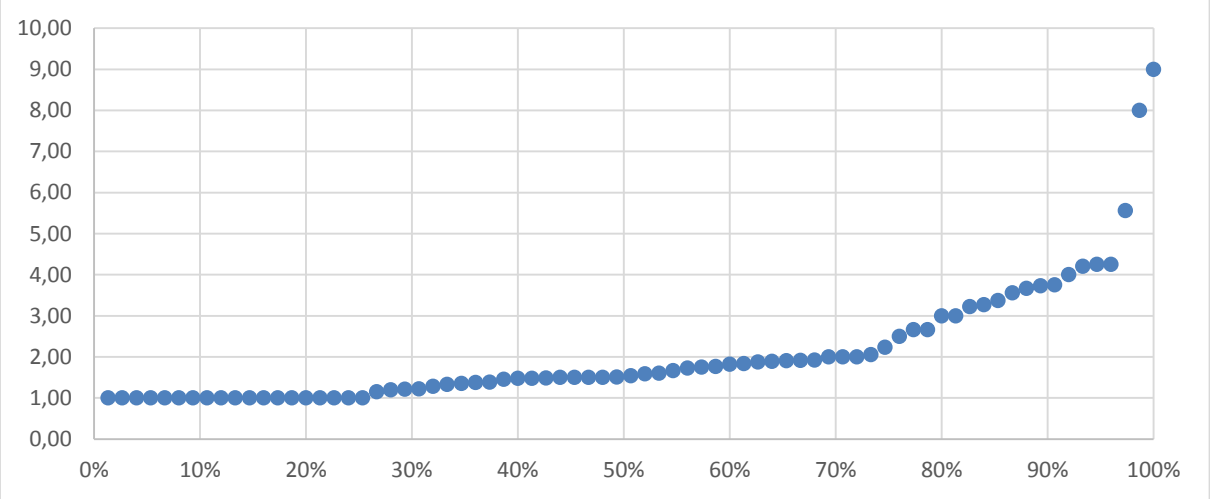


Figura 24 – Gráfico de dispersão usado na definição dos valores de referência da métrica de Complexidade Média em Métodos Modificados

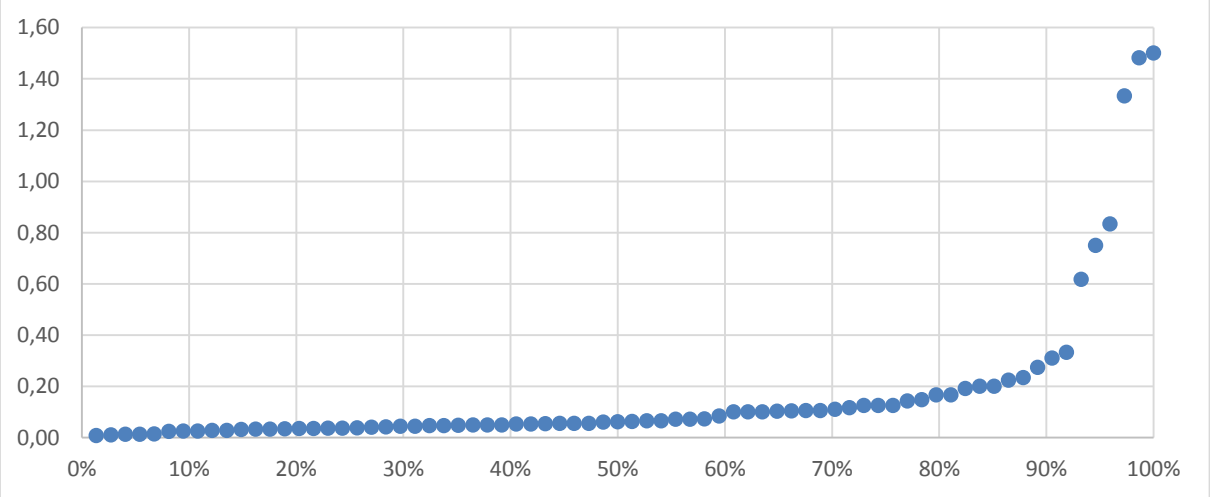
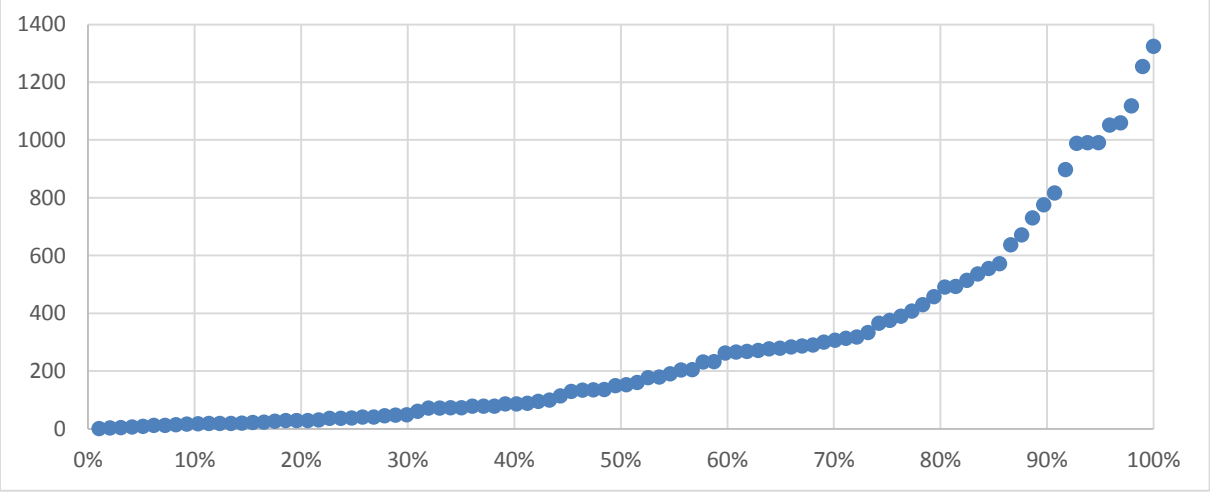


