Pós-Graduação em Ciência da Computação

ANDERSON FONSECA E SILVA

UM MÉTODO BASEADO EM MÉTRICAS PARA PREDIÇÃO DE IMPACTO EM ATIVIDADES DE CODIFICAÇÃO EM PROJETOS DE SOFTWARE

Universidade Federal de Pernambuco
posgraduacao@cin.ufpe.br
http://cin.ufpe.br/~posgraduacao

Recife
2018

ANDERSON FONSECA E SILVA

UM MÉTODO BASEADO EM MÉTRICAS PARA PREDIÇÃO DE IMPACTO EM ATIVIDADES DE CODIFICAÇÃO EM PROJETOS DE SOFTWARE

Trabalho apresentado ao Programa de Pós-graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Doutor em Ciência da Computação.

Área de Concentração: Engenharia de Software
Orientador: Vinícius Cardoso Garcia

Recife
2018

Catalogação na fonte
Bibliotecária Monick Raquel Silvestre da S. Portes, CRB4-1217

S586m Silva, Anderson Fonseca e
Um método baseado em métricas para predição de impacto em atividades de codificação em projetos de software / Anderson Fonseca e Silva. – 2018.
152 f.: il., fig., tab.

Orientador: Vinícius Cardoso Garcia.
Tese (Doutorado) – Universidade Federal de Pernambuco. CIn, Ciência da Computação, Recife, 2018.
Inclui referências e apêndices.

1. Engenharia de software. 2. Manutenção de software. I. Garcia, Vinícius Cardoso (orientador). II. Título.

005.1 CDD (23. ed.) UFPE- MEI 2019-024

Tese de Doutorado apresentada por **ANDERSON FONSECA E SILVA** à Pós-Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco, sob o título "**UM MÉTODO BASEADO EM MÉTRICAS PARA PREDIÇÃO DE IMPACTO EM ATIVIDADES DE CODIFICAÇÃO EM PROJETOS DE SOFTWARE**" **Orientador: Vinícius Cardoso Garcia** e aprovada pela Banca Examinadora formada pelos professores:

Prof. Lincoln Souza Rocha
Departamento de Computação/UFC

Prof. Sergio Castelo Branco Soares
Centro de Informática/UFPE

Prof. Vanilson Andre de Arruda Buregio
Departamento de Estatística e Informática/UFRPE

Prof. Leandro Marques do Nascimento
Departamento de Estatística e Informática/UFRPE

Prof. Daniel Lucredio
Departamento de computação/UFSCar

Prof. **Orientador**: Vinícius Cardoso Garcia
Centro de Informática / UFPE

Visto e permitida a impressão.
Recife, 14 de setembro de 2018.

Prof. Aluizio Fausto Ribeiro Araújo
Coordenadora da Pós-Graduação em Ciência da Computação do
Centro de Informática da Universidade Federal de Pernambuco.

Eu dedico esta tese ao meu pai (in memorian), meu sogro (in memorian), a minha família e ao meu orientador que me deram o suporte necessário para o alcance do próximo degrau.

AGRADECIMENTOS

Agradecimentos especiais a todos que contribuiram diretamente para este trabalho. Primeiramente ao meu orientador, professor Vinicius Garcia, pela parceria, paciência, amizade e cuidado durante todas fases desse trabalho. Agradeço ao CIn e aos seus funcionários pela estrutura e formação de qualidade, que pude perceber durante todo o curso. Gostaria de agradecer ao pessoal do INES — Instituto Nacional de Ciência e Tecnologia para Engenharia de Software, do AssertLab e da Ustore, pelos direcionamentos, discussões e apoio. A Accenture, empresa na qual trabalho e que me deu subsídios para a concepção, elaboração, construção e aplicação deste trabalho, e em especial aos amigos Isaac Babsky e César Araújo. E por fim, a todos que indiretamente me apoiaram: minha esposa Sandra Fonseca, minha filha Sophia Fonseca pelo fato de sua existência me impulsionar na busca pela evolução contínua, familiares e amigos.

RESUMO

Ao longo dos anos, trabalhos no estado da arte de Engenharia de Software, vêm demonstrando que os custos relacionados à manutenção de software alcançaram aproximadamente 70% do valor total de construção dos sistemas. Deste modo, questões como avaliação de qualidade contínua, bem como, o custo envolvido em evoluções, se tornam fatores de suma importância para a tomada de decisão no que tange à continuidade de um determinado software, sua replataformização ou substituição. Observando tal contexto, o propósito deste trabalho está em entender como subsidiar tal tomada de decisão, direcionando e, esclarecendo implicações e custos associados, bem como, quais ações iniciais podem ser adotadas. Durante um período de 2 anos, este trabalho avaliou como a tomada de decisão impactava na qualidade dos projetos desenvolvidos por uma consultoria de TI com atuação global no desenvolvimento e manutenção de sistemas. De forma prática, foi selecionado um conjunto de projetos de software, segmentados em verticais da indústria, permitindo um levantamento sobre o estado atual de cada projeto com relação a métodos de construção, acompanhamento das atividades com avaliações periódicas, direcionamentos junto com lideranças e discussões.

Palavras-chaves: Engenharia de software. Manutenção e evolução de software. Retorno do investimento.

ABSTRACT

Over the years, works in the state of the art of Software Engineering have demonstrated that the diversity of technologies and the complexity in the construction and evolution of the software, causes costs related to software maintenance to reach approximately 70 % of the total value of Construction of systems. In this way, issues such as continuous quality assessment, as well as the cost involved in evolutions, become important factors for decision making regarding the continuity of a particular software, its replataformization or replacement. Looking at this context, the purpose of this work is to understand how to subsidize such decision making, directing and clarifying associated costs and implications, as well as what initial actions can be taken. In the light of the above scenario, this work began with the verification of how decision-making impacts on the approaches adopted to continuous quality improvement in projects in an IT consultancy with a global role in the development and maintenance of systems over a period of 2 years.

Key-words: Software engineering. Software maintenance and evolution. Return of investment.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

Diante do cenário econômico atual, empresas segmentadas em diversos mercados procuram automatizar seus processos e operações através do uso de software. Seja por meio do aumento em investimentos em novos projetos ou integrando sistemas legados, existe uma busca crescente à consultorias especializadas em Tecnologia da Informação (TI) para apoiar tais empresas em termos de execução e planejamento.

Através dos dados disponibilizados pela Associação Brasileira das Empresas de Software (ABES)[1], no período de 2004 (data do primeiro relatório publicado) até 2017[2] é possível verificar a evolução do mercado de software brasileiro, para as áreas de software e serviços com a seguinte segmentação:

- **Software** - aplicativos, ambientes de desenvolvimento, infraestrutura e segurança, produção local para exportação;
- **Serviços** - *outsourcing*, suporte, integração de sistemas, consultorias e planejamento, software sob encomenda, serviços para exportação, treinamento e software desenvolvido no exterior.

Conforme a Figura 1, é possível identificar que o setor de serviços que se iniciou com investimentos com pouco mais de US$ 3.620 milhões em 2004, atingiu um pico de US$ 17.510 milhões em 2012, e, se manteve em torno de US$ 10.000 milhões nos últimos dois anos em média. Observando o setor de software, também verificamos que partindo de investimentos por volta de US$ 2.360 milhões em 2004, atingiu um pico de US$ 12.300 milhões em 2015, e se manteve nos últimos dois anos com um volume de US$ 8.180 milhões. Em ambos os setores, foi possível constatar o crescimento do volume de investimentos ao longo do período.

Considerando o Brasil no cenário global, verificamos que houve uma evolução saindo da décima quinta posição em 2004, se estabelecendo na sétima posição durante o período entre 2012 a 2015, descendo para a nona posição nos últimos dois anos.

Observando a evolução na demanda à consultorias, integração de sistemas e outsourcing, considerando que estes dois últimos fazem parte do portifólio de serviços oferecidos por consultorias, vemos que os resultados conforme a Tabela 1, apresentaram um crescimento do volume de investimentos nos três segmentos, bem como, uma taxa de crescimento anual significativa no período compreendido entre 2004 a 2017.

Ainda conforme a ABES, atualmente o mercado interno total de TI conta com 15.707 empresas, sendo 11.237 delas dedicadas ao desenvolvimento e a comercialização de software e 4.470 dedicadas aos serviços de TI. Neste sentido, podemos verificar um crescimento

[1] http://www.abessoftware.com.br/ - acessado 09/2018
[2] http://www.abessoftware.com.br/dados-do-setor/anos-anteriores acessado em 09/2018

Figura 1 – Evolução do mercado brasileiro de software - ABES - (Elaborada pelo autor)

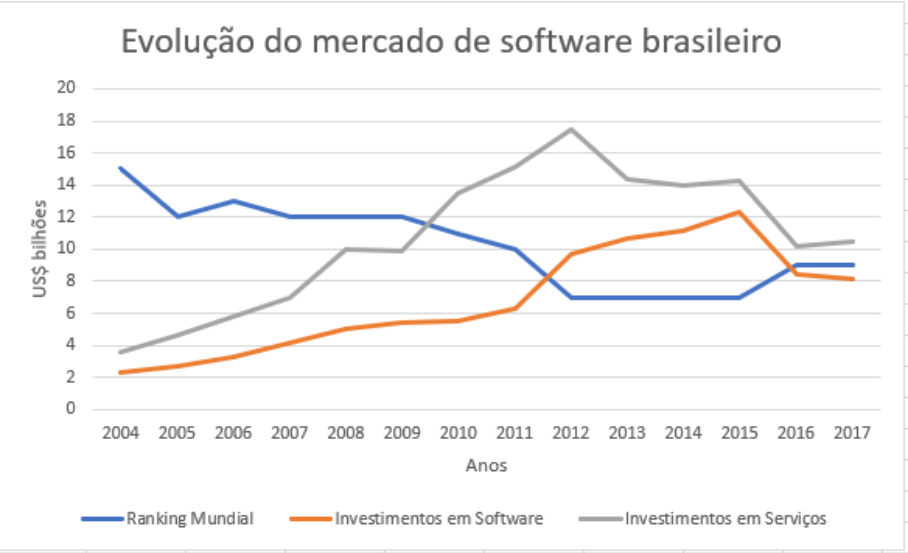


Tabela 1 – Comparativo evolução 2004-2017 - Em US$ milhões

Segmentos	Volume 2004	Volume 2017	Variação	Crescimento anual
Consultorias	470,5	1.108	135,49%	6,30%
Integração de sistemas	740,3	1.620	118,„83%	5,75%
Outsourcing	1.117,4	4.515	304%	10,48%

considerável, dado que, em 2004 o mercado de software e serviços era composto por 7.700 empresas.

Deste modo, podemos afirmar que conforme a avaliação dos dados apresentados sobre a evolução do mercado brasileiro, é visivel o aumento das empresas, demanda de serviços e o volume de investimentos no setor de TI.

Considerando o contexto de projetos de TI, foram investigados trabalhos e publicações em sites especializados de forma não-exaustiva e de acesso gratuíto, procurando entender o panorama geral sobre as falhas e sucessos. Neste sentido, foram selecionados estudos entre 2011 a 2015, comparando-os com o CHAOS report elaborado em 1995, conforme a Tabela 2. Verificando os resultados apresentados, é possível identificar uma certa estabilidade no percentual de sucessos e falhas, cabendo uma verificação se os fatores ainda permaneceriam os mesmos.

Para a verificação dos fatores de sucesso, selecionamos dentre os estudos na Tabela 2, mais o Extreme CHAOS Report de 2001, o dois fatores mais relevantes, conforme a Tabela 3. Neste sentido, verificamos que o suporte executivo persisitu com o passar dos anos afirmando a sua importância dentro de projetos de TI.

Outros pontos a serem observados são as habilidades dos times e a maturidade emocional, quando o envolvimento do usuário estava figurando entre as duas posições nos anos iniciais. Deste modo, é possível presumir que os direcionadores passaram a considerar a

Tabela 2 – Comparativo de sucessos e falhas em projetos de TI (2000-2015)

Referência	Ano	Sucesso (%)	Falha (%)
CHAOS Report[3]	1994	16%	31%
CHAOS Report[4]	2011	29%	22%
	2012	27%	17%
	2013	31%	19%
	2014	28%	17%
	2015	29%	19%
Wrike[5]	2015	39%	18%

qualidade dos times e da gestão como contribuintes para o sucesso dos projetos.

Tabela 3 – Fatores de sucesso em projetos de TI (2000-2015)

Referência	Ano	Sucesso
CHAOS Report	1995	Envolvimento do usuário
		Suporte executivo
Extrema CHAOS [6]	2001	Suporte executivo
		Envolvimento do usuário
CHAOS Report	2015	Suporte executivo
		Maturidade emocional
Wrike	2015	Habilidades técnicas dos times
		Suporte executivo

Associando a demanda crescente por consultorias e sua necessidade de modernização de aplicações legados, visto o crescimento de serviços de integração de sistemas, se torna importante entender a dinâmica de trabalho e os custos relacionados ao termo manutenção de sistemas.

Segundo a ISO (1995), podemos entender manutenção como: "um processo de modificação do código e documentação associada devido a um problema ou necessidade de melhoria". Uma concepção comum sobre a manutenção é a correção de defeitos, porém, verificando alguns estudos conforme a Tabela 4, ações não-corretivas como solicitação de melhorias são comuns em projetos em produção.

Segundo Barry, Kemerer e Slaughter (1999), o custo maior de esforço em manutenção está associado a ações não-corretivas, onde 78% das solicitações dos usuários se relacionam à melhorias. Para Pigoski (2001), o percentual do esforço em ações não-corretivas atinge 80%, neste sentido, o autor relatou que através da análise de relatórios submetidos por usuários solicitando melhorias, ficou constatado que se tratavam de evoluções, adicionando funcionalidades ao invés de correções.

Considerando a atuação de consultorias em serviços de integração de sistemas e/ou modernização de legado, existe a necessidade de entendimento sobre a situação dos proje-

Tabela 4 – Manutenção de software - percentuais de esforço em ações não-corretivas

Referência	Percentual
(SWANSON, 1976)	60,3%
(BARRY; KEMERER; SLAUGHTER, 1999)	78%
(GLASS, 2001)	60%
(PIGOSKI, 2001)	80%
Daniel Galorath (2008) [7]	75%

tos a serem absorvidos e mantidos em termos de qualidade. Neste contexto, considerando o suporte executivo na absorção de tais serviços, é interessante estabelecer um vocabulário comum permitindo o entendimento de todos os envolvidos no processo, possibilitando que ações e controles contínuos de predição de impactos direcionem a tomadas de decisão com segurança.

Deste modo, esta tese apresenta um método sistemático que parte da mineração em repositórios de software, seguindo um fluxo de atividades que vão desde a caracterização do projeto em termos de métricas à projeções de manutenção e custos, como suporte para a tomada de decisão de forma estratégica.

1.1 ESTABELECIMENTO DO PROBLEMA

Todos nós tomamos decisões diariamente, mesmo as mais simples, incluindo o que vestir, comer, se utilizar um ônibus ou ir andando. No contexto da Engenharia de Software, tomadas de decisão nos direcionam para a especificação, desenvolvimento, validação e evolução de software, como por exemplo, o que seria mais interessante: desenvolver uma funcionalidade ou adquirir um produto de terceiros, ou qual a metodologia mais adequada para o projeto e quanto tempo um determinado sistema deveria ser mantido?

Para Carrell, Jennings e Heavrin (1997), o termo decisão é descrito como "o processo fundamental nas organizações pelo qual gerentes escolhem uma alternativa dentre outras". Segundo Jones (2010), decisões podem ser classificadas em programadas e não-programadas, onde, programada é uma decisão rotineira podendo ser repetida, e, não-programada lida com a complexidade de situações não-estruturadas.

Neste sentido, temos a teoria de análise de decisão, definida por (KEENEY R. L., 1976.) como algo projetado para ajudar individuos a fazerem uma escolha dentro de um conjunto de alternativas pré-especificadas, porém, conforme Khurum, Gorschek e Wilson (2013a), embora diversas pesquisas apresentem ideias e soluções para medir/avaliar a tomada de decisão sobre o desenvolvimento de produtos de software, tais contribuições se apresentam de forma isolada focando somente em custos ou características do produto como usabilidade.

Deste modo, dado o propósito desta tese relacionado a impactos em atividades de

codificação, considerando a avaliação de métricas e qualidade em termos de design, foram investigados modelos de tomada de decisão relativos à manutenção de software, através da busca de artigos de forma não-exaustiva, conforme a Tabela 5.

Tabela 5 – Dados da busca

Origem	String de busca	Período	Resultados
IEEEXplore	"("Document Title":decision-making model OR "Document Title":decision support model) AND "Abstract":software maintenance	2010-2018	2
ACM	acmdlTitle:(+decision-making +decision +support) AND recordAbstract:(+software +maintenance)	Sem filtro	2
ScienceDirect	Title: decision-making OR decision support Title, abstract, keywords: software maintenance Terms: software maintenance	2010-2018	2
Springer	"words: software maintenance" title: decision-making model	2010-2018	1
Total			7

Dentre os sete artigos encontrados, três foram retirados por se encontrarem fora do escopo analisado ou por não possuirem citações. Os resultados da busca conforme a Tabela 6, apresentam abordagens para a tomada de decisão com direcionamentos diversificados: com viés em arquitetura de software associando sistemas de suporte a decisão, utilizando inferência bayesiana com multiplos fatores, definindo modelos de maturidade ou utilizando bases de conhecimento.

Nesta análise inicial verificamos a necessidade de especialistas como opinadores, seja definindo ou atribuindo valores a fatores que direcionam tais decisões, e, a necessidade de um histórico de projetos similares para apoio no julgamento, o que dificultaria um cenário onde há ausência de parâmetros para comparações.

De maneira informal, a prática sobre o uso de base histórica se confirmou quando em entrevista por telefone com um dos membros do time de precificação de uma consultoria, um profissional com três anos de experiência avaliando e estimando projetos de software em diversos segmentos de mercado, informou que o trabalho de estimativa de custos e definição de times para desenvolvimento/manutenção se apoiava em uma base histórica de projetos, onde existia a busca por propostas similares, seja em tamanho, négocio ou planejamento.

Após o levantamento inicial, foram analisados trabalhos relacionados a tomada de decisão acrescentando o fator custo, verificando no IEEEXplore por meio da string de busca: (("Document Title":"software maintenance"AND "Document Title":cost AND "Document Title":decision making AND "Document Title":"decisional framework") OR ("Abstract":"software maintenance"AND "Abstract":cost AND "Abstract":decision making OR

Tabela 6 – Resultados da busca

Titulo	Ano	Autores	Resumo	Citações
Decision support for moving from a single product to a product portfolio in evolving software systems	2010	Muhammad Irfan Ullah, Günther Ruhe, Vahid Garousi	Propõe um modelo de tomada de decisão baseado nas preferências dos clientes direcionando fatores que possivelmente permitam a migração de sistemas únicos, subdividindo-os em um portifolio de sistemas	20
Constraint-Based Consistency Checking between Design Decisions and Component Models for Supporting Software Architecture Evolution	2012	Ioanna Lytra ; Huy Tran ; Uwe Zdun	Checa a consistência entre modelos de decisão e componentes arquiteturais	5
Challenges of shared decision-making: A multiple case study of agile software development	2012	Nils Brede Moe, Aybüke Aurum, Tore Dybå	Apresenta os principais desafios relativos a tomadas de decisão compartilhadas	53
Building a maintenance policy through a multi-criterion decision-making model	2012	Elahe Faghihinia1 and Naser Mollaverdi	Aborda a decisões baseadas em múltiplos fatores de competitividade associados a um sistema	11

"Abstract":"decisional framework")), a partir de 2000, tendo como resultado seis ocorrências.

Após leitura dos abstracts e considerando no mínimo cinco citações, foi selecionado um artigo, sendo ele *A decisional framework for legacy system management* (LUCIA; FASOLINO; POMPELLE, 2001) objetivando a criação de um conjunto de fatores que permite suportar a tomada de decisão, categorizados da seguinte forma:

- **Valor do negócio** observando o valor econômico, valor dos dados, utilidade e especialização.
- **Valor técnico** considerando manutenibilidade, arquitetura, deterioração e obsolescência do software.

Como validação, o framework foi executado em um projeto na indústria de TI, utilizando um conjunto de ferramentas para a coleta de métricas, bem como, uma série de entrevistas foi aplicada considerando o aspecto de negócio. Contudo, não foi divulgada a tomada de decisão adotada após a utilização da avaliação.

Mais adiante, utilizando a tecnica apresentada por (WOHLIN, 2014), dentro do trabalho proposto por (LUCIA; FASOLINO; POMPELLE, 2001), foram encontrados mais seis artigos, sendo dois deles removidos por não possuirem citações, conforme a Tabela 7

Tabela 7 – Resultados da busca

Titulo	Ano	Autores	Resumo	Citações
Empirical analysis of massive maintenance processes	2002	A. De Lucia, A. Pannella, E. Pompella, S. Stefanucci	Verifica a correlação entre tamanho da manutenção, métricas de produtividade e modelos produzidos para estimativa de custos	1
An AHP-Based Framework for Quality and Security Evaluation	2009	V. Casola, A.R. Fasolino, