UNIVERSIDADE DO SUL DE SANTA CATARINA

FERNANDO ANTUNES RODRIGUES

USO DE INDICADORES PARA MEDIR A QUALIDADE DO CÓDIGO ATRAVÉS DAS FERRAMENTAS SONAR E UNDERSTAND/STRUCTURE

Florianópolis

2017

FERNANDO ANTUNES RODRIGUES

USO DE INDICADORES PARA MEDIR A QUALIDADE DO CÓDIGO ATRAVÉS DAS FERRAMENTAS SONAR E UNDERSTAND/STRUCTURE

Trabalho de Conclusão de Curso apresentado ao Curso de Engenharia de Projetos de Software da Universidade do Sul de Santa Catarina como requisito parcial à obtenção do título de Especialista em Engenharia de Projetos de Software.

Orientador: Prof. Flavio Ceci, Dr.

Florianópolis

2017

FERNANDO ANTUNES RODRIGUES

USO DE INDICADORES PARA MEDIR A QUALIDADE DO CÓDIGO ATRAVÉS DAS FERRAMENTAS SONAR E UNDERSTAND/STRUCTURE

Este Trabalho de Conclusão de Curso foi julgado adequado à obtenção do título de Especialista em Engenharia de Projetos de Software e aprovado em sua forma final pelo Curso de Engenharia de Projetos de Software da Universidade do Sul de Santa Catarina.

Florianópolis, 31 de maio de 2017.

_____

Professor e orientador Flávio Ceci, Dr.
Universidade do Sul de Santa Catarina

_____

Prof. e avaliador Edson Orivaldo Lessa Junior, Esp.
Universidade do Sul de Santa Catarina

Dedico a realização desse trabalho a Deus que me proporcionou chegar até aqui.

AGRADECIMENTOS

Agradeço a minha esposa por estar ao meu lado nesse momento e que principalmente teve a sabedoria e paciência de aguardar a finalização desse trabalho.

Aos meus pais que mesmo distante tenho certeza absoluta que tem orado por mim para que tudo venha dar certo.

E aos demais colegas de classe e amigos que me acompanharam durante esse período.

Também agradeço ao meu orientador professor Flávio, pelo apoio incondicional e indispensável na orientação em relação a esse trabalho, pela prestatividade e colaboração em cada etapa da pesquisa.

“Aquilo que não se pode medir, não se pode melhorar” (William Thompson, 1900).

RESUMO

O progresso a cada dia do desenvolvimento de software traz a tona a necessidade de se preocupar cada vez mais com a qualidade de software, cliente se torna exigente a cada dia e isso mostra que o tempo a ser gasto com essas ferramentas deve ser muito explorado. Hoje preocupa-se muito mais em entregar o produto para o cliente e realizar o faturamento o quanto antes e deixa-se em segundo plano a qualidade do mesmo pelo fato muitas vezes do custo, do desconhecimento de práticas que proporcionam um código limpo e principalmente pelo fato de não saber quais pontos melhorar dentro do código, ou seja, como evoluir a qualidade sem ao menos saber onde melhorar. Esta pesquisa vem retratar alguns dos conceitos do mercado de desenvolvimento de software que atualmente tem se tornado primordial no acompanhamento dos projetos, as métricas, onde com esse estudo pretende-se mostrar ferramentas que coletam métricas de código fonte de um sistema desenvolvido em linguagem de programação Delphi, afim de transparecer os principais pontos do sistema que deve-se atacar e melhorar. Na seção final desse trabalho é apresentado um questionário que foi aplicado a uma empresa de desenvolvimento de software, que dá ênfase a avaliar as ferramentas Sonar e Understand com base na eficiência da geração de seus dados e a forma com que os mesmos são apresentados para os integrantes das equipes de desenvolvimento. Com isso pretende-se obter o conhecimento de o quanto as ferramentas Sonar e Understand podem ser grandes aliados da equipe de arquitetura, desenvolvedores e gestores no acompanhamento da saúde do código fonte e o quanto conceitos de código limpo tem sido aplicados dentro do mesmo. Ao final a pesquisa é encerrada apresentando os resultados do questionário onde pode se analisar alguns pontos relevantes em relação a eficiência do Sonar e Understand na amostragem dos resultados, e também o conhecimento dos entrevistados em relação a código limpo e algumas das métricas de código fonte que foram o cerne da pesquisa.

Palavras-chave: Qualidade. Métricas. Software.

ABSTRACT

The progress every day of software development brings out the need to worry more and more about the quality of software, customer becomes demanding every day and this shows that the time to be spent with these tools should be greatly explored. Today, we are much more concerned with delivering the product to the customer and making the billing as soon as possible, and the quality of the invoice is left in the background due to the fact that it is often inexpensive, ignoring practices that provide a clean code and Fact of not knowing which points to improve within the code, that is, how to evolve quality without even knowing where to improve. This research presents some of the concepts of the software development market that has now become paramount in the monitoring of the projects, the metrics, where with this study we intend to show tools that collect source code metrics from a system developed in programming language Delphi, in order to show the main points of the system that must be attacked and improved. In the final section of this paper, a questionnaire was presented that was applied to a software development company, which emphasizes the evaluation of the Sonar and Understand tools based on the efficiency of the generation of their data and the way in which they are presented to the Members of the development teams. With this, we intend to obtain the knowledge of how much the Sonar and Understand tools can be great allies of the architecture team, developers and managers in the health monitoring of the source code and how much concepts of clean code have been applied within it. At the end, the research is concluded presenting the results of the questionnaire where we can analyze some relevant points regarding the efficiency of Sonar and Understand in the sampling of results, as well as the knowledge of the interviewees regarding clean code and some of the source code metrics Were at the heart of the research.

Keywords: Quality. Metrics. Software.

LISTA DE ILUSTRAÇÕES

LISTA DE GRÁFICOS

LISTA DE QUADROS

SUMÁRIO

1 INTRODUÇÃO

Quanto trata-se de métricas de software é inegável que vem à mente uma ideia de números, gráficos entre outras informações que aparentemente podemos não entender inicialmente, a bem da verdade é que quando menciona-se esse assunto, não há como não lembrar da qualidade de software e que por sua vez também está atrelada ao processo de desenvolvimento de software utilizado.

Neste cenário faz-se necessário voltar um pouco no tempo e resgatar a expressão "crise do software" vinculada as dificuldades encontradas nos anos 70 quando ainda era muito pouco mencionado o termo engenharia de projetos de software, e que fazia falta frente a demanda crescente de desenvolvimento, que encontrava vários problemas com a complexidade e o tamanho dos novos programas e praticamente não se adotava nenhuma técnica ou padrão de desenvolvimento, e também como o tema qualidade não era observado com grande intensidade e muito menos pensava-se em medir o que era feito.

Essa mudança tecnológica se transformou em forma dramática e com um breve período de tempo, os recursos de hardware aumentaram muito e permitiram que produtos mais complexos fossem criados. Sendo que a primeira vez que se utilizou o termo "Engenharia de Software" foi em uma conferência com esse nome realizada em 1968 na Alemanha com o apoio de uma entidade que não possuía nenhuma ligação com a área: o comitê de ciência da NATO(North Atlantic Treaty Organization – Organização do Tratado do Atlântico Norte) Koscianski e Soares (2007, p.21).

Mesmo naquela época muitos dos problemas eram parecidos com os problemas encontrados pelas empresas de hoje, porém a consciência das mesmas tem mudado gradualmente e percebe-se que as mesmas têm observado a qualidade de software como algo extremamente essencial para seus clientes porque sem isso irão perder mercado.

Hoje vê-se grandes empresas que utilizam ferramentas para adequar o nível de qualidade de software exigido pelos clientes de uma forma mais abrangente, ao passo que em pequenas empresas ainda essa consciência é imatura.

A ponta de tudo isso que foi comentado é onde se quer chegar e que se define como: qualidade de software, porém isso tudo passa por um processo de desenvolvimento que faz com que tenhamos o nível de qualidade satisfatório alcançado, porém se o processo é deficiente é claro que o resultado final será o mesmo, ou seja baixa qualidade. Para ficar mais claro tecnicamente Pressman (2011, p.52) define processo como:

> Processo de software é definido como uma metodologia para as atividades, ações e tarefas necessárias para desenvolver um software de alta qualidade. “Processo” é sinônimo de engenharia de software? A resposta é “sim e não”. Um processo de software define a abordagem adotada conforme um software é elaborado pela engenharia. Mas a engenharia de software também engloba tecnologias que fazem parte do processo – métodos técnicos e ferramentas automatizadas.

É lógico que o processo de desenvolvimento não vai garantir a qualidade do produto no final e ainda mesmo assim é necessário o comprometimento dos desenvolvedores em utilizar a disciplina e metodologias corretas para se ter um bom código, afinal um código mal escrito não poderá ser escalável e não sendo escalável quer dizer que no final o mesmo não irá produzir um produto com nível de qualidade satisfatório.

Assim como Koscianski e Soares (2007, p.18) relatam que a qualidade de software é dependente principalmente do correto emprego de boas metodologias pelos desenvolvedores e ainda que sejam apoiados por várias ferramentas, ainda restam problemas sérios sem tal suporte.

No que tange a insistência no tema de qualidade de software, temos o seu principal indicador para atingir esse nível que é conseguido através de métricas de software. Métricas nada mais são que uma forma de poder medir e qualificar o código fonte desenvolvido bem como o processo de desenvolvimento utilizado. Para deixar mais claro Pressman (2011, p.538) define medição como:

> Medição é o processo pelo qual números ou símbolos são anexados aos atributos de entidades no mundo real para defini-los de acordo com regras claramente estabelecidas... Nas ciências físicas, medicina, economia e mais recentemente nas ciências sociais, podemos medir os atributos antes considerados incomensuráveis... É claro que essas medidas não são tão refinadas como muitas feitas nas ciências físicas, mas existem (e são tomadas decisões importantes baseadas nelas). Percebemos que a obrigação de tentar “medir o incomensurável” para melhorar nossa compreensão de entidades particulares é tão poderosa na engenharia de software quanto em qualquer outra disciplina.

Dessa forma a pesquisa apresenta indicadores que são aplicados em um sistema escrito em linguagem Delphi em uma empresa da cidade de Florianópolis com intuito de mostrar em alto nível, como realmente está o código fonte do produto afim de extrair indicadores que irão fornecer informações fundamentais para conhecer e melhorar a tomada de decisão afim de produzir um software com qualidade.

O software a ser analisado está escrito em linguagem Delphi e dentro da pesquisa será dada ênfase aos indicadores extraídos das ferramentas Sonar e Understand/Structure, onde métricas já são geradas, e atualmente os dados coletados tem se transformado em indicadores,

para uma melhor tomada de decisão em relação a equipe, estimativas de trabalho, reduzir frustrações e pressões de cronogramas e qualidade e produtividade do processo de desenvolvimento.

Para fins de comparação de métricas foi escolhido um software da empresa de terceiro assim referenciada nesse trabalho e todas as métricas mencionadas no trabalho que foram aplicada ao primeiro software será aplicado também ao software de terceiro.

1.1 PROBLEMÁTICA

A busca por qualidade tem sido algo falado frequentemente em todas as áreas nos dias de hoje, porém em se tratando de desenvolvimento de software isso tem sido debatido de forma mais ampla nos últimos anos por considerar que a qualidade deva ser aplicada também e principalmente em software. Existe um grande esforço em cima disso, para que se possa melhorar os produtos de forma contínua, porém como se pode saber onde melhorar se não se tem noção de onde encontram-se os principais problemas?.

Existem várias razões pela qual um software é medido, onde se pode citar: indicar a qualidade do produto, avaliar a produtividade das pessoas que desenvolvem o software, formar uma linha básica para estimativas, justificar os pedidos de novas ferramentas ou treinamentos, avaliar os benefícios em relação a qualidade e produtividade para derivação de novos métodos e ferramentas de software. Como exemplo de medidas diretas do processo é possível citar o custo e esforço aplicado, já para medidas de produto pode-se citar como exemplo as linhas de código produzidas, velocidade de execução, tamanho de memória, defeitos registrados, etc (REZENDE,p.101, 2005).

Segundo Subramanyan (2017) avaliar a qualidade de um software é um mistério, muitos profissionais de TI ficam frustrados sobre como definir e medir a qualidade das aplicações e essas dificuldades resultam de um foco incorreto sobre o processo pelo qual é construído e dessa forma achar uma definição de como medi-lo com precisão para que as pessoas possam ver e focar nas atividades necessárias para criar, melhorar e gerenciar o software.

Dessa forma ter uma visibilidade deficiente em relação ao código fonte acarreta em uma série de problemas de gestão do software como a incapacidade de criar mapas de arquitetura do aplicativo, dificuldade em conhecer o próprio código fonte em meio a uma

centena de padrões de implementação, dificuldade em estimar as tarefas e dificuldade em reconhecer determinados pontos para providenciar melhorias do sistema.

Quando se enxerga um cenário onde mostra a dificuldade de implantar uma forma clara para medir o código fonte, o importante em um primeiro momento é identificar os gargalos críticos e verificar realmente o que é necessário medir inicialmente e começar pelas métricas menos complexas conforme Oakland (p.180, 1994).

> Já foi estabelecido que um bom sistema de medição deve começar com o cliente e medir as coisas certas. O valor de qualquer medida precisa, obviamente, ser comparado com o custo de produzi-la. Deverão existir medidas adequadas para os diferentes setores da organização, mas em cada um deles o desempenho deve estar relacionado com as necessidades do cliente do processo. Todas as partes críticas do processo devem ser medidas, e é muitas vezes melhor começar com medidas simples para em seguida aperfeiçoá-las.

A utilização de métricas em outras áreas da engenharia já existe e é amplamente utilizado, porém em se tratando de engenharia de software ainda é deixado de lado pelo fato de ser um ponto que gera muita discussão de como montar indicadores que possam corroborar para um resultado confiável e que possa trazer um resultado não somente subjetivo, mas que venha trazer uma visão do que precisamos realmente observar no código. A dificuldade em concordar sobre o que medir e como aproveitar essas informações obtidas ainda tem sido um grande empecilho dentro das empresas.

Ao passo que nesse momento se encontra esse cenário atual no mercado de software é relevante responder a questão da pesquisa: Quais os benefícios que podem ser vislumbrados e as ações que podem ser tomadas a partir de uma coleta bem estruturada de métricas de código fonte?

1.2 OBJETIVOS

O objetivo desta seção é apresentar o propósito deste trabalho acadêmico. Sua proposta é composta por Objetivo Geral e seus Objetivos Específicos.

1.2.1 Objetivo Geral

Objetiva-se com este trabalho analisar as métricas utilizadas em códigos-fonte como forma de mensuração da qualidade interna do produto, utilizando as ferramentas sonar e understand/structure.

1.2.2 Objetivo Específico

Os objetivos específicos que se seguem visam findar o objetivo geral deste trabalho:

- Compreender as principais métricas para qualidade de código e discutir a sua importância durante o ciclo de vida de um software.
- Mostrar os resultados extraídos das medições através das ferramentas sonar e understand/structure comparando-as com software de terceiro.
- Submeter um questionário para a identificação da importância dos indicadores apresentados pela ferramenta.

1.3 JUSTIFICATIVA

Uma das justificativas pelas quais o conceito de métricas pode ser importante é o fato de a mesma demonstrar fatores internos de um software bem feito como é o caso da manutenibilidade por exemplo, algo exclusivo dele e não é percebida através de uma visão externa revelada por exemplo através de uma bateria de testes. Nesse contexto as métricas de código fonte não podem ser negligenciadas em relação a outras abordagens da matéria de qualidade de software FILHO (2013, p.01).

Conforme é definido na NBR ISO/IEC 9126-1 (2003) o conceito de manutenibilidade se refere a capacidade do produto do software ser modificado e através dessas alterações podem incluir correções, melhorias ou adaptações do software devido a mudanças no ambiente e nos seus requisitos ou especificações funcionais.

Quando o desenvolvimento acontece é possível ao longo do tempo ir mapeando os pontos a serem melhorados a cada dia, é claro que o que foi mencionado anteriormente é uma das características que com o auxílio de indicadores podem revelar onde estão os principais problemas encontrados dentro do código e assim podendo expor as fragilidades que torna um código de manutenção complicado e suscetível a uma grande quantidade de defeitos, tornando o mesmo complexo para evoluir e inserir correções, segundo Pressman (2011, p.583).

> Quando você pode medir o que você está falando e expressar em números, você sabe alguma coisa sobre aquilo; mas quando você não pode medir, quando você não pode expressar em números, o seu conhecimento é escasso e insatisfatório: pode ser o começo do conhecimento, mas você avançou muito pouco, em seu raciocínio, para o estágio de uma ciência.

Dessa forma, a implantação de uma ferramenta que tem facilitado a extração de indicadores traz uma abordagem relacionada a visão do todo, não estará eliminando totalmente problemas cruciais nas liberações de versões e outras etapas do desenvolvimento de software, porém tornando a visão do código fonte em alto nível será possível direcionar medidas para correção e melhorias e também mensurar um tempo mais próximo da realidade na projeção de estimativas para desenvolvimento de uma funcionalidade entre outras medidas que já foram citadas anteriormente.

O trabalho visa descrever as principais métricas do mercado, avaliar as ferramentas de extração de métricas com base nos resultados que são apresentados para o software proposto, compará-los com software de terceiro e entender o quão eficiente as ferramentas são, para isso as opiniões de quem trabalha dentro do ciclo de desenvolvimento de software serão ouvidas e avaliadas também.

1.4 ESTRUTURA DA MONOGRAFIA

Quanto da estrutura do trabalho acadêmico, se dá por:

Capítulo 1 – Introdução, problemática, os objetivos e a justificativa do trabalho acadêmico.

Capítulo 2 – Revisão bibliográfica, sustentada pela pesquisa e conhecimento das métricas de qualidade código, desenvolvimento de software, conceitos de qualidade e teste de software. Nesta seção serão explorados os seguintes temas conceituais:

- Qualidade de Software
- Garantia da Qualidade de Software;
- Métricas de Software;
- Métricas de código fonte;

Capítulo 3 – Metodologia de Pesquisa.

Capítulo 4 – Experimento

Capítulo 5 – Resultados e discussões

Capítulo 6 – Considerações finais e trabalhos futuros.

2 REVISÃO BIBLIOGRÁFICA

Neste capítulo será detalhada toda a fundamentação teórica necessária para o desenvolvimento deste trabalho. Nela são apresentadas conceitos de qualidade, garantia da qualidade, TQM (Total Quality Management), métricas de software e algumas das principais métricas de código fonte do mercado.

2.1 QUALIDADE DE SOFTWARE

A qualidade de software pode ser vista de formas diferentes pelo cliente e a mesma implica em atender requisitos básicos daquele que o está adquirindo afim de satisfazê-lo, porém o que faz a diferença não é o que está por trás do mesmo ou o que foi utilizado para seu desenvolvimento e sim o quão aquilo irá atendê-lo, porém para quem trabalha com desenvolvimento de software saber medir e entender toda a parte estrutural de um código fonte é extremamente importante assim como menciona Koscianski e Soares (2007, p.27):

> A qualidade de um produto tem um propósito: satisfazer o cliente. Esse objetivo implica tratar um domínio, em geral, bastante nebuloso. Para compreender o motivo, considere novamente o caso de uma pessoa que deve adquirir um produto comum no mercado. Ninguém compra uma camisa pensando nas propriedades mecânicas do tecido com o qual ela foi fabricada: “Que bela camisa! Pode resistir a 10 kg de tração!”. Em vez disso, são fatores muito difíceis de medir que, em geral, terão maior peso na decisão.

Além disso esse conceito tem sido levado de forma competitiva para dentro das empresas que entenderam que a qualidade não é mais um simples conceito visto somente na literatura, e sim passou para uma consolidação dentro do mercado para a maioria das empresas que tem visto o ganho que isso tem em várias áreas dos projetos de desenvolvimento de software, e assim define Morais (2007, p.34):

> O conceito da qualidade tem hoje importância fundamental para alavancar a competitividade das empresas. Atualmente, a preocupa- ção com a qualidade deixou de ser um diferencial competitivo e passou a ser um pré-requisito básico para participação no mercado. No setor de software não é diferente. A disseminação do uso do software em todas as áreas, envolvendo monitoração, controle e gestão de funções críticas, tem aumentado consideravelmente a importância da qualidade de software

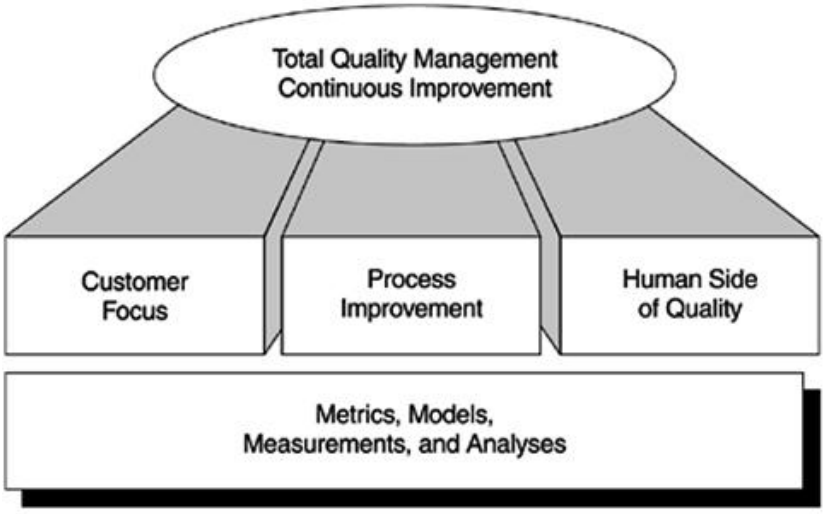
Dessa forma chegamos ao ponto onde a qualidade deriva para o conceito de garantia da qualidade de software citado por Koscianski e Soares (2007, p.27):

> O propósito da subárea de garantia da qualidade é assegurar que os objetivos planejados no início do projeto serão cumpridos. De forma geral, isso significa estabelecer sistemas para controlar tudo o que ocorre durante o ciclo de vida, com o propósito de garantir que o programa que será fabricado fará aquilo que se espera dele.

Construir um produto de qualidade implica em executar um conjunto de tarefas que ao final irá resultar a satisfação desejada ao cliente, o resultado deve refletir a aplicação total dos métodos para garantir isso e só poderá ser alcançado através da execução total e não parcial de todos os processos envolvidos, dessa forma entramos em outro conceito que vai nos ajudar a entender ainda mais os conceitos até aqui tratados.

O termo TQM ilustra de forma mais clara tais conceitos de qualidade e garantia de qualidade na Figura 1.

Figura 1 – Conceitos Principais do TQM



Fonte: Kan (2003, p. 09).

O conceito de TQM tem relação direta com a melhoria contínua e a satisfação do cliente, o princípio dessa abordagem se refere principalmente ao foco com acentuação em vários setores dentro da empresa desde o nível da produção passando pelos processos da mesma, chegando até o nível gerencial, conforme menciona Oakland (p.09, 1994)

> A redução contínua dos custos, a produtividade e a melhoria da qualidade têm demonstrado que são essenciais para as organizações se manterem em operação. Não podemos deixar de ver como a qualidade transformou-se na mais importante arma competitiva e muitas organizações estão convencidas de que o TQM é o modo de gerência do futuro. O TQM, em suas aplicações, vai muito além do que apenas garantir a qualidade do produto ou do serviço – é uma maneira de gerenciar os processos da empresa para assegurar a completa satisfação do "cliente", em cada etapa, tanto interna como externamente.

O fato de mencionarmos esses conceitos acima são importantes para embasar o quão é importante a consciência das empresas de software em aderir a abordagens vislumbrando a qualidade, isso não é simplesmente um luxo mas sim necessário no mercado atualmente concorrido já que o software é considerado uma prestação de serviço, conforme menciona Lobo (p.07, 1997).

> Esse conceito abrangente de qualidade, ligado à gestão administrativa, aplica-se tanto para a produção de mercadorias e bens de consumo como para serviços. O software é considerado, para finalidades legais ou considerações de qualidade, como uma prestação de serviços. Isso se aplica tanto para o produto mais concreto - um programa e seu conjunto de manuais (o chamado 'software de prateleira') - como para o treinamento, instalação, manutenção, desenvolvimento sob medida, etc...)

Dessa forma, a qualidade total é extremamente importante estabelecer e superar as expectativas do cliente. Com isso para atingir o nível de qualidade desejável a empresa deve atender o que o cliente realmente precisa para que o que está sendo construído, seja ele um bem ou serviço possa ser concebido com excelência. Resumindo conhecer, compreender e praticar esses conceitos de qualidade total é importante pois é uma forma de ação que coloca a qualidade dos serviços e produtos como foco principal e como consequência a satisfação do cliente.

2.2 GARANTIA DA QUALIDADE DE SOFTWARE

Quando se aborda o conceito de qualidade de software não se pode deixar de mencionar também um dos principais elementos que garantem a qualidade de software que é o teste de software e também por muitos autores definido como uma etapa de verificação e validação. Há muitos anos a atividade propriamente era resumida em uma simples checagem do próprio desenvolvedor, porém Pressman (p. 468, 2002) define sua importância e retrata de forma mais clara essa etapa.

> A verificação e validação incluem uma ampla gama de atividades de SQA (software quality assurance – garantia da qualidade de software): revisões técnicas, auditorias de qualidade e configuração, monitoramento de desempenho, simulação, estudo de viabilidade, revisão de documentação, revisão de base de dados, análise de algoritmo, teste de desenvolvimento, teste de usabilidade, teste de qualificação, teste de aceitação e teste de instalação. Embora a aplicação de teste tenha um papel extremamente importante em V&V, muitas outras atividades também são necessárias.

O que se pode ver no cenário atual é que as empresas tem mudado seu olhar para o teste de software e tem entendido que sem o processo de teste não dá para conviver, conforme Bartié (p.04, 2002).

> Trata-se de uma tendência natural. As organizações estão buscando eficiência para conseguir sobreviver em um ambiente cada vez mais hostil – o de um mercado cada vez mais competitivo. As empresas estão buscando a tecnologia para reduzir custos e ampliar sua forma de atuação. Estão sofisticando seus sistemas para tomar decisões cada vez mais complexas, com a intenção de ganhar eficiência e controle. Tudo isso pode ser observado através de diversos indicadores econômicos, como volume de negócios feitos pela indústria de software e hardware, proporção de horas profissionais diretamente ligados à tecnologia por total de horas de profissionais, entre outros.

É fato que quando o cenário do mercado muda mais rápido do que os softwares que o suportam, para se "adequar" a isso o teste de software é quase sempre cortado para conter gastos orçamentários. E também que os benefícios são inegáveis em reter um defeito logo na sua raiz conforme Humble e Farley (p.171, 2014) relatam:

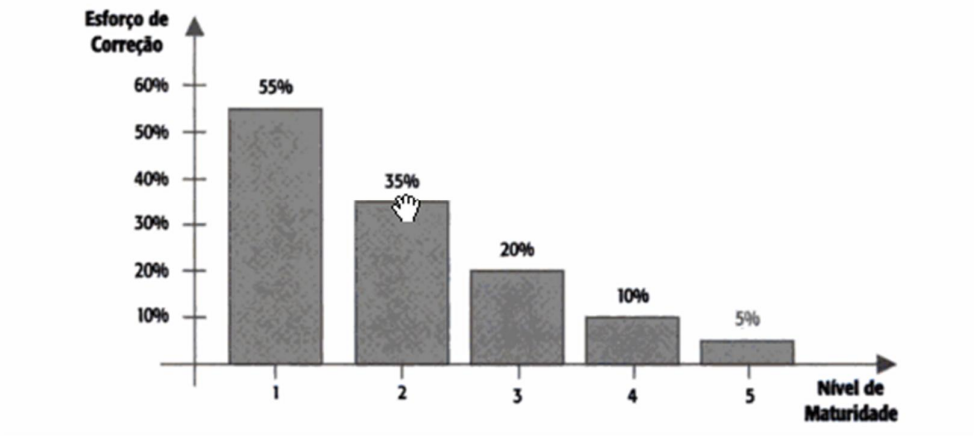
> Erros são mais fáceis de corrigir se forem detectados cedo, próximo ao momento em que foram introduzidos. Isso se dá não só porque as informações relevantes ainda estão frescas na memória daqueles que introduziram o erro no sistema, mas também porque a mecânica da descoberta da causa do erro também é mais simples. Se um desenvolvedor faz uma mudança que resulta em uma falha em um teste, e a causa não é evidente, a atitude natural a tomar é analisar tudo que mudou desde a última vez em que o sistema estava funcionando para reduzir o foco na busca.

Conforme as empresas vão crescendo e aumentando a sua consciência em relação a qualidade, automaticamente a maturidade em processos de qualidade de software crescem e os resultados tornam-se positivos segundo Bartié (p.13, 2002).

> Um dado importante que o SEI estima é o fato de que empresas de nível dedicam cerca de 55% dos esforços direcionados para corrigir defeitos do projeto de software desenvolvido. Á medida que a empresa vai adquirindo maturidade, esses índices vão sendo gradativamente reduzidos para 35%, 20%, 10% até a empresa alcançar o nível 5 e reduzir esse patamar para 5%. Imagine o impacto no tempo total de desenvolvimento, custo final do projeto, número de profissionais envolvidos e o nível de confiabilidade organizacional.

Para ilustrar as palavras de Bartié (p.13, 2002) segue a Figura 2 que mostra um gráfico referente ao nível de maturidade alinhado ao esforço de correção.

Figura 2 – Esforço de correção no desenvolvimento de software



Fonte: Bartié (2002, p. 14).

Relatórios gráficos conforme apresentado anteriormente revelam informações importantes que servirão de base para identificar o quanto após a aplicação da etapa de teste a equipe tem melhorado o seu nível de maturidade e reduzindo assim o esforço de correção, a partir desse momento também se faz importante olhar para as métricas de software afim de entender o que está acontecendo no código fonte e assim manter a cadência de maturidade, reduzindo cada vez mais o esforço de correção, ou seja um ciclo que pode dar muito certo aumentando a qualidade do produto.

2.3 MÉTRICAS DE SOFTWARE

As métricas de software têm ajudado atualmente a revelar o factual estado do software, é uma das abordagens utilizada para apontar os principais pontos a serem atacados na melhoria da qualidade, e dentro desse contexto é importante conhecer as diferenças entre métricas, medidas e indicadores. O conceito de medição é definido por Lord Kelvin (apud PRESSMAN, 2002, p.73).

> Você tem algum conhecimento sobre o que fala, quando pode medir e expressar isso em números; mas quando você não pode medir, quando não pode expressar isso em números, seu conhecimento é fraco e insatisfatório: pode ser o princípio do conhecimento, mas, não na sua opinião, você provavelmente está no estágio inicial de uma ciência.

Já sobre métricas define Martins (2007) que em processos e projetos de software as métricas possibilitam a avaliação da eficácia dos mesmos e através de dados que são coletados e comparados com dados de referência ou anteriores, permite-se assim medir o produto de

software, que são medidas indiretas do produto, incluindo funcionalidade, qualidade, complexidade, eficiência, confiabilidade, manutenibilidade, dentre outras.

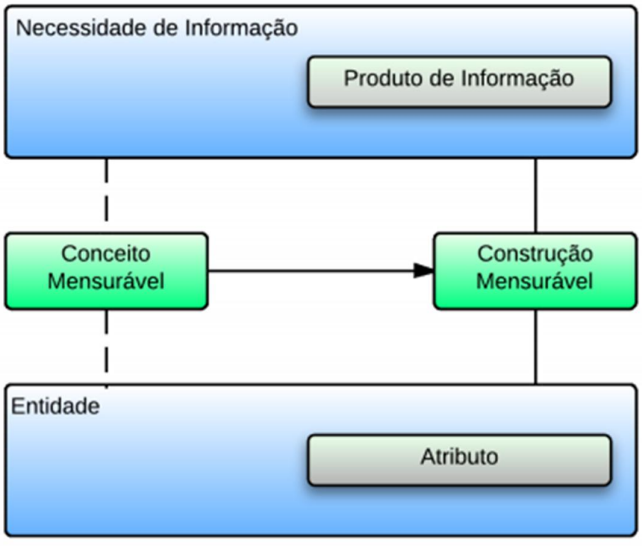
Indicadores são conhecidos no mercado como Indicador-Chave de Desempenho conforme Bonel (2015) são mensuráveis e funcionam como um medidor de desempenho, tem como principal objetivo acompanhar se as ações implementadas estão funcionando ou não de acordo com as metas definidas e dessa forma se os resultados esperados não estão sendo obtidos, então os indicadores irão revelar isso e uma ação diferente da anterior deverá ser tomada.

Para um melhor entendimento Reisswitz (p. 25, 2013) define:

> Medição: Ato de determinação de uma medida
> Métrica: Medida quantitativa do grau em que um sistema se encontra em relação a um determinado atributo.
> Indicadores: Métrica ou combinação de métricas que fornece uma compreensão de um processo/projeto/produto.

Conforme a ISO/IEC 15939 (2002) existe um processo de medição que se baseia em um modelo de informação mostrado na Figura 3, utilizado para obter resultados para cada necessidade de informação. Para conseguir chegar no resultado adequado faz-se necessário conhecer os objetivos, metas, riscos e problemas e tudo isso deve ser mensurável. A partir desse conceito é possível definir atributos para diferenciar de forma qualitativa ou quantitativa o item a ser medido que pode ser um processo, software ou um projeto.

Figura 3 – Modelo de Informação da ISO 15939



Fonte: ISO 15939 (2002, p. 14).

Para um bom aproveitamento das métricas de software, é necessário antes de utilizá-las, identificá-las, organizá-las e planejá-las conforme relata Sommerville (2003).

- Inicialmente, definir um padrão para as métricas:
- Escolha de medições a serem feitas
- Seleção de componentes a serem avaliados
- Medição de características dos componentes
- Adquirir as ferramentas necessárias
- Identificar medições anômalas

Métricas de software, é necessário que as mesmas sejam simples e práticas para um bom entendimento do engenheiro, segundo Pressman (2006) existem atributos que são muito representativos para a validação das métricas:

- Uma métrica deve ter propriedades matemáticas desejáveis: Deve estar em um intervalo significativo (exemplo 0 significa o valor mais baixo, 0,5 valor médio e 1 o valor mais alto).
- Proporcionalidade: Deve subir ou descer os valores conforme eventos positivos ou negativos aconteçam
- Cada métrica deve ser validada empiricamente em diferentes contextos antes de ser publicada ou utilizada para tomar decisões: Deve se adequar em qualquer contexto sempre considerando o fator de interesse não sofrendo interferência com outros fatores.

Entre as métricas existem aquelas que tem uma maior facilidade de avaliar do que outras, uma métrica relacionada ao tempo que o sistema leva para abrir é de certa forma mais fácil de se conseguir e interpretar se está conforme se espera ou não, porém medir a complexidade de manutenção de um certo sistema é mais complicado. Existem processos e produtos que farão parte do contexto e deverão ser considerados pois impactará no resultado final, se não forem levadas em conta poderão tornar as mesmas inválidas, além do mais que este tipo de métrica está totalmente ligada ao nível de entendimento dos desenvolvedores envolvidos Sommerville (2003).

Pressman (2006), cita as mais importantes áreas de métricas:

- **Métricas para o modelo de análise:** Avalia a qualidade da especificação de requisitos.

- **Métricas para o modelo de projeto:** Permite ao engenheiro de software avaliar a qualidade do projeto
- **Métricas de teste:** Auxiliam no projeto de casos de teste e avalia a eficácia dos testes.
- **Métricas para código fonte:** Usadas para avaliar a complexidade, manutenibilidade, testabilidade, entre outras características.

Essas áreas citadas anteriormente tem como principal objetivo determinar até onde vai cada medição, auxiliando na melhoria dos processos utilizados em cada área que está sendo medida. A qualidade final de um produto está totalmente relacionada as ações de melhoria realizadas com base nos resultados das métricas de cada área, qualidade final essa medida através da quantidade de *bugs* encontrados.

2.3.1 Métricas para o modelo de análise

Segundo Maxim e Pressman (2016, p. 659) embora relativamente poucas métricas de análise e especificação tenham aparecido na literatura, é possível adaptar as usadas frequentemente para a estimativa de projeto e aplica-las nesse contexto. Essas métricas podem analisar o modelo de requisitos com a intenção de prever o tamanho do sistema resultante, onde o tamanho é, as vezes (mas nem sempre) um indicador da complexidade do projeto e quase sempre é um indicador de trabalho cada vez maior de codificação, integração e teste.

2.3.2 Métricas para o modelo de Projeto

Segundo Maxim e Pressman (2016, p. 679) métricas para projeto consideram aspectos de arquitetura, projeto em nível de componente e projeto de interface, as métricas de projeto de arquitetura consideram os aspectos estruturais do modelo de projeto, já as métricas de projeto a nível de componente fornecem indicação de qualidade do módulo e estabelecem medidas indiretas para coesão, acoplamento e complexidade e as métricas de projeto de interface fornece indicação da facilidade com que uma GUI (interface gráfica do utilizador) pode ser usada, aspectos da interface com usuário e estética da aplicação conteúdo e navegação.

2.3.3 Métricas de Teste

Segundo MAGAZINE (2008) a essência dos teste é encontrar e fornecer informações, com isso os resultados dos esforços de teste serão utilizados pelos stakeholders para verificar o andamento do projeto e assim utilizar essas informações para tomar decisões importantes sobre o produto. Por exemplo, em um projeto que está a dois dias do prazo para entregar ao cliente, em que são reportadas apenas informações referentes aos defeitos em aberto e as tendências dos defeitos, esquecendo de métricas importantes como cobertura dos testes ou ainda sobre a eficácia dos testes, decisões podem ser tomadas sem conhecimento suficiente dos fatos.

2.3.4 Métricas orientadas ao Código

Métricas de código podem caracterizar bem o produto de software em todas as suas fases de desenvolvimento e assim revelar os esforços necessários para começar as medições e acompanhamento da qualidade interna do produto. Segundo Meirelles(2013), a avaliação de qualidade de código fonte no início do desenvolvimento pode ser valiosa a fim de auxiliar equipes inexperientes durante as etapas seguintes do desenvolvimento de software.

Segundo Novais (2016), métricas de qualidade são necessárias quando se necessita manter uma aderência à sua arquitetura no ciclo de desenvolvimento de software, reduzindo custo, esforço, alterações e manutenções, além de se transformar em fator motivacional para a equipe de desenvolvimento que pode perceber que está em um ambiente que se preocupa com o padrão técnico e busca constantemente minimizar os principais débitos técnicos de um projeto de desenvolvimento.

Quando se analisam as métricas de qualidade está sendo feito na verdade um gerenciamento dos débitos técnicos como já mencionado de um projeto de desenvolvimento de software e isso é extremamente fundamental, quanto maior o débito técnico de um projeto de desenvolvimento, menor será a produtividade e consequentemente os gastos financeiros serão elevados, porém ao contrário se torna um diferencial se tornando um motivador para melhorar a qualidade ao utilizar abordagens que venham a revelar esses débitos que é o caso de se utilizar as próprias métricas, sem isso seria impossível identificar as “dívidas técnicas” do código fonte.

2.4 MÉTRICAS DE CÓDIGO FONTE

Conforme a ISO/IEC 25000 define qualidade de software essencialmente em qualidade interna e qualidade externa. A qualidade externa se trata da visão do produto em relação a sua própria execução a nível de cliente e a qualidade interna resume-se basicamente em avaliar documentos, modelos e código fonte, onde esses valores de qualidade interna podem ser utilizados para prever os valores de qualidade externa que o produto pode apresentar.

A categorização das métricas podem ser definidas de maneiras diferentes, como métricas diretas e indiretas, orientadas a tamanho e função ou métricas de produto e produtividade entre outras que serão citadas logo mais.

2.4.1 Linhas de código (LOC)

A técnica de medida por linhas de código é uma das medidas mais antigas para determinar esforço, tamanho e produtividade no ciclo de desenvolvimento de software e também uma das mais simples a mesma pode trazer alguns indicadores interessantes conforme menciona Daniel (2016):

- Analisar qualidade e produtividade do processo de desenvolvimento e manutenção do produto de software;
- Qualificar a performance técnica dos produtos do ponto de vista do desenvolvedor;
- Qualificar a performance dos produtos pela perspectiva do usuário;
- Utilizadas para comparar a produtividade de diferentes técnicas e tecnologias;
- Entender e aperfeiçoar o processo de desenvolvimento de software;
- Determinar parâmetros como quantidade de teste necessário e impacto de mudanças;

Imagine-se em um cenário onde é necessário prestar manutenção em um código com uma grande quantidade de linhas de código, isso demonstra que ao estimar uma tarefa não será muito trivial, porém com base em uma métrica igual a essa ficará mais fácil de justificar perante ao grupo gestor do projeto a reavaliação de prazos com base em pontos de incerteza que podem existir no desenvolvimento.

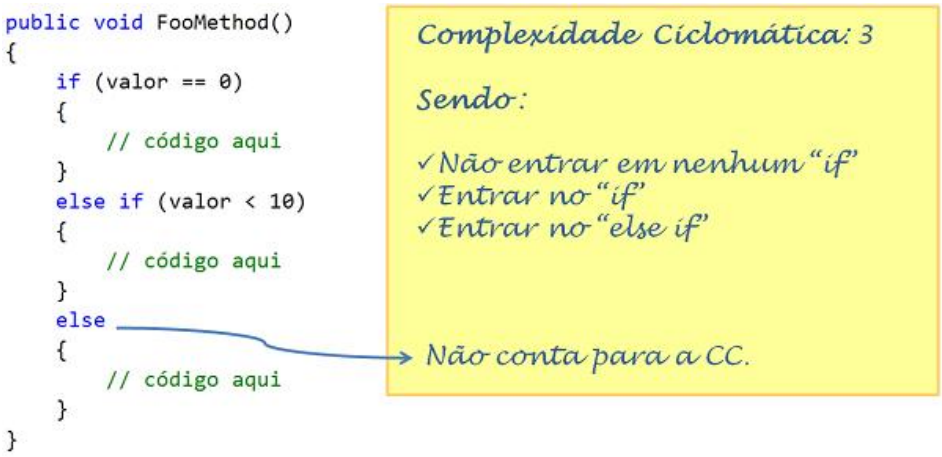
2.4.2 Média de complexidade ciclomática por método (ACCM)

Através dessa métrica é possível verificar quais caminhos ou decisões possíveis de execução dentro de um código e é mencionada por Charney (2016), em seu artigo, onde a métrica de complexidade ciclomática foi introduzida por Thomas McCabe em 1976 e provavelmente é a métrica lógica mais útil.

Ela mede o número de caminhos linearmente independentes através de um programa, por exemplo, para uma função simples que não tem condição, haverá um único caminho, dessa forma esse tipo de código é fácil de executar, porém programas que tem muitas condições tendem-se a ser mais difíceis de executar e essa dificuldade pode aumentar quando existirem vários caminhos e por isso, a dor de cabeça aumenta ao desenvolver, depurar ou testar um software que contém muitos caminhos.

Segundo Daniel (2016), quando obtemos como resultado 30 (ou mais) de CC, temos um código extremamente complexo de entender e prestar manutenção, para um melhor entendimento considere o exemplo da figura 4.

Figura 4 – Cálculo de Complexidade Ciclomática



Fonte: Daniel (2016)

Conforme pode ser visualizada na figura anterior, fica fácil de entender como a complexidade ciclomática de uma seção do código fonte é interpretada e como é definida pela quantidade de caminhos independentes que o código percorre durante a execução do sistema, isso torna uma forma de medir quão grande e complexo é o sistema e até mesmo quanto de esforço para executar uma bateria de testes ou até mesmo dar manutenção nesse trecho do código, em seguida outra métrica que pode informar outra característica importante do código.

2.4.3 Resposta para uma classe (RFC)

Essa métrica se refere a quantidade número de métodos da classe somando-se ao número de métodos que ela chama, conforme Chidamber (1994), é definida pelo número de diferentes métodos que podem ser chamados em resposta a uma mensagem para um objeto da classe ou por algum método na classe, isso também inclui todos os métodos que é acessado dentro da hierarquia.

Para Rosenberg (2016), um número que pode ser considerado aceitável para esse tipo de métrica seria 100, sendo que, para a maioria das classes este número é menor ou igual a 50.

2.4.4 Profundidade na árvore de herança (DIT) / Número de subclasses (NOC)

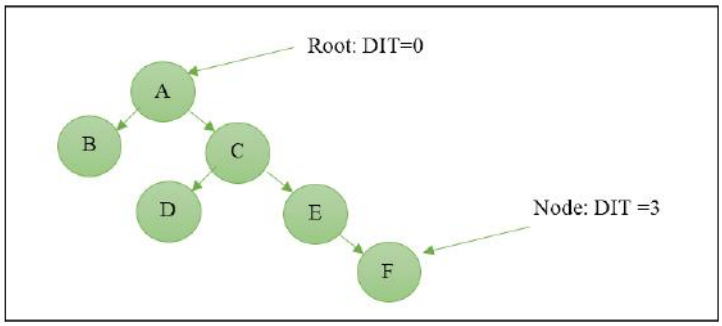
Apesar de determinações diferentes DIT e NOC são métricas relativamente semelhantes por representarem o mesmo tipo de informação, a árvore de herança dentro do código porém podem ser interpretadas de forma diferente.

DIT é uma métrica que mede a profundidade que uma classe se encontra na árvore de herança, e caso haja herança múltipla, DIT mede a distância máxima até o nó raiz da árvore de herança e dessa forma se ela não herda nada tem DIT igual a 0 se herdar tem profundidade igual a 1 e assim respectivamente segundo Chidamber e Kemere (1994).

NOC mede a quantidade de filhotes que uma classe possui, caso ninguém herde dela, o valor é 0 e vai aumentando em 1 para cada classe que a estende de forma direta, e assim, filhos de filhos não são contados.

Na Figura 5 é apresentado um exemplo de DIT e NOC para ilustrar o conteúdo anteriormente explorado:

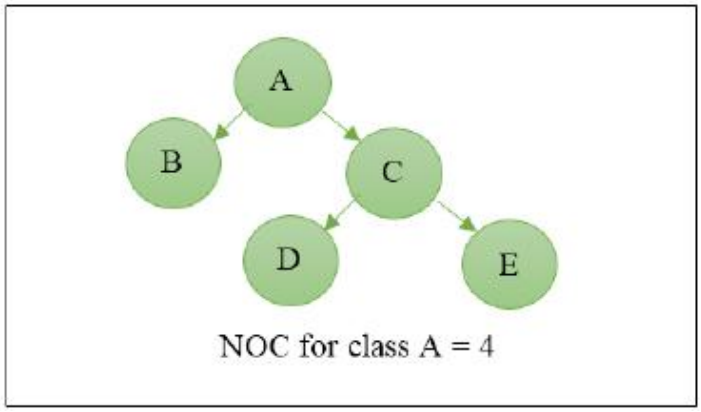
Figura 5 – Exemplo de DIT



Fonte: Aljadaa (2016)

Nessa métrica referente a DIT pode se verificar duas características, uma positiva que é fato da utilização da característica de herança que faz com que a reutilização de código fonte, pelo lado negativo é o fato de que quanto mais profundo a classe estiver na árvore de herança mais imprevisível e complexa a mesma será, essa métrica está totalmente vinculada a próxima métrica.

Figura 6 – Exemplo de NOC



Fonte: Aljadaa (2016)

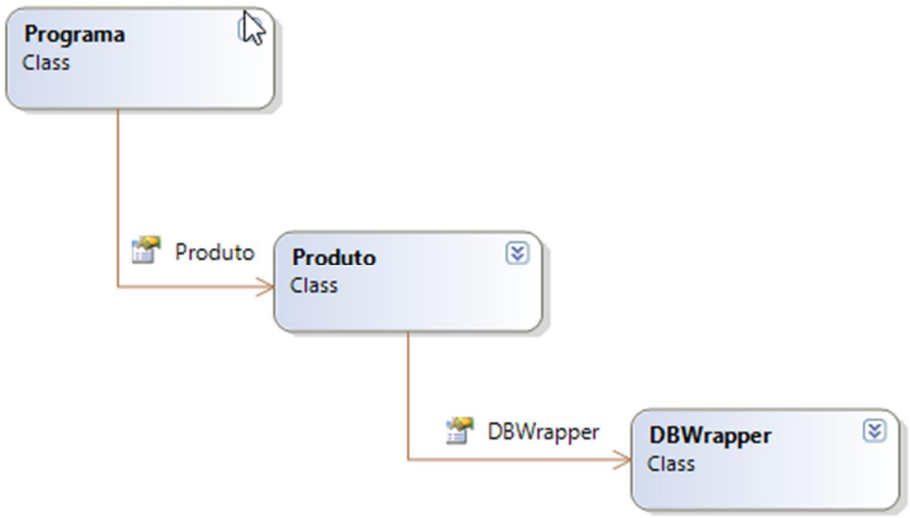
A métrica de NOC pode informar a quantidade de classes filhas que uma determinada classe pode ter e isso como pode ser vista na figura anterior, pode determinar duas características sendo uma positiva pelo fato também do reuso de código fonte e negativa pelo fato da complexidade em onde alterar uma classe deverá ser alterada suas classes filhas causando um impacto com consequências negativas no código, pois muitos métodos que são usadas nessa mesma classe(pai) está sendo utilizado nas classes filhas. Isso mostra que são métricas extremamente delimitadas e trazem resultados bem próximos assim como as próximas métricas a serem mencionadas em seguida de acoplamento e coesão.

2.4.5 Acoplamento entre Objetos (CBO)

Segundo Hirama(p.83, 2012), antes de começar a codificação de um software, é importante definir os módulos e realizar algumas definições em relação a estrutura de relacionamento entre eles. Sendo assim, surge o acoplamento que se refere ao grau de dependência de um módulo em relação ao outro, quanto menos os módulos forem acoplados o desempenho e manutenibilidade do mesmo serão melhores.

Os conceitos de coesão que será comentado em seguida e acoplamento são muito próximos porém existem diferenças um para com o outro, abaixo um exemplo de acoplamento na figura 7 onde mostra o quanto pode ser custosa a manutenção de um código que esteja fortemente acoplado por causa das mudanças afetarem toda a cadeia de classes.

Figura 7 – Acoplamento



Fonte: Cadu (2016)

Na figura 7 pode-se ver o forte acoplamento entre as classes que demonstra que uma depende da outra e dessa forma qualquer manutenção no código fonte deverá ser replicada para seus dependentes, tornando assim o trabalho demorado e consequentemente bastante custoso, já a próxima métrica mostrada em seguida mostra exatamente o contrário disso.

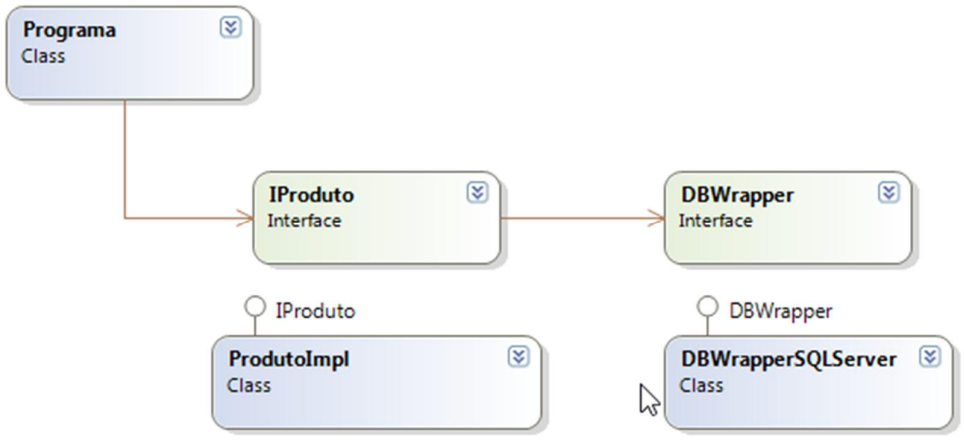
2.4.6 Falta de Coesão em métodos (LCOM)

A falta de coesão está diretamente ligada com a métrica de acoplamento mencionada anteriormente, porém ambos devem ser atendidos. O conceito de coesão se refere ao grau de integração de funções que tem objetivos comuns dentro de um módulo, Hirama(p.83, 2012).

Segundo Cadu (2016) no mundo real nem sempre é possível ter um baixo acoplamento e alta coesão, porém cabe ao arquiteto tomar decisões corretas que dependendo da mesma poderá ser revertida ou não, onde ter classes com responsabilidades claras e um baixo acoplamento embora não seja fácil de construir traz inúmeros benefícios em relação a baixo impacto em manutenção,

gerenciamento e mudança de negócio facilitados, abaixo segue um exemplo na figura 8 de como resolver a situação anterior de acoplamento deixando as classes mais coesas.

Figura 8 – Coesão



Fonte: Cadu (2016)

Observa-se na figura 8 que aqui nada mais é uma resolução do problema encontrado na figura 7 que mostrava o forte acoplamento entre as classe, nessa figura percebe-se a alta coesão criada com a inclusão de classes abstratas que fazem com que o relacionamento de dependência total entre elas seja quebrado e irão flexibilizar possíveis mudanças que venham ocorrer no código fonte.

2.4.7 Violações de código

Durante a evolução do produto de software existem algumas anomalias que podem ser introduzidas através de codificação que não são aderentes ao modelo arquitetural que foi especificado para a aplicação antes mesmo da codificação atual. Em etapas anteriores arquitetos de software definiram como serão as relações de dependências de módulos e componentes entre si e com a integração com sistemas externos, e quando o desenvolvedor realiza alguma alteração poderá provocar algum desvio de implementação em relação a arquitetura planejada.

2.4.8 Duplicações de código

Considerado um débito técnico e também um dos problemas mais comuns para softwares que crescem de tamanho exponencialmente, pode tornar a manutenção do código

extremamente custosa e dificultar o trabalho dos desenvolvedores que agora ao invés de entender 200 linhas de código devem considerar 1000 linhas de código por exemplo avaliando seus impactos e entendendo elas.

Segundo Junior (2014) vários fatores podem levar à duplicação de código, e entre elas estão a de ter mais de um desenvolvedor no mesmo projeto onde isso é bastante comum, e também os mesmo podem estar trabalhando em funcionalidades parecidas e acabam criando códigos parecidos ou que fazem a mesma coisa sendo que as mesmas poderiam ser unificadas e reutilizadas.

3 METODOLOGIA

Esta seção tem por objetivo abordar o tipo e a metodologia de pesquisa, também é definido o escopo e a forma com que as atividades serão executadas afim de atender os objetivos gerais e específicos desse trabalho.

3.1 TIPO DE PESQUISA

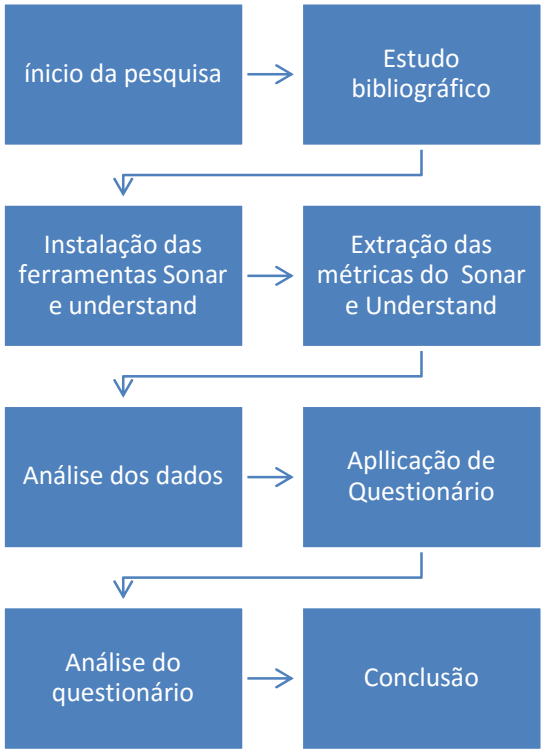
A metodologia de pesquisa é o estudo dos métodos, ou então as etapas a seguir em um determinado processo, o objetivo é captar e analisar as características dos vários métodos indispensáveis, avaliar suas capacidades, potencialidades, limitações ou distorções e criticar os pressupostos ou as implicações de sua utilização e também explicar de forma minuciosa, detalhada, rigorosa e exata de toda a ação desenvolvida no método (caminho) do trabalho de pesquisa.

Este trabalho utilizará da análise de conteúdo, visto que as análises serão feitas sobre o código fonte. Dessa forma é utilizada inicialmente a metodologia quantitativa, pois são levantados números em relação as métricas de código fonte do sistema, e por fim, a análise qualitativa através de um questionário relacionado ao tema dirigido a todos os integrantes das equipes de desenvolvimento.

3.2 ETAPAS DE DESENVOLVIMENTO DA PESQUISA

Nesta seção são mencionadas as etapas da pesquisa que findam o seu propósito, onde constam o escopo, as técnicas, um fluxograma de como é a pesquisa, o experimento e as análises que serão vistas nas próximas páginas.

Figura 9 – Fluxograma das etapas de pesquisa



As etapas acima mencionadas foram seguidas durante a execução desse trabalho, a primeira etapa iniciou-se pelo estudo bibliográfico passando depois pela etapa de instalação das ferramentas Sonar e Understand utilizadas para a extração das métricas, depois passando pela etapa de análise dos dados coletados com base em relatórios retirados das ferramentas, e finalizando com a aplicação de um questionário para os membros das equipes ágeis com sua posterior análise baseado nas respostas dos membros sobre o quão pode ser útil a utilização de métricas de código em suas tarefas diárias relacionadas ao ciclo de desenvolvimento de software.

3.2.1 Escopo

A pesquisa tem por objetivo realizar a avaliação da adoção de práticas relacionadas a utilização de métricas de código fonte em equipes ágeis através das ferramentas Understand/Structure e Sonar mostrando assim o impacto e ações tomadas com base nos resultados afim de melhorar qualidade do desenvolvimento do produto.

O estudo deste trabalho será um único software que vem sendo evoluído ao longo de vinte anos e sendo assim recebe evoluções constante em períodos muito curtos para vários clientes, sendo assim a aplicação de métricas será limitada a versão 4.0.19 do sistema.

Desta forma este trabalho abordará algumas das principais métricas de código fonte que puderam ser coletadas e acompanhadas pela equipe de arquitetura de desenvolvimento da empresa durante a versão acima mencionada.

A extração das métricas estará limitada a utilização das ferramentas Sonar e understand e entre as métricas que foram extraídas são: Linhas de código(LOC), Complexidade ciclomática(CC), Fator de acoplamento (COF) e Falta de coesão em métodos (LCOM), Duplicações e Violações de código.

3.2.2 Ameaças ao experimento

Esta pesquisa possui algumas informações que podem ser limitadas, e dessa forma existe a preocupação em alguns pontos que foram elencados no Quadro 1.

Quadro 1: Ameaças a pesquisa

Ameaça	Ação
Existe a preocupação da não autorização dos dados coletados da empresa em se tratando de sigilo de negócio	Solicitada autorização junto ao Gerente da Unidade e formalizado através de e-mail.
Problemas nas ferramentas podem ocasionar Falsos-positivos na extração de algumas métricas, fazendo com que a confiabilidade das extrações seja comprometida.	Ferramentas vem sofrendo atualizações constantes para minimizar tais impactos.

3.2.3 Definição das técnicas

Para findar os objetivos desta pesquisa serão apresentadas o resultado das coletas das métricas de um sistema comparadas com outro sistema definido nesse trabalho como software de terceiro, apresentando as ações tomadas com base na leitura e interpretação de cada métrica, e também mostrar o resultado da opinião dos membros das equipes de desenvolvimento em relação ao conceito de código limpo, utilização dos resultados das métricas e sobre seu conhecimento e importância em possuir uma forma de ter a visibilidade do código fonte.

3.2.4 Métodos de Coletas de Dados

Para esta pesquisa foram realizadas extração dos relatórios de métricas dentro de um período e feito o acompanhamento de cada uma das métricas avaliando assim a evolução e tomando as ações respectivas para atacar os resultados para transformá-los em medições positivas para os períodos seguintes.

O monitoramento das métricas via understand/structure e sonar se referiram a versão 4.0.19 do PGE.net em um período de 9 meses até a presente data desde que a versão 4.0.18 foi liberada. O PGE.net hoje é o sistema a qual é feito o maior trabalho de evolução dentro do ciclo de desenvolvimento no projeto de procuradorias na empresa Softplan.

3.3 DELIMITAÇÕES DO TEMA DE PESQUISA

Não faz parte do escopo desse trabalho aprofundar o conhecimento em conceitos em relação a padrões de código ou codificação, também não serão objeto do estudo discutir que tipos de técnicas podem ser aplicadas no código fonte para melhorar o mesmo com base nos resultados das métricas.

Também não faz parte desta pesquisa mostrar percentuais de diminuição de falhas ou ainda mensurar a quantidade de retrabalho que pode ser diminuída com base no resultado das métricas, ou ainda mencionar que tal métrica é correta ou não para entender e compreender a qualidade do código.

4 EXPERIMENTO

Nessa etapa é apresentada a avaliação dos resultados através da verificação dos percentuais das métricas coletadas na release específica da versão 4.0.19.

Esse capítulo está dividido em seções que demonstram as ferramentas utilizadas para extração das métricas de código, definição do software a ser medido, o porquê da escolha dessas ferramentas, a demonstração da coleta dos dados das métricas de complexidade ciclomática, linhas de código, duplicações e violações de código da ferramenta Sonar e também a extração das métricas de acoplamento e coesão realizadas por meio da ferramenta Understand/Structure e finaliza-se demonstrando o questionário aplicado as equipes em relação a utilização das métricas no seu cotidiano e as ações que já foram realizadas com base nos resultados obtidos.

4.1 FERRAMENTAS PARA MEDIÇÃO - SONAR

O Sonar é uma ferramenta de código aberto construída para avaliar a qualidade do código que está sendo desenvolvido. Questões de arquitetura, código duplicado, pontos potenciais de bugs, nível de complexidade e cobertura dos testes entre outras métricas são avaliadas pelo Sonar, essas informações vêm a calhar para a equipe de desenvolvimento afim de passar um feedback da qualidade do seu código segundo Araújo(2016).

A avaliação do código fonte se faz através da submissão do mesmo ao Sonar por meio do SonarQube Runner ou plug-ins do Maven e Ant e após isso o Sonar analisa e gera os resultados que ficam disponíveis no dashboard do projeto.

O Sonar gerencia a qualidade do código separando as métricas de código em sete eixos de qualidade e onde cada um destes eixos é relacionado a um dos sete “pecados espirituais mortais” que são cometidos pelos desenvolvedores listados abaixo segundo Novais(2016).

- Arquitetura e Projeto;
- Duplicações;
- Testes unitários;
- Complexidade;
- Bugs potenciais;
- Regras de codificação;
- Comentários

Ferramentas como o Sonar tornam possível apresentar informações que abrange diversos aspectos relacionados ao código e a principal vantagem de se utilizar a ferramenta é de poder gerenciar o rumo do código e traçar ações corretivas para melhorar a produção com qualidade do código fonte e consequentemente melhorar o produto final.

4.2 FERRAMENTAS PARA MEDIÇÃO – UNDERSTAND/STRUCTURE

O Understand se trata de uma ferramenta de análise de código estático fonte desenvolvido pela empresa Scitool, projetado para auxiliar a explorar o código fonte que podem ser grandes ou complexos e com bastante desenvolvimento legado que muitas vezes sem nenhuma documentação. A ferramenta é personalizável e extensível e ajuda a visualizar e entender o código legado complexo e realizar análises de impacto fornecendo métricas poderosas segundo Scitools (2016).

A ferramenta proporciona uma flexibilidade muito grande sendo possível utilizar com várias linguagens de programação(C / C++, Java, Pascal, Python entre outras) e dessa forma por ter o suporte também ao Delphi foi de bastante valia para esse trabalho.

O Understand calcula várias métricas comuns à maioria das ferramentas de métricas de código como número de linhas de comentário e número de linhas de código, ainda disponibiliza 29 métricas de orientação ao objetos como profundidade máxima da árvore de herança (DIT), e 27 métricas de complexidade de código como complexidade ciclomática, que indica quão complexo o código é a partir do número de caminhos independentes segundo Millani(2013, p. 23).

Em conjunto com o understand a ferramenta structure trabalha para reorganizar a base do código e montar diagramas que tornam a visibilidade do código mais intuitiva ajudando o desenvolvedor a atacar os pontos de dependência e demais métricas.

O Structure é uma ferramenta que se acopla ao understand pois dá visibilidade dos resultados coletados pelo mesmo através de relatórios para interface do usuário. O structure reorganiza sua base de código para reduzir riscos e mapear diretamente o código criando diagramas de arquitetura que mapeiam os arquivos de origem reais e podem ser verificados na codificação e tempo de compilação, fornecendo um entendimento comum e uma orientação contínua para os desenvolvedores segundo Klocwork (2016).

4.3 FERRAMENTA ESCOLHIDA

A decisão pela escolha de ambas as ferramentas citadas anteriormente para coletar as métricas a serem relatadas nesse trabalho, partiu antes de mais nada na facilidade em que o sonar disponibiliza as métricas em sua interface gráfica afim de dar uma visão mais ampla ao arquiteto que está monitorando as mesmas, além do fato das ferramentas se completarem dando uma visibilidade e informação ao desenvolvedor em relação a saúde de seu código também.

A ferramenta understand/structure foi levada em consideração também pelo fato de trazer métricas importantes que também são citadas nesse trabalho e que não podem ser monitoradas diretamente pelo sonar e também pela forma de integração junto ao sonar o que facilita a visibilidade das suas amostragens para o desenvolvedor pois também mostra diretamente na interface do sonar seus resultados além de poder ter uma interface própria para resgatar os resultados.

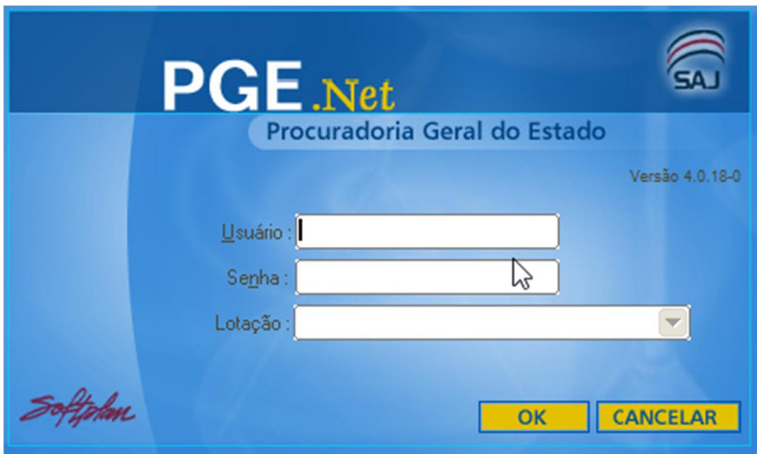
4.4 CENÁRIO DO EXPERIMENTO

A Softplan é uma das maiores empresas do Brasil no desenvolvimento de softwares de gestão. Atualmente suas soluções estão presentes em todos os estados brasileiros, em países da América Latina e nos Estados Unidos. Ao longo desses anos, a Softplan se especializou no desenvolvimento e na implantação de softwares de gestão para os segmentos de Justiça, Infraestrutura e Obras, Gestão Pública, Projetos Cofinanciados por Organismos Internacionais e Indústria da Construção.

Dentro desse cenário foi escolhido o software PGE.net para a realização das métricas, um programa desenvolvido em Delphi que atualmente está implantado em órgãos vinculados aos governos estaduais e municipais auxiliando procuradores estaduais e municipais no controle e gestão de processos judiciais onde o estado ou município se coloca como parte dos mesmos.

Também dentro desse contexto foi eleito um software da empresa SP denominado software de terceiro, nesse trabalho afim de extrair as mesmas métricas que foram retiradas do pge.net e compará-las e compreendendo-as individualmente facilitando a visualização da saúde do código presente no experimento.

Figura 10 – Software PGE.Net



Fonte: Sofplan (2016)

O PGE.net com sua tela inicial apresentada na Figura 10 é um software desenvolvido para atender as procuradorias municipais e estaduais dentro da esfera judicial brasileira, através do software é possível ter visibilidade e proceder com os trâmites e movimentações de forma virtual em processos que atualmente não utilizam mais papel nos tribunais de justiça do Brasil, e dessa forma auxiliam o estado a tornar célere as discussões judiciais onde o mesmo está envolvido, já que a procuradoria tem o papel de atuar como representante do governo em questões judiciais.

O software é desenvolvido através da linguagem Delphi e atualmente conta com uma equipe aproximada de 60 pessoas que trabalham nas funções de analistas implementadores, analistas de testes, analistas de sistemas e analistas de suporte aos cliente provendo novas soluções que comportem a evolução constante do sistema.

Atualmente em seu desenvolvimento são aplicadas técnicas que complementam a experiência do usuário como tecnologias de integração com outros sistemas da esfera judicial via protocolos de comunicação SOAP e serviços de visualização Web que utiliza-se de ferramentas como JBOSS, entre outras. O pge.net hoje conta com um número total de 11 clientes ativos de PGE's distribuídos nas capitais em várias regiões do Brasil.

4.5 DEFINIÇÃO DAS MÉTRICAS

Nesta seção define-se a escolha das métricas a serem extraídas e demonstradas nesse trabalho são elas:

- Linhas de Código (LOC);

- Complexidade Ciclomática (ACCM);
- Falta de coesão em métodos (LCOM);
- Acoplamento entre Objetos (CBO);
- Violações de código;
- Duplicações de código

Os critérios para a escolha das métricas baseou-se no fato das mesmas serem passíveis de interpretação na linguagem de programação Delphi que é a linguagem utilizada para desenvolvimento do software a ser analisado. Neste mesmo processo de escolha também se levou em consideração o fato das métricas serem passíveis de gerar relatórios a partir das ferramentas Sonar e Understand.

Com base nisso será possível demonstrar o resultado da extração de cada métrica e avaliar separadamente cada uma delas comparando-a com os resultados de softwares de terceiro, para isso será avaliado a versão 4.0.19 como já mencionado.

4.6 EXECUÇÃO DA COLETA DE DADOS

Na seção anterior, onde foram definidas as métricas a serem coletadas é necessário passar a fase de executar propriamente a coleta das métricas nesta seção. Nesse momento será realizado a coleta de informações das métricas.

Demonstrar a evolução do sistema ao ponto que o sistema cresce em tamanho e complexidade por causa das manutenções constantes e com isso mostrar como as métricas vêm se mantendo no patamar de qualidade exigido pelo mercado controlando-se assim de forma mais adequada o sistema visto que anteriormente não se tinha a visão ampla do código em se tratando de métricas.

Dessa forma é necessário recuperar o código fonte a partir da release citada e assim executar as ferramentas Sonar e recuperar os valores para cada uma das métricas mencionadas, e assim será necessário levantar as informações de todas as ações que foram tomadas para que tais métricas pudessem ter uma evolução ao longo do tempo.

4.6.1 Métricas a partir do Sonar PGE.net

Na figura 11 uma figura extraída do Sonar do software PGE.net o que demonstra uma breve visualização de métricas do software, nessa figura tem-se as métricas de violações

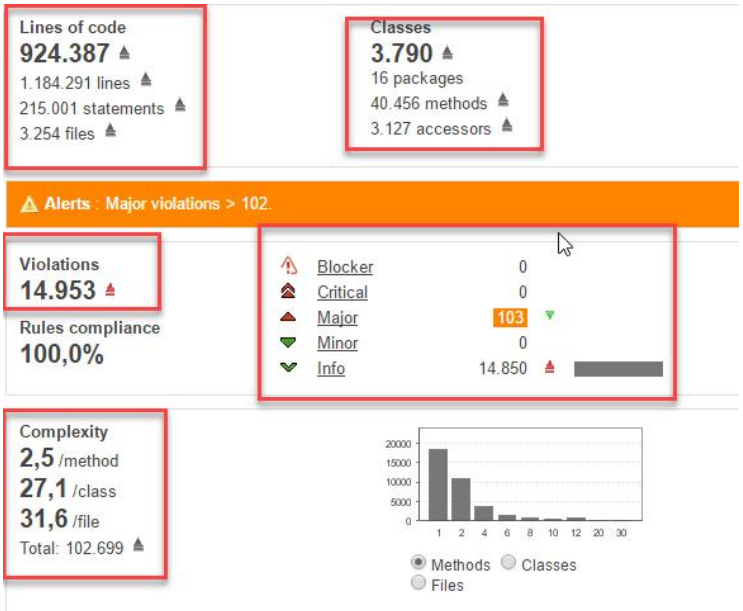
separadas por severidade e percentual de padrões de codificação, porém o sonar não se limita somente a isso como pode vislumbrar nas outras figuras a seguir demais métricas.

Figura 11– Métricas do PGE.net no Sonar

Fonte: Sonar Softplan (2016)

A Figura 12 apresenta a tela inicial do Sonar e já pode-se observar alguns valores que demonstram algumas das métricas da versão que foram identificadas no projeto como a quantidade de violações classificadas por severidade e a métrica de complexidade Ciclomática (*Complexity Ciclomatic*) além do percentual para conformidade de regras de codificação (Rules compliance).

Figura 12 – Demais métricas do PGE.net no Sonar



Fonte: Sonar Softplan (2016)

Na figura 12 pode-se identificar outras métricas presentes na tela inicial do sonar como a quantidade de Linhas de código (Lines of code), total de classes (Classes), percentual de comentários (Comments), percentual de Duplicações no código (Duplications) e também a métrica de Complexidade Ciclomática, onde esses dados mais pra frente servirão de comparativo com software de terceiro.

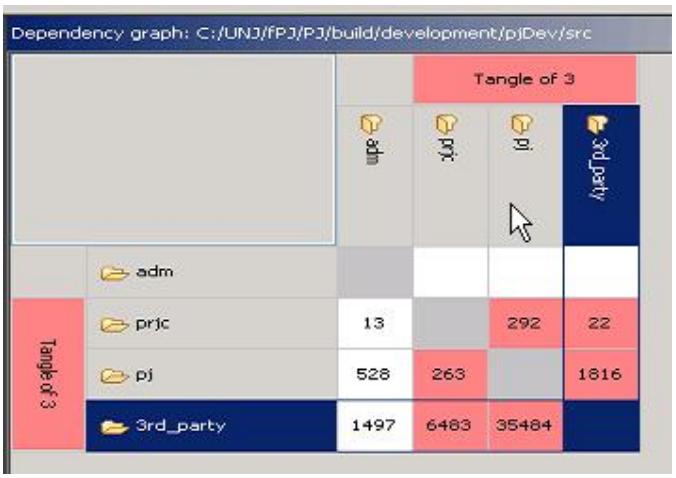
4.6.2 Métricas a partir do understand/structure Pge.net

O understand vem sendo uma ferramenta amplamente utilizada na gestão do código fonte afim de prover análises estáticas do legado e assim visualizar as métricas que são produzidas pela ferramenta afim de otimizar o design do software. Atender os padrões da indústria e prevenir problemas evitando assim desastres quando se fala de código fonte é uma das funções do understand (SCITOOLS, 2016).

O structure101 é uma ferramenta utilizada no ciclo de desenvolvimento afim de permitir que a equipe de desenvolvimento organize a sua base de código capacitando arquitetos para trabalhar com a equipe mantendo as regras de arquitetura, diagramas e lista de ações que organizam a base de código em uma hierarquia modular com acoplamento baixo e controlado vide Structure101 (2016).

A seguir tem-se a figura 13, que traz uma matriz de dependência extraída das medições via understand/structure que mostra a relação de coesão e acoplamento existente dentro do sistema pge.net.

Figura 13– Matriz de Dependência Understand – Coesão e Acoplamento



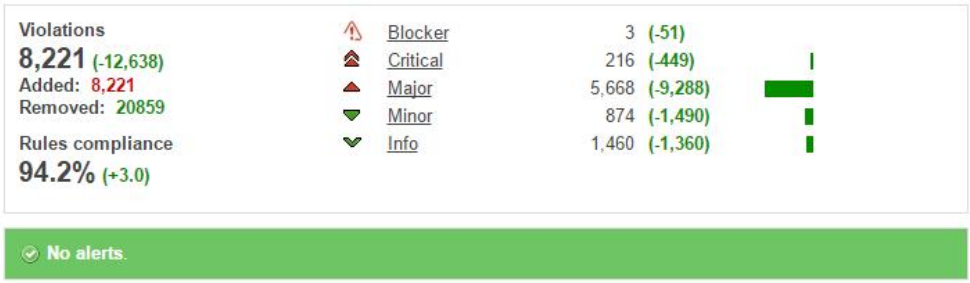
Fonte: sonar softplan(2016)

No exemplo anterior, todos os valores que se encontram acima dos 3 quadrados cinza na diagonal(292, 22 e 1816) significa a quantidade de objetos que são utilizado por outros objetos como exemplo podemos verificar que o objeto PJ é utilizado 1,523 vezes pelo método PRJC fazendo com que o mesmo se torne um objeto de baixa coesão e alto acoplamento.

4.6.3 Métricas a partir do Sonar de Software de Terceiro

A título de comparação tem-se na figura 15, os resultados apresentados do Sonar de um software de terceiro o que demonstra as mesmas métricas extraídas do Pge.net na seção anterior. Nele podemos fazer uma breve comparação de métricas com o mesmo, nessa figura temos as métricas de violações separadas por severidade também e percentual de padrões de codificação.

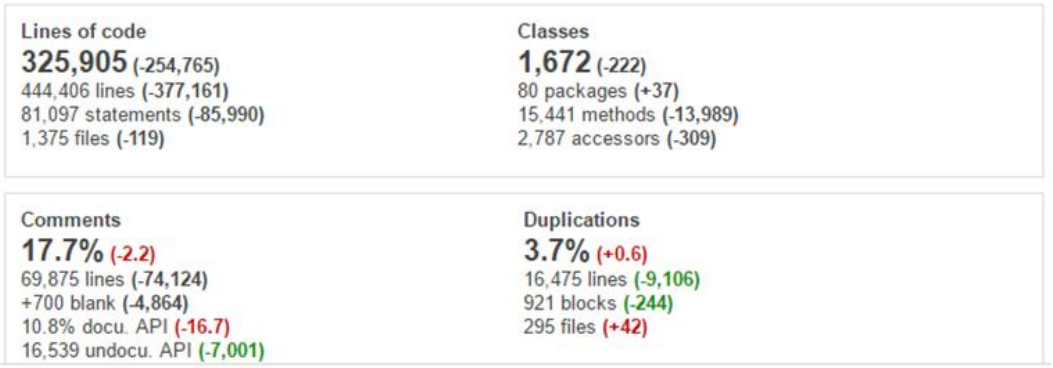
Figura 14– Métricas sonar I de software de terceiro



Fonte: Sonar softplan(2016)

Pode se perceber algumas diferenças na figura 15, somente por conta da quantidade de violações adicionadas e removidas e também as regras de codificação alcançadas com um pequeno gráfico na direita, porém os resultados gerais podem ser comparados com as métricas já extraídas do Pge.net.

Figura 15– Métricas sonar II de software de terceiro



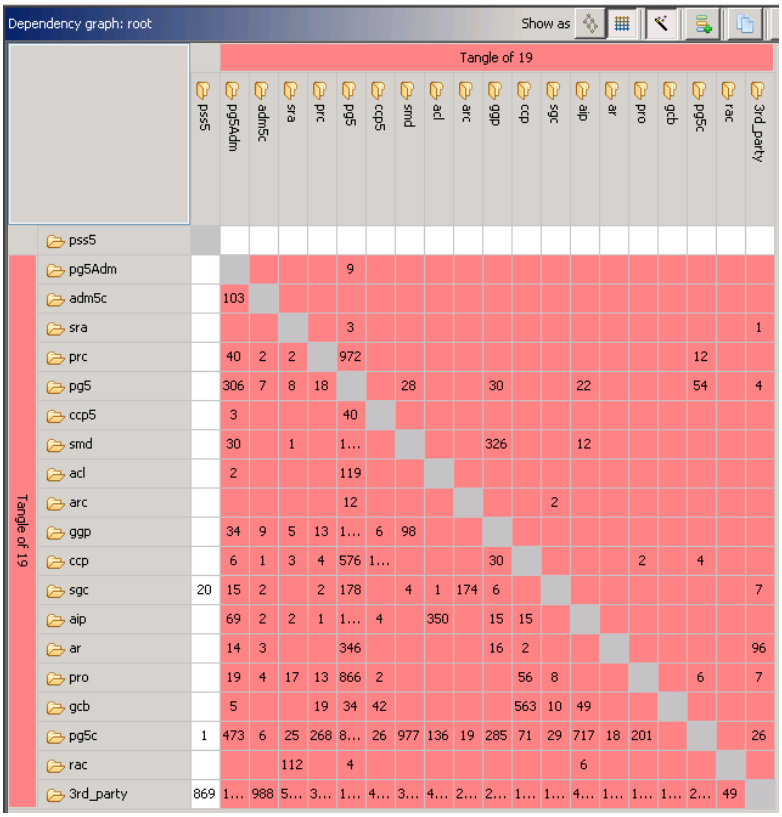
Fonte: Sonar softplan(2016)

Na figura 16 métricas de linhas de código, classes, percentual de comentário e duplicações de código, ou seja, mesmas métricas extraídas também do pge.net e que dessa forma podem ser comparadas afim de ter uma idéia de quanto o código pode estar perante demais softwares do mercado.

4.6.4 Métricas a partir do understand/structure Software de Terceiro

As métricas do understand puderam ser retiradas afim de comparativo conforme pode ser visualizado na figura 17.

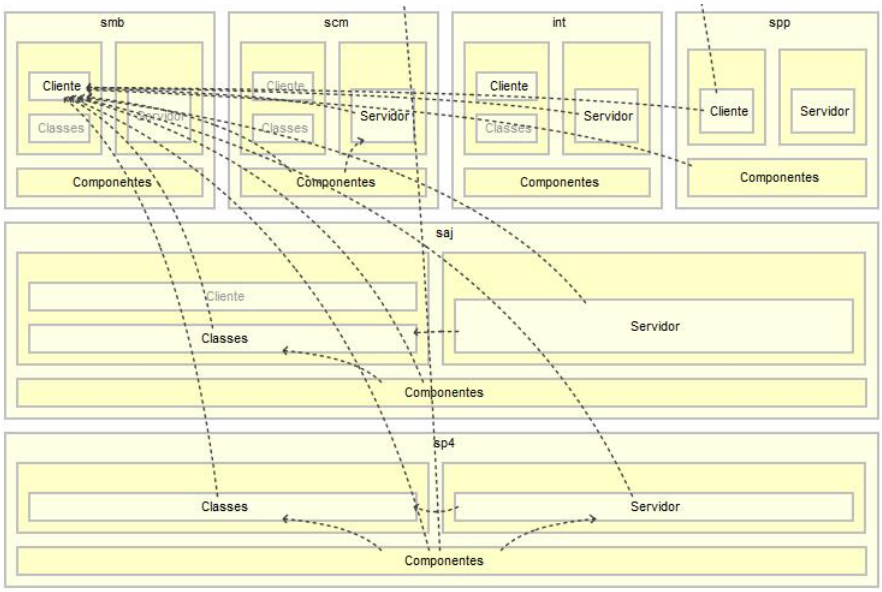
Figura 16– Métricas understand I de software de terceiro



Fonte: Sonar Software de terceiro (2017)

Na figura acima, pode se visualizar as métricas de acoplamento e coesão assim como foi feito para o pge.net, com base nisso podemos compará-los afim de verificar o nível de o quanto está coeso ou não o código assim como o quanto possui baixo ou alto acoplamento, verifica-se através da mesma que o nível do software de terceiro está melhor.

Figura 17– Métricas understand II de software de terceiro



Fonte: Sonar software de terceiro (2017)

Na figura 18 apresenta-se o diagrama que o understand fornece com a arquitetura do software de terceiro, como já mencionado anteriormente esse gráfico torna a visualização um pouco mais ampla e mais fácil visualmente para entender os relacionamentos e métricas de coesão e acoplamento no código, dessa forma podemos realizar uma comparação a nível de métricas entre os dois sistemas.

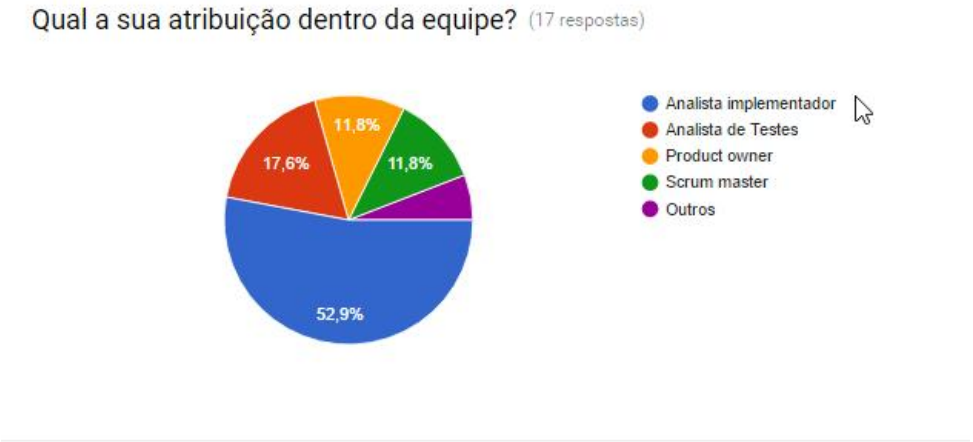
5 RESULTADOS E DISCUSSÕES

Nesta seção foi aplicado um questionário em relação a métricas de software para entender o que implementadores e testadores acham de possuir uma forma de ter a visibilidade do código através de métricas, e em quanto isso pode ajudar em seu trabalho ao realizar análise de impacto e estimar as tarefas de desenvolvimento e teste de software, o questionário está disposto no Apêndice A desse trabalho.

Foram selecionadas pessoas da área de tecnologia da informação, num total de 20 pessoas, que se disponibilizaram voluntariamente a responder o questionário. Foram consideradas as respostas de todo os integrantes das equipes ágeis formadas exclusivamente por analistas implementadores, analistas de testes, PO's e Scrum master das equipes.

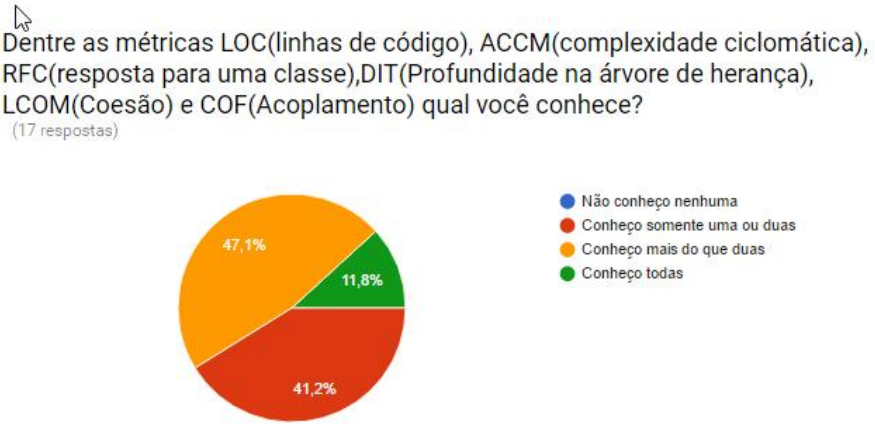
Com um total de 17 respostas, algumas pessoas possivelmente não responderam o questionário pelo fato de não conhecer o assunto e se sentiram inseguras em opinar e dessa forma teve uma abstenção de 3 votos no total, nas próximas páginas seguem os resultados de cada pergunta realizada que teve como objetivo analisar a eficiência das ferramentas Sonar e Understand/Structure.

Gráfico 1 – Gráfico de respostas da questão 1



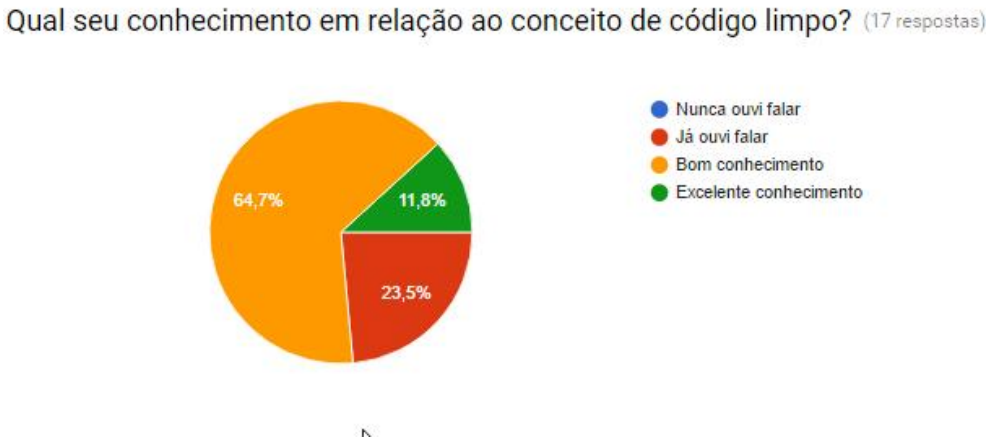
A pergunta 1 teve como objetivo descobrir o percentual de entrevistados com base em suas atribuições, isso é importante para saber de fato quem está respondendo a pesquisa já que o trabalho de análise de métricas revela a importância da participação de todos.

Gráfico 2 – Gráfico das respostas da questão 2



A pergunta 2 teve como objetivo nivelar o conhecimento dos entrevistados com base no que conheciam do assunto a ser perguntado, isso é importante para saber que de fato quem estava respondendo a pesquisa sabia do assunto, e dessa forma o resultado mostrou que a maioria conhecia as principais métricas do mercado.

Gráfico 3 – Gráfico das respostas da questão 3



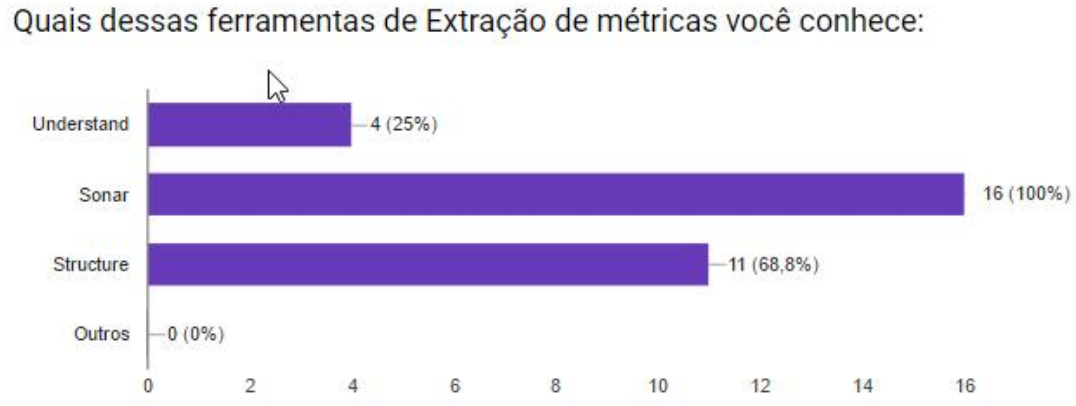
A pergunta 3 tinha o objetivo de afunilar ainda mais o assunto em relação ao conteúdo métrica de código fonte, essas perguntas além de contextualizar cada vez mais o entrevistado sobre o que o questionário deseja perguntar, mostrou através desse gráfico que

todos responderam tinham um bom conhecimento em relação ao tema, porém grande parte ainda desconhecia o assunto.

Gráfico 4– Gráfico das respostas da questão 4

A pergunta 4 teve como propósito igual ao da pergunta anterior que foi o de nivelar ainda mais o conhecimento dos entrevistados com base no que conheciam do assunto a ser perguntado e assim leva-los para uma discussão cada vez mais aprofundada em relação ao assunto.

Gráfico 5 – Gráfico das respostas da questão 5

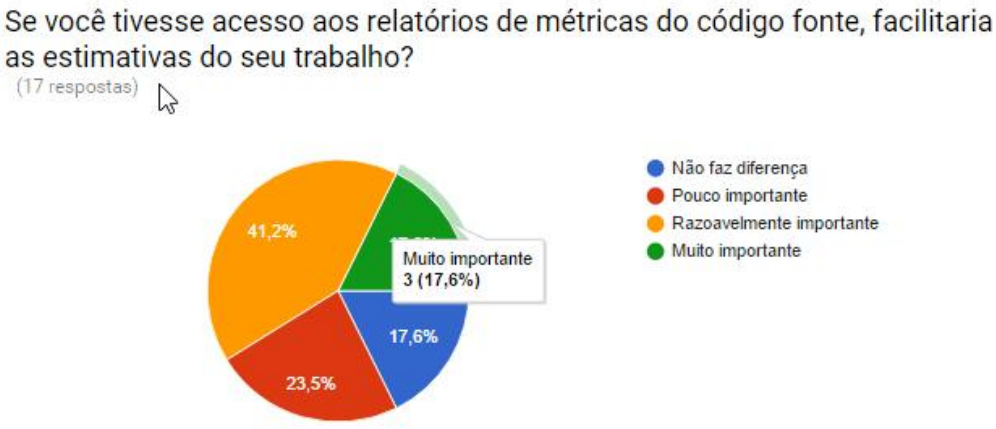


A pergunta 5 questionou o conhecimento dos entrevistados nas ferramentas que eram o cerne desse trabalho e como se pode visualizar no resultado do Gráfico 5 percebe-se que todos que responderam conheciam de fato as ferramentas apresentadas.

Gráfico 6 – Gráfico das respostas da questão 6

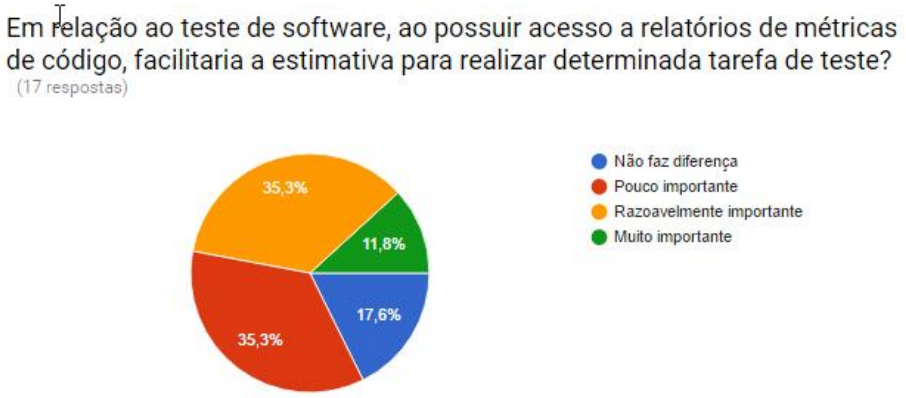
A pergunta 6 teve como objetivo questionar os entrevistados da importância em relação a ter uma ferramenta que venha revelar métricas no código fonte, e como os entrevistados pertenciam a equipes diferentes pode perceber que mais da metade dos entrevistados não tinham ainda nas suas equipes uma ferramenta de extração de métricas de código fonte.

Gráfico 7 – Gráfico das respostas da questão 7



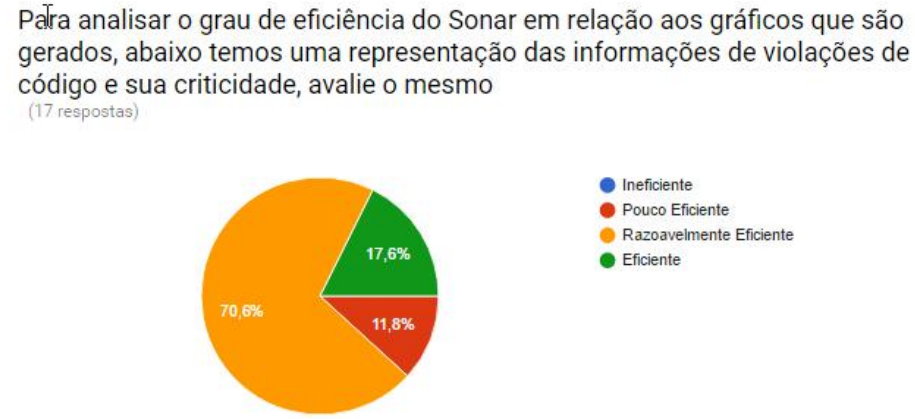
A pergunta 7 teve como objetivo questionar os entrevistados da importância em relação a ter acesso a relatórios de métricas de código fonte com intuito de melhorar as estimativas de seu trabalho, já que esse é um dos propósitos que se têm em extrair essas métricas houve um percentual 41,2 apontando a importância razoável de se possuir ferramentas como essas para analisar a saúde do código.

Gráfico 8 – Gráfico das respostas da questão 8



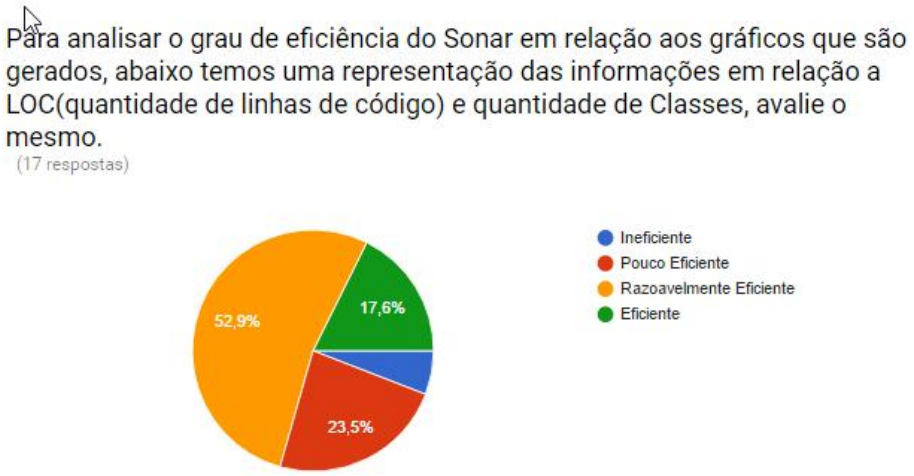
A pergunta 8 questionou os entrevistados da importância em relação as atividades de teste de software com o intuito de descobrir se com acesso aos relatórios de métricas de código tornaria mais fácil estimar as devidas tarefas em relação ao teste de cada funcionalidade, revelou-se pelo resultado do gráfico 8 um balanço entre razoavelmente importante ou pouco importante.

Gráfico 9 – Gráfico das respostas da questão 9



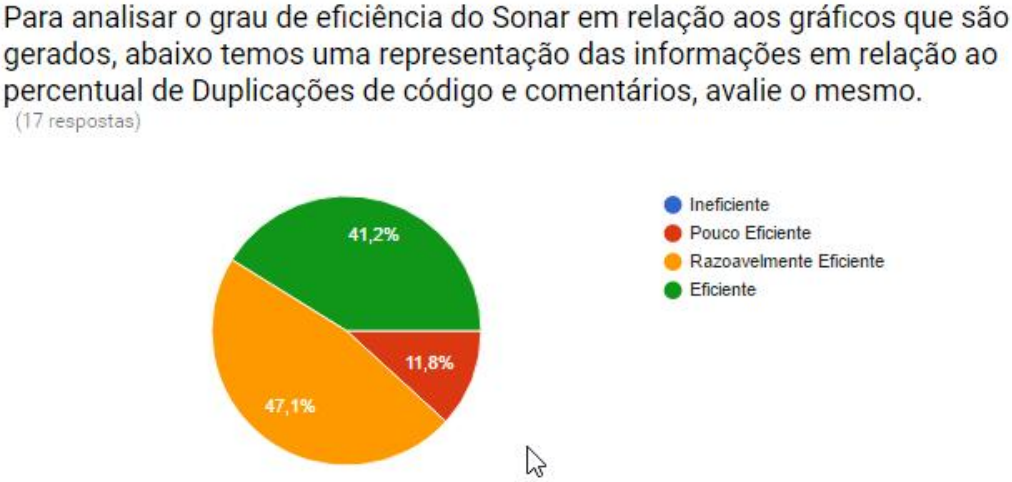
A pergunta 9 questionou os entrevistados em relação a eficiência da representação das informações de violação de código e sua criticidade que são gerados pelo Sonar em sua interface visual, mostrou-se que dentre todos os entrevistados mais de 70% sinalizam positivamente a esse quesito.

Gráfico 10 – Gráfico das respostas da questão 10



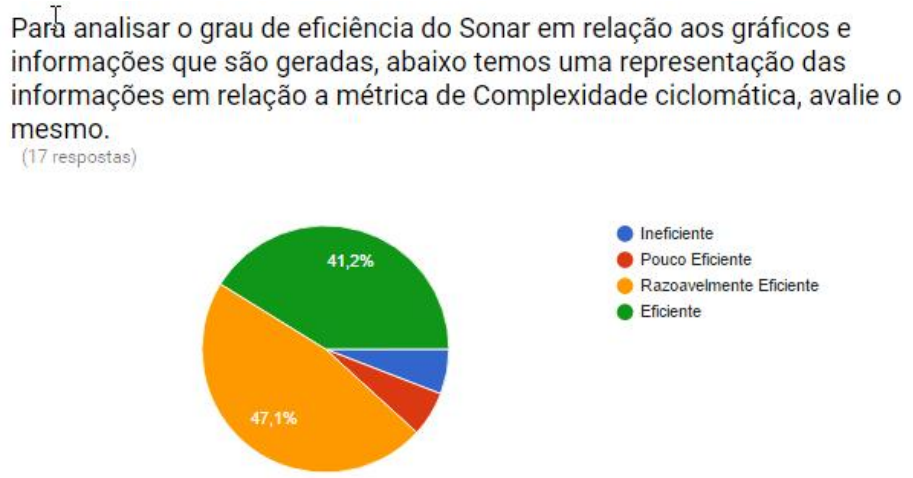
A pergunta 10 questionou os entrevistados em relação a eficiência da representação das informações da métrica LOC (quantidade de linhas de código) e Quantidade de Classes que são gerados pelo Sonar em sua interface visual, mostrou-se que dentre todos os entrevistados mais de 50% sinalizam positivamente a esse quesito.

Gráfico 11– Gráfico das respostas da questão 11



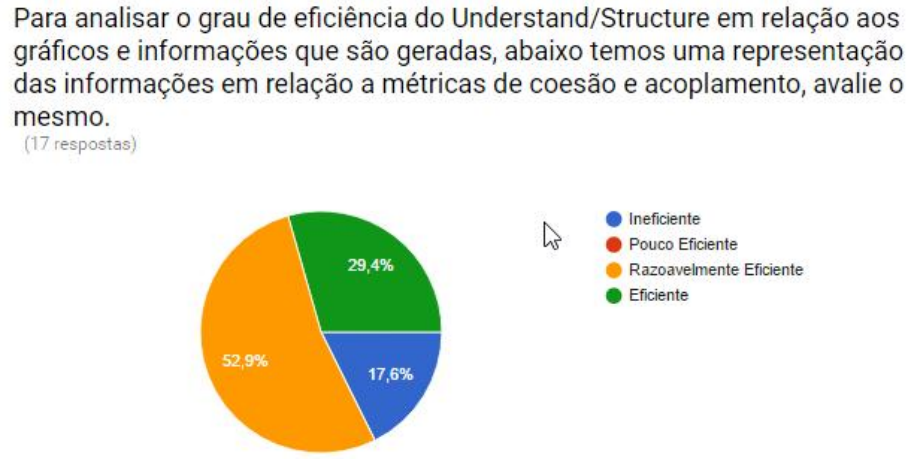
A pergunta 11 questionou os entrevistados em relação a eficiência da representação das informações da métrica de Duplicações de código e comentários que são gerados pelo Sonar em sua interface visual, mostrou-se que dentre todos os entrevistados 47,1% sinalizam razoavelmente eficiente a esse quesito e 41,2% sinalizam eficiente.

Gráfico 12 – Gráfico das respostas da questão 12



A pergunta 12 questionou os entrevistados em relação a eficiência da representação das informações da métrica de Complexidade Ciclomática que é gerada pelo Sonar em sua interface visual, mostrou-se que dentre todos os entrevistados 47,1% sinalizam razoavelmente eficiente a esse quesito e 41,2% sinalizam eficiente.

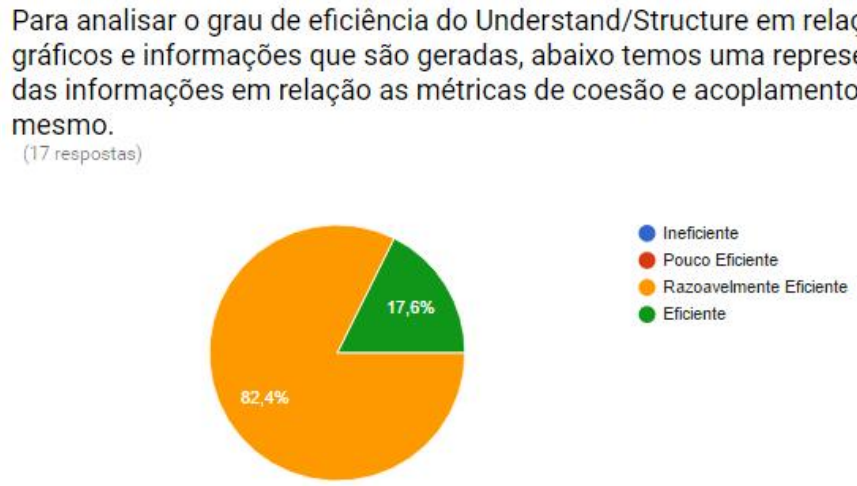
Gráfico 13 – Gráfico das respostas da questão 13



A pergunta 13 questionou os entrevistados em relação a eficiência da representação das informações da métrica de coesão e acoplamento que é gerada pelo Understand em forma

de matriz de dependência em sua interface visual, mostrou-se que dentre todos os entrevistados 52,9% sinalizam razoavelmente eficiente a esse quesito e 29,4% sinalizam eficiente.

Gráfico 14 – Gráfico das respostas da questão 14



A pergunta 14 questionou os entrevistados em relação a eficiência da representação das informações da métrica de coesão e acoplamento que é gerada pelo Understand através de uma visão mais arquitetural, mostrou-se que dentre todos os entrevistados 82,4% sinalizam de forma que o mesmo aplica eficiência razoável a esse quesito e 17,4% e não houve sinalização negativa.

6 CONSIDERAÇÕES FINAIS E TRABALHOS FUTUROS

Este trabalho realizou uma pesquisa que teve por objetivo principal colocar em prática toda a coleta de informações baseada no estudo bibliográfico, também foi possível concretizar a avaliação de ferramentas de coleta de métricas de código fonte como Understand/structure e Sonar e com isso possibilitou identificar o quanto o trabalho de medição é valoroso e como pode agregar valor ao processo de desenvolvimento de software.

Em um mercado de TI totalmente competitivo é imprescindível que as empresas tenham a preocupação com a qualidade do desenvolvimento de seus softwares prezando pela preocupação com o usuário final, e dessa forma essa mesma preocupação deve começar em casa ou seja em seu próprio ambiente organizacional aplicando conceitos de código limpo.

As ferramentas mencionadas nesse trabalho são uma sugestão de como as mesmas podem ser utilizadas para analisar o código fonte de empresas que possuem desenvolvimento com suporte à linguagem Delphi, onde as mesmas mostraram onde e como atacar as métricas mencionadas e assim dar uma visão geral do código fonte com uma interface visual e de certa forma prática.

Para cada empresa há uma realidade, sendo assim deve-se usar essa pesquisa como uma forma de norte para onde seguir quando se desejar utilizar ferramentas como essas para realizar a medição de código fonte, as métricas e ferramentas aqui mencionadas não podem ser consideradas como as melhores ou corretas mas sim formam um conjunto de sugestões para quem deseja realizar maiores estudo ou até mesmo aplicar um processo de medição de software dentro da empresa.

Espera-se que com o projeto definido neste trabalho, possa haver uma avaliação e a possibilidade de melhorias sobre o emprego das ferramentas Sonar e Understand/Structure como forma de medir a qualidade do código fonte com o intuito de monitorar e agregar valor ao código fonte, contribuindo assim para a aplicação de métodos que preconizam o desenvolvimento de um código limpo.

Na construção deste trabalho houve uma pequena dificuldade na definição das métricas a serem estudadas pelo fato da empresa utilizar uma linguagem de programação de certa forma defasada no mercado e também por isso as ferramentas escolhidas não suportam determinadas métricas, por isso foram escolhidas algumas entre as principais e optou-se por trabalhar com essa quantidade limitada.

Contudo o resultado foi atingido com base no que foi estudado e levantado durante a bibliografia que foi a de demonstrar ferramentas que podem ser utilizadas a fim de utilizar

métricas para medir a qualidade de código fonte, e mostrar o resultado das extrações de métricas determinadas que puderam ser escolhidas para análise.

Ao refletir sobre a montagem dos questionários pode-se verificar que o mesmo preconizou em enquadrar não somente desenvolvedores na resolução das mesmas, mas sim todos os envolvidos nas equipes e que estão diretamente ligados ao resultado final da qualidade do produto, ou seja, desenvolvedores, analista de testes, product owner's e scrum master's já que a empresa utiliza a metodologia ágil em seu ciclo de desenvolvimento.

Nas perguntas de 1 a 8 que vão desde o conhecimento direto as ferramentas Understand e Sonar, conhecimento sobre as métricas de código, conceitos de código limpo e o quanto o aproveitamento das informações geradas pelas ferramentas podem ser úteis na realização do trabalho da equipe, serviram justamente para entender se todos sabiam do que estava sendo falado. Ao final dos resultados entre alguns dos gráficos apresentados pode-se entender que isso não é um assunto somente direcionado a equipe de desenvolvimento pois todos responderam de forma positiva sabendo do assunto e assumiram pelo resultado o conhecimento e a importância em ter essa metodologia aplicada para auxiliar a melhorar cada o seu trabalho e consequentemente o produto final.

Refletindo mais afundo em relação a eficiência do Sonar e Understand entre todos os entrevistados mais especificamente as seis últimas perguntas do questionário de 9 a 14, revelaram que todos os entrevistados conheciam as ferramentas, conheciam as métricas e que mais de 50% acenaram positivamente com relação aos gráficos gerados pelo Sonar e Understand.

A aplicação do questionário foi importante também para atender um dos objetivos específicos desse trabalho e consistiu em revelar que as ferramentas apresentadas se mostraram com uma boa opção, na extração de métricas de código fonte para sistemas de grande porte como é o caso do PGE.net, apesar de existirem outras ferramentas no mercado, com base nas respostas dos entrevistados grande parte tem aprovado a forma com que as informações são geradas pelo sonar e understand.

Com base no trabalho desenvolvido, diversas ramificações podem ser vislumbradas afim de continuar possíveis trabalhos futuros, onde um deles é o fato de utilizar o Sonar afim de monitorar a construção de Build's com pipeline que pode ser aplicado em diversos times, nesse mesmo contexto podemos aplicar a utilização do utilitário chamado SRCCCHECK onde pode-se produzir gráficos ainda mais elaborados como gráficos do tipo histogramas, e também gráficos do tipo Kiviat que pode demonstrar os valores atuais e os valores máximos das métricas e correlacionar uma métrica a outra.

REFERÊNCIAS

ABRAN, Alain. Full function point measurement manual: V2.0. Canadá: Software Engineering Laboratory in Applied Metrics - Universidade de Quebec, 1999.

ALJADAA, Ali. Software Metrics. Disponível em <https://www.researchgate.net/publication/272830375_Software_Metrics/>. Acesso em: 27 nov. 2016.

ANICHE, Maurício. Como medir a coesão do seu código? A métrica LCOM. Disponível em: < http://blog.caelum.com.br/>. Acesso em: 14 jul. 2016.

ARAÚJO, Igor. Como medir com eficiência a qualidade de software na sua empresa. Disponível em: <http://blog.myscrumhalf.com/2013/06/sonar-apoiando-a-qualidade-do-desenvolvimento-de-software///>. Acesso em: 10 jan. 2017.

BARTIÉ, Alexandre. **Garantia da Qualidade de Software.** Rio de janeiro: Campus, 2002.

BONEL, Cláudio. Afinal, o que é Business Intelligence: Micros, pequenas, médias e grandes Empresas com o poder da informação nas mãos. Disponível em: <https://books.google.com.br/books?id=eS6xCQAAQBAJ&printsec=frontcover&hl=pt-BR#v=onepage&q&f=false/>. Acesso em: 06 nov. 2016.

CADU. Entendendo Coesão e Acoplamento. Disponível em: < http://www.devmedia.com.br/entendendo-coesao-e-acoplamento/18538/> . Acesso em: 02 dez. 2016.

CHARNEY, Reg. Programming Tools: Code Complexity Metrics. Disponível em: <http://www.linuxjournal.com/article/8035/>. Acesso em: 26 nov. 2016.

CHIDAMBER, Shyam R; KEREMER, Chris F. A Metrics suite for object oriented desing. IEEE – Transactions on software enginieering. São Paulo, p. 476-493, vol. 20, 1994.

CORDEIRO, Marco Aurélio. Métricas de Software. Disponível em: <http://www.batebyte.pr.gov.br/modules/conteudo/conteudo.php?conteudo=88/>. Acesso em: 14 out. 2016.

DANIEL, Leandro. Code metrics (parte 1) – Métricas de código são aliadas do arquiteto. Disponível em: <http://leandrodaniel.com/index.php/code-metrics-parte-1-metricas-de-codigo-sao-aliadas-do-arquiteto//>. Acesso em: 14 jul. 2016.

DANIEL, Leandro. Code metrics (parte 2) – Conhecendo algumas métricas. Disponível em: < http://leandrodaniel.com/index.php/code-metrics-parte-2-conhecendo-algumas-metricas//>. Acesso em: 26 nov. 2016.

FILHO, Carlos Morais de Oliveira. *Kalibro: interpretação de métricas de código-fonte*. 2013. Dissertação (Mestrado em Ciências) – Instituto de matemática e estatística, Universidade de São Paulo, São Paulo. 2013.

HIRAMA, Kechi. **Engenharia de Software:** Qualidade e Produtividade com Tecnologia. 1. ed. Rio de Janeiro: Elsevier, 2012.

JEZ, Humble. Farley, David. **Entrega Contínua:** Como Entregar Software de Forma Rápida e Confiável. 2. ed. Porto Alegre: BOOKMAN Editora, 2014.

KAN, S. H. Metrics and models in software quality engineering. 2. ed. Minnesota: Addison-Wesley, 2002.

STRUCTURE101KLOCWORK. Disponível em: < http://www.klocwork.com/products-services/code-architecture/structure101http://structure101.com/> . Acesso em: 13 12 jandez. 20176.

KOSCIANSKI, André. SOARES, Michel dos Santos. **Qualidade de Software:** Aprenda as metodologias e técnicas mais modernas para o desenvolvimento de software. 2. ed. São Paulo: NOVATEC, 2007.

LOBO, Adilton. Princípios da Qualidade Total Aplicados a Softwares. Disponível em: < http://pages.udesc.br/~r4al/qt.htm/>. Acesso em: 03 nov. 2016.

LOBO, Adilton. *Princípios da Qualidade Total Aplicados a Softwares – Processo e Produto*. 2013. Universidade Federal do Rio Grande do Sul/Universidade do Estado de Santa Catarina, Florianópolis. 1997.

MARTINS, José Carlos Cordeiro Martins. **Técnicas para Gerenciamento de projetos de Software**. 1. ed. Rio de Janeiro Alegre: BRASPORT, 2007.

MAXIM, Bruce. PRESSMAN, Roger. **Engenharia de Software:** uma abordagem Profissional**.** 8ª ed. Porto Alegre: AMGH, 2016.

MILLANI, Luís Felipe Garlet. *Análise de correlação entre métricas de Qualidade de Software e métricas físicas*. 2013. Universidade Federal do Rio Grande do Sul, Porto Alegre. 2013.

MORAIS, Lenildo. Qualidade de Software. **Engenharia de Software Magazine:** Desvendando um requisito essencial no processo de desenvolvimento, Rio de Janeiro, v. 29, n. 29, p. 34-38. Ano 3. Disponível em: <http://arquivo.devmedia.com.br/Diagramacao/Revistas_PDF/ES/ENGENHARIA-SOFTWARE_029_ONHJCNUC.pdf>. Acesso em: 01 nov. 2016

NBR ISO/IEC 9126-1: Engenharia de software - Qualidade de produto. <http://www.abnt.org.br/>. Acesso em: 14 out. 2016.

NOVAIS, Gabriel. Sonar Java: Avaliando o código através de métricas: Disponível em: < http://www.devmedia.com.br/sonar-java-avaliando-o-codigo-atraves-de-metricas/31278/>. Acesso em 02 dez. 2016

OAKLAND, John. **Gerenciamento da qualidade total.** NBL Editora, 1994.

PRESSMAN, R. S. **Engenharia de Software**. 8. ed. Rio de Janeiro: McGraw-Hill, 2002.

PRESSMAN, Roger. **Engenharia de Software:** uma abordagem Profissional. 7. ed. Porto Alegre: AMGH, 2011.

PRESSMAN, Roger. **Engenharia de Software:** uma abordagem Profissional. 6. ed. Porto Alegre: AMGH, 2011.

REISSWITZ, Flavia. Análise de Sistemas V7. Disponível em: < https://books.google.com.br/books?id=Z-ZEBQAAQBAJ&printsec=frontcover&hl=pt-BR#v=onepage&q&f=false/>. Acesso em: 06 nov. 2016.

REZENDE, Denis Alcides. **Engenharia de Software e Sistemas de informação**. 3. ed. Rio de Janeiro: Brasport, 2005.

ROSENBERG, Linda. Applying and interpreting object oriented metrics. Utah, abr. 1998. Disponível em: < http://www.literateprogramming.com/ooapply.pdf/> . Acesso em: 26 nov. 2016.

SCITOOLS. Disponível em: < https://scitools.com/static-analysis-toolhttps://scitools.com/> . Acesso em: 13 12 jandez. 20176.

SOFTPLAN. Disponível em: < https://intranet.sofplan.com.br/> . Acesso em: 10 dez. 2016.

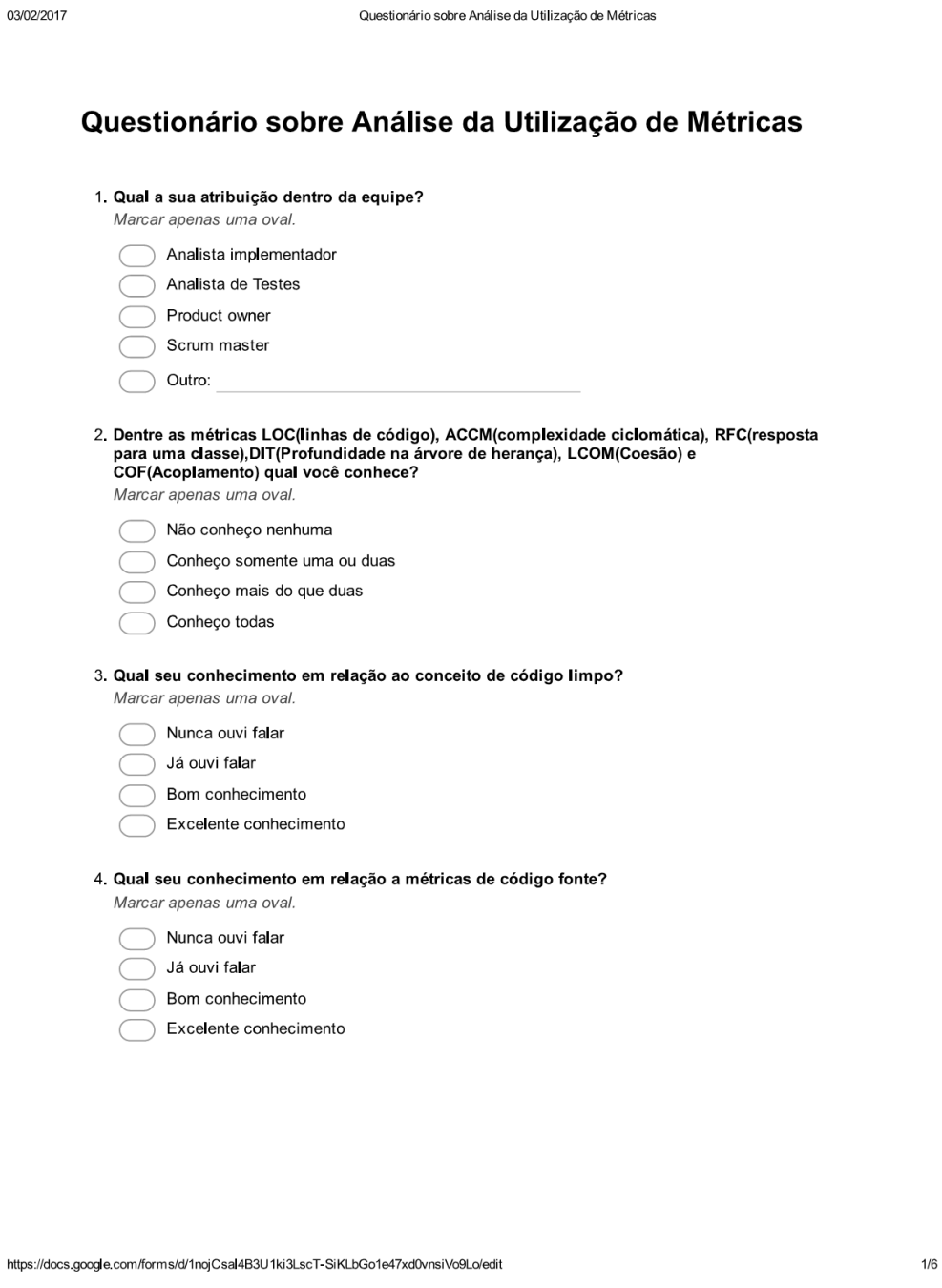
SOMMERVILLE, I. **Engenharia de Software**. 6ª Ed. São Paulo: Addison Wesley, 2003.

SONAR SOFTPLAN. Disponível em: < http://jenkins//> . Acesso em: 10 dez. 2016.

SUBRAMANYAN, Jitendra. Como medir com eficiência a qualidade de software na sua empresa. Disponível em: < http://computerworld.com.br/tecnologia/2014/05/13/como-medir-com-eficiencia-a-qualidade-de-software-na-sua-empresa//>. Acesso em: 10 jan. 2017.

APÊNDICE A

Figura 18 – Página 1 do questionário



03/02/2017 Questionário sobre Análise da Utilização de Métricas

Questionário sobre Análise da Utilização de Métricas

1. **Qual a sua atribuição dentro da equipe?**
   Marcar apenas uma oval.
   - ◯ Analista implementador
   - ◯ Analista de Testes
   - ◯ Product owner
   - ◯ Scrum master
   - ◯ Outro: _____

2. **Dentre as métricas LOC(linhas de código), ACCM(complexidade ciclomática), RFC(resposta para uma classe),DIT(Profundidade na árvore de herança), LCOM(Coesão) e COF(Acoplamento) qual você conhece?**
   Marcar apenas uma oval.
   - ◯ Não conheço nenhuma
   - ◯ Conheço somente uma ou duas
   - ◯ Conheço mais do que duas
   - ◯ Conheço todas

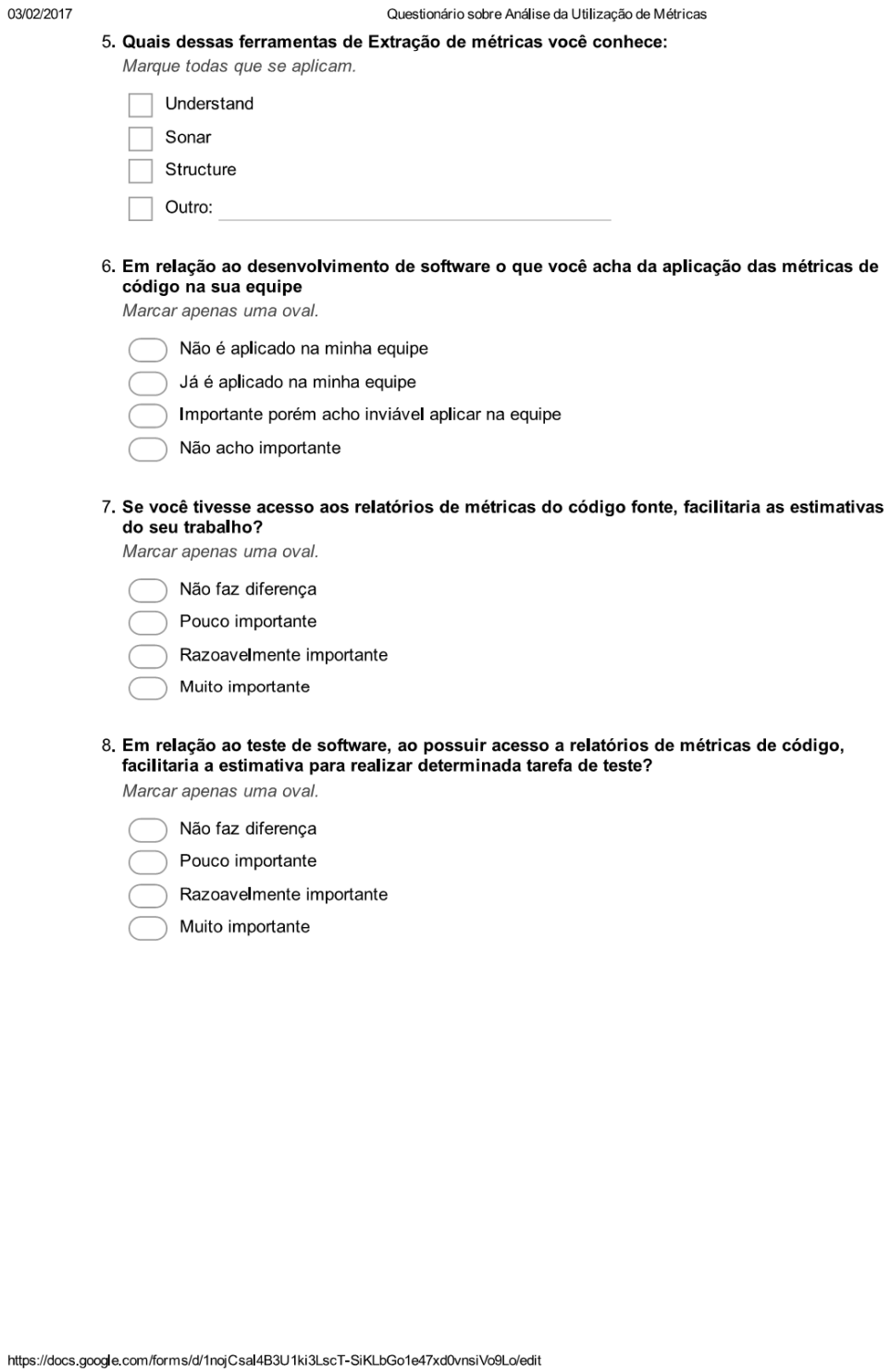
3. **Qual seu conhecimento em relação ao conceito de código limpo?**
   Marcar apenas uma oval.
   - ◯ Nunca ouvi falar
   - ◯ Já ouvi falar
   - ◯ Bom conhecimento
   - ◯ Excelente conhecimento

4. **Qual seu conhecimento em relação a métricas de código fonte?**
   Marcar apenas uma oval.
   - ◯ Nunca ouvi falar
   - ◯ Já ouvi falar
   - ◯ Bom conhecimento
   - ◯ Excelente conhecimento

https://docs.google.com/forms/d/1nojCsaI4B3U1ki3LscT-SiKLbGo1e47xd0vnsiVo9Lo/edit 1/6

Fonte: Elaborado pelo autor (2017)

Figura 19 – Página 2 do questionário



03/02/2017 Questionário sobre Análise da Utilização de Métricas

5. **Quais dessas ferramentas de Extração de métricas você conhece:**
Marque todas que se aplicam.

- [ ] Understand
- [ ] Sonar
- [ ] Structure
- [ ] Outro: _____

6. **Em relação ao desenvolvimento de software o que você acha da aplicação das métricas de código na sua equipe**
Marcar apenas uma oval.

- ◯ Não é aplicado na minha equipe
- ◯ Já é aplicado na minha equipe
- ◯ Importante porém acho inviável aplicar na equipe
- ◯ Não acho importante

7. **Se você tivesse acesso aos relatórios de métricas do código fonte, facilitaria as estimativas do seu trabalho?**
Marcar apenas uma oval.

- ◯ Não faz diferença
- ◯ Pouco importante
- ◯ Razoavelmente importante
- ◯ Muito importante

8. **Em relação ao teste de software, ao possuir acesso a relatórios de métricas de código, facilitaria a estimativa para realizar determinada tarefa de teste?**
Marcar apenas uma oval.

- ◯ Não faz diferença
- ◯ Pouco importante
- ◯ Razoavelmente importante
- ◯ Muito importante

https://docs.google.com/forms/d/1nojCsaI4B3U1ki3LscT-SiKLbGo1e47xd0vnsiVo9Lo/edit 2/6

Fonte: Elaborado pelo autor (2017)

Figura 20 – Página 3 do questionário

Questionário sobre Análise da Utilização de Métricas

9. **Para analisar o grau de eficiência do Sonar em relação aos gráficos que são gerados, abaixo temos uma representação das informações de violações de código e sua criticidade, avalie o mesmo**

Marcar apenas uma oval.

- Ineficiente
- Pouco Eficiente
- Razoavelmente Eficiente
- Eficiente

10. **Para analisar o grau de eficiência do Sonar em relação aos gráficos que são gerados, abaixo temos uma representação das informações em relação a LOC(quantidade de linhas de código) e quantidade de Classes, avalie o mesmo.**

Marcar apenas uma oval.

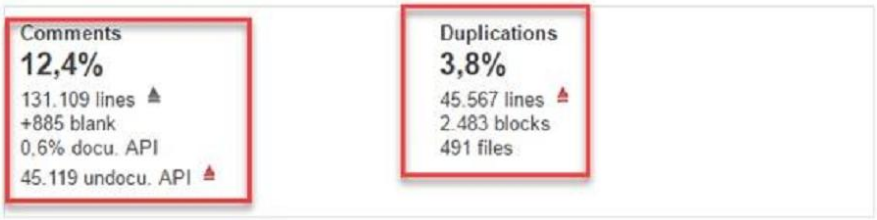
- Ineficiente
- Pouco Eficiente
- Razoavelmente Eficiente
- Eficiente

Fonte: Elaborado pelo autor (2017)

Figura 21 – Página 4 do questionário

Questionário sobre Análise da Utilização de Métricas

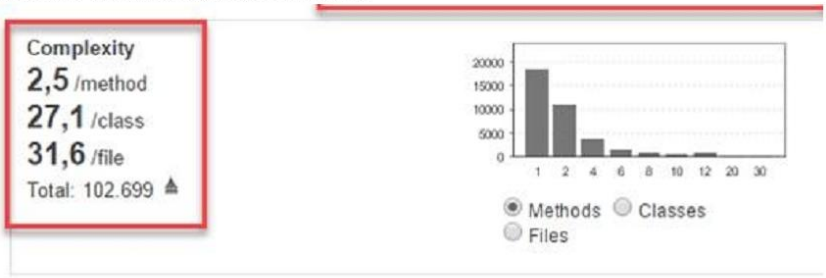
11. **Para analisar o grau de eficiência do Sonar em relação aos gráficos que são gerados, abaixo temos uma representação das informações em relação ao percentual de Duplicações de código e comentários, avalie o mesmo.**



Marcar apenas uma oval.

- ◯ Ineficiente
- ◯ Pouco Eficiente
- ◯ Razoavelmente Eficiente
- ◯ Eficiente

12. **Para analisar o grau de eficiência do Sonar em relação aos gráficos e informações que são geradas, abaixo temos uma representação das informações em relação a métrica de Complexidade ciclomática, avalie o mesmo.**



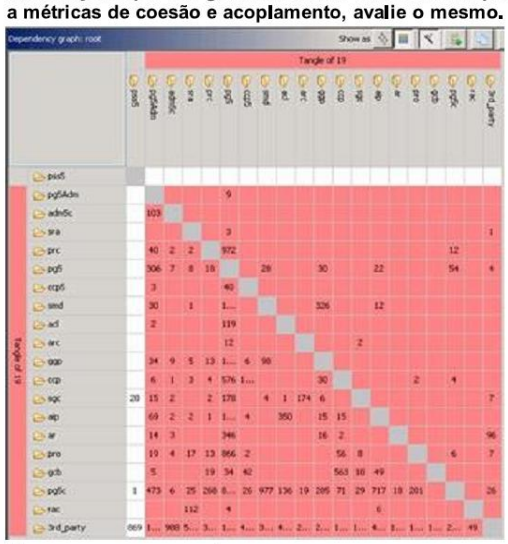
Marcar apenas uma oval.

- ◯ Ineficiente
- ◯ Pouco Eficiente
- ◯ Razoavelmente Eficiente
- ◯ Eficiente

Figura 22 – Página 5 do questionário

Questionário sobre Análise da Utilização de Métricas

13. **Para analisar o grau de eficiência do Understand/Structure em relação aos gráficos e informações que são geradas, abaixo temos uma representação das informações em relação a métricas de coesão e acoplamento, avalie o mesmo.**



Marcar apenas uma oval.

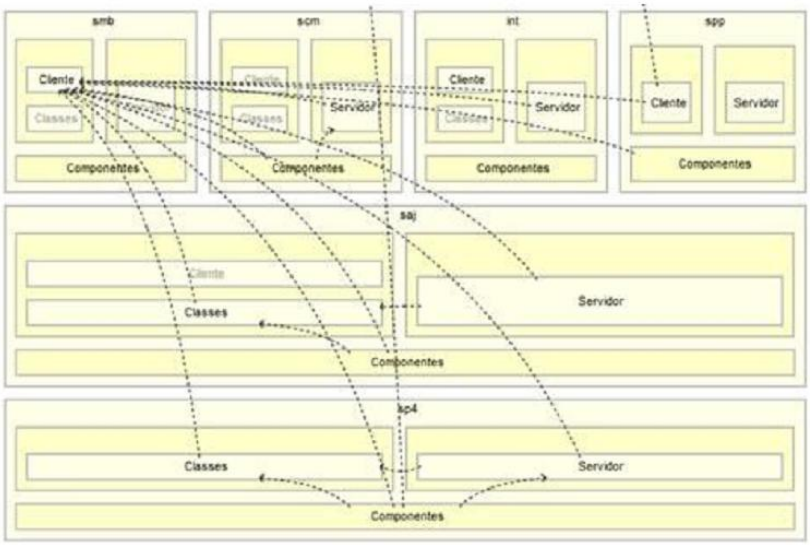
- Ineficiente
- Pouco Eficiente
- Razoavelmente Eficiente
- Eficiente

Elaborado pelo autor (2017)

Figura 23 – Página 6 do questionário de análise de métricas

Questionário sobre Análise da Utilização de Métricas

14. **Para analisar o grau de eficiência do Understand/Structure em relação aos gráficos e informações que são geradas, abaixo temos uma representação das informações em relação as métricas de coesão e acoplamento, avalie o mesmo.**



Marcar apenas uma oval.

- ◯ Ineficiente
- ◯ Pouco Eficiente
- ◯ Razoavelmente Eficiente
- ◯ Eficiente

Powered by
Google Forms

Elaborado pelo autor (2017)

APÊNDICE B

AUTORIZAÇÃO

SOFTPLAN PLANEJAMENTO E SISTEMAS LTDA. inscrita no CNPJ sob o nº 82.845.322/0001-04, com sede à Av. Luiz Boiteux Piazza – nº 1302 – Sapiens Parque S.A – Fone (48) 3027-8000 – Fax (48) 3027-8008 – CEP 88056-000 – Florianópolis – SC, por seu representante legal infrafirmado, **AUTORIZA** Fernando Antunes Rodrigues a divulgar em seu Trabalho de Conclusão de Curso (TCC), intitulado USO DE INDICADORES PARA MEDIR A QUALIDADE DO CÓDIGO ATRAVÉS DAS FERRAMENTAS SONAR E UNDERSTAND/STRUCTURE, no curso de pós-graduação em Engenharia de Projetos de Software, da Universidade do Sul de Santa Catarina - UNISUL, o nome da empresa, informações sobre o fluxo de trabalho coletadas dos profissionais da empresa, diagramas e detalhes do processo de desenvolvimento da equipe **SAJ/Procuradorias**.

Florianópolis, 17 de fevereiro de 2017

SOFTPLAN PLANEJAMENTO E SISTEMAS LTDA
Roger Pedroso
CPF: 774.457.839-20