Figura 25 – Gráfico de dispersão usado na definição dos valores de referência da métrica de Volume de Contribuição



Sucupira

Figura 26 – Gráfico de dispersão usado na definição dos valores de referência da métrica de Introdução de *Bugs*

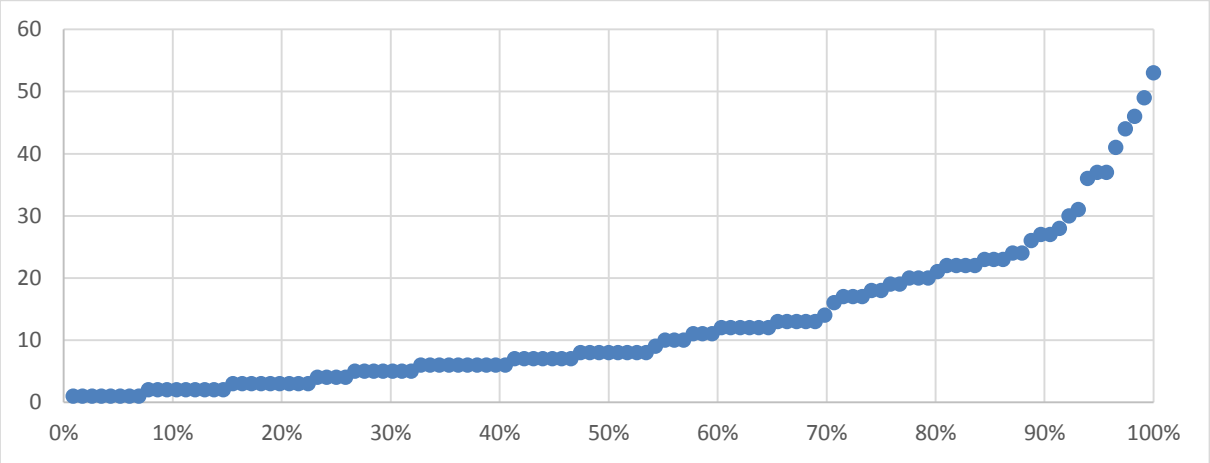


Figura 27 – Gráfico de dispersão usado na definição dos valores de referência da métrica de Correção de Erros

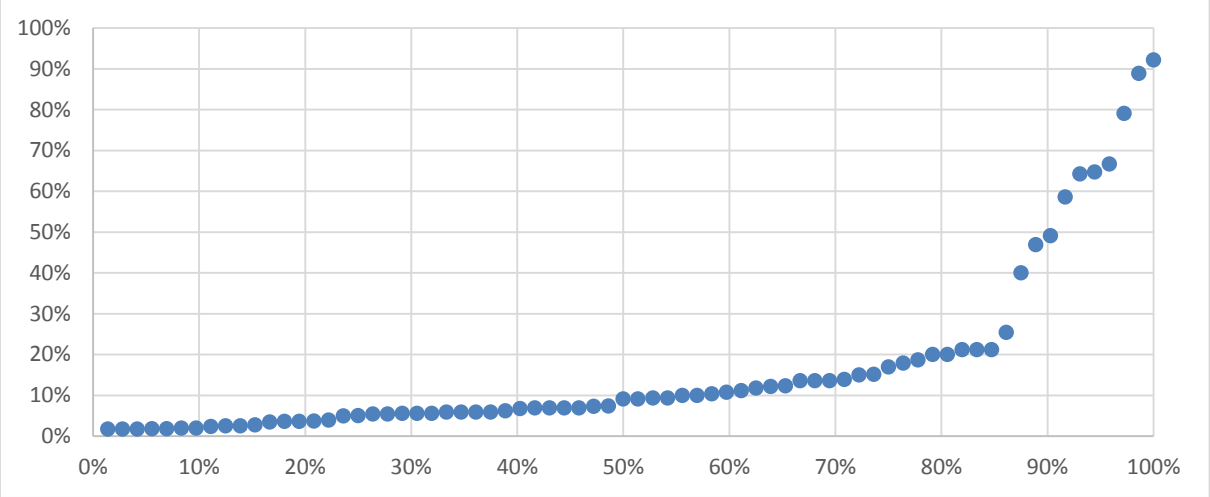


Figura 28 – Gráfico de dispersão usado na definição dos valores de referência da métrica de Complexidade Média em Métodos Adicionados

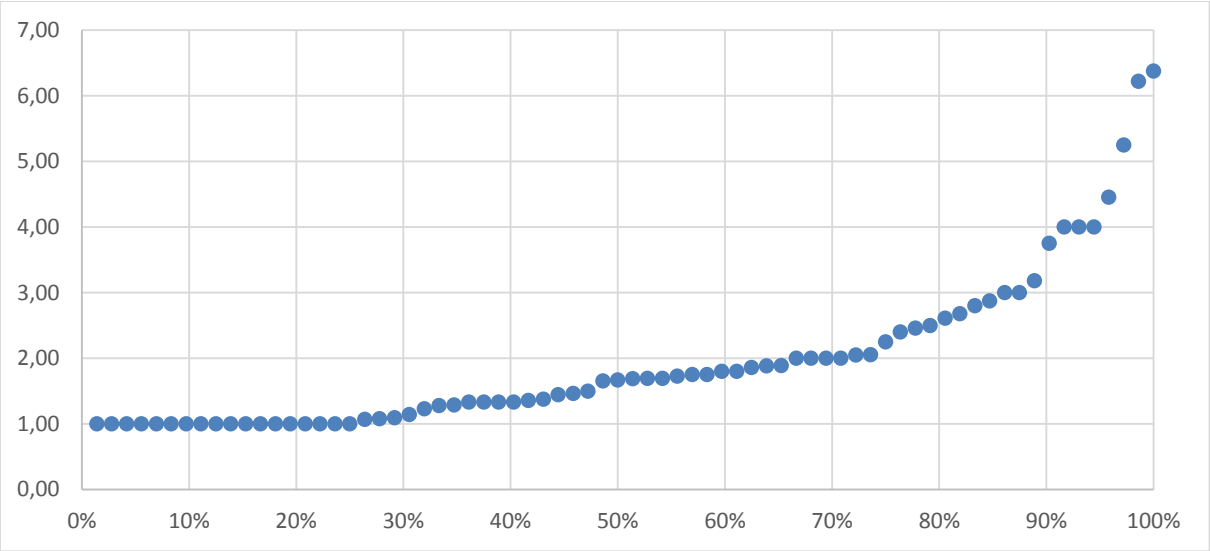


Figura 29 – Gráfico de dispersão usado na definição dos valores de referência da métrica de Complexidade Média em Métodos Modificados

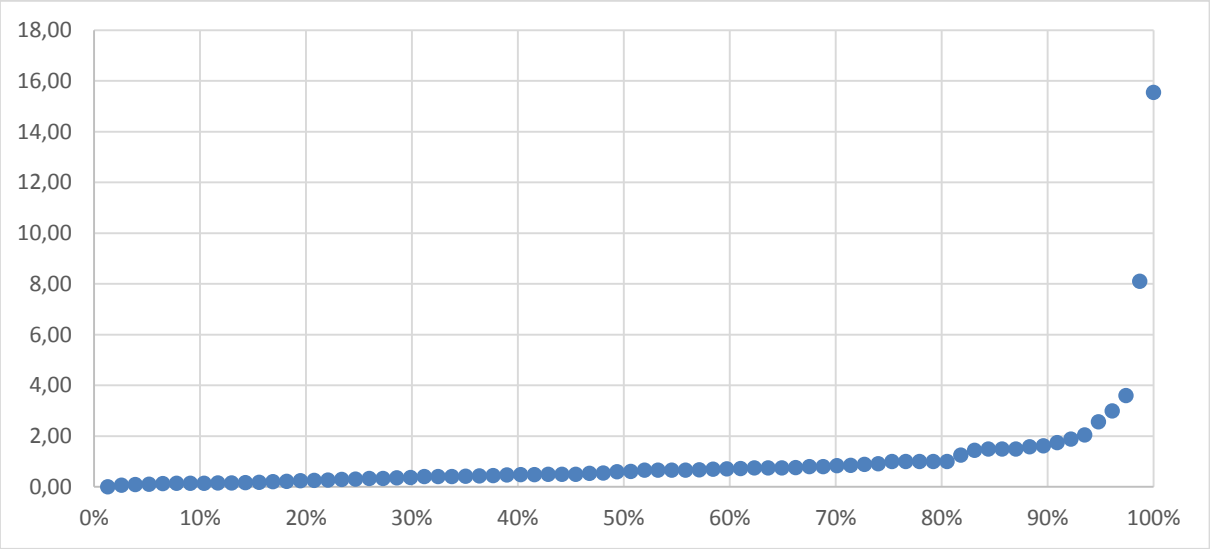
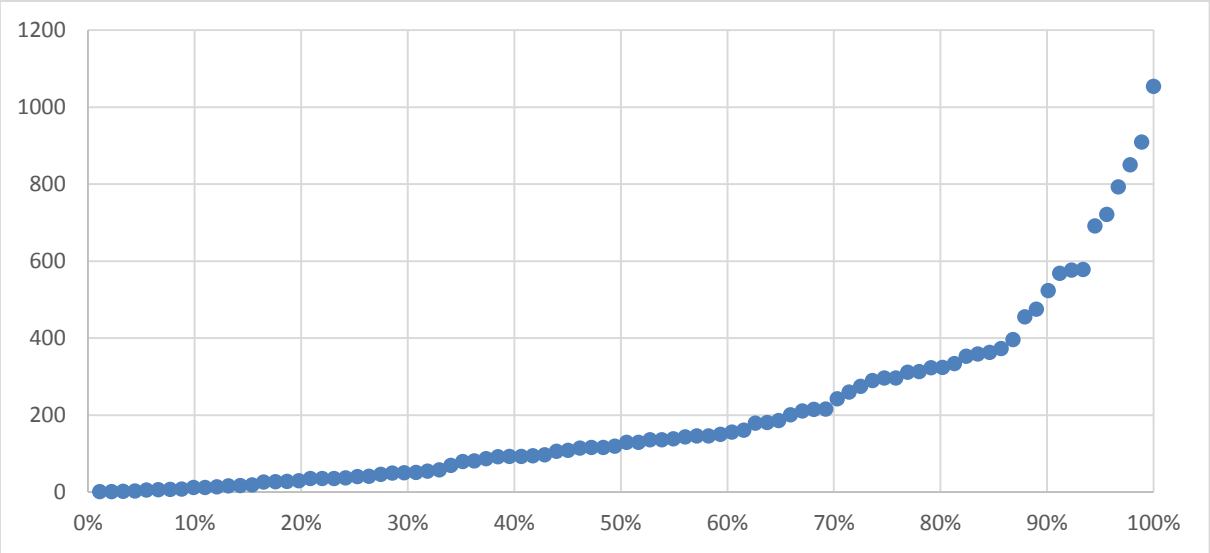


Figura 30 – Gráfico de dispersão usado na definição dos valores de referência da métrica de Volume de Contribuição


1200
1000
800
600
400
200
0
0%
10%
20%
30%
40%
50%
60%
70%
80%
90%
100%

APÊNDICE C: GUIA DE APLICAÇÃO DA ABORDAGEM

Este apêndice apresenta um guia de aplicação da abordagem proposta nesta dissertação, observando os passos a serem executados para a formatação de abordagem e sua posterior execução.

1) **Definição dos atributos de avaliação:** o primeiro passo para a aplicação da abordagem é a definição dos atributos de avaliação, ou seja, determinar sob quais aspectos de contribuição os desenvolvedores serão avaliados;
2) **Definição das métricas que quantificam os atributos:** uma vez definidos os atributos, é preciso escolher quais métricas serão usadas para quantificar tais atributos;
3) **Implementação/adaptação das minerações:** caso não existam minerações prontas para extrair os dados e produzir as métricas escolhidas, será necessário implementar tais minerações; caso já existam minerações que possam produzir as métricas escolhidas, elas ainda assim, provavelmente, precisarão de ajustes para se adequar à dinâmica de trabalho (*workflow*) das equipes de desenvolvimento a serem analisadas;
4) **Escolha das técnicas de identificação de *outliers* e definição de valores de referência:** conforme apresentado no Capítulo 3, a abordagem proposta é composta por uma técnica de identificação e descarte de *outliers*, e uma técnica de definição de valores de referência. Nesse passo é importante definir se essas técnicas serão usadas na abordagem, uma vez que, apesar de sua importância, são opcionais, e se forem utilizadas, quais serão essas técnicas, haja vista que existem outras técnicas além daquelas apresentadas nesta dissertação;
5) **Execução da abordagem:** uma vez executados os passos anteriores, já é possível executar a abordagem através dos passos apresentados na Seção 3.1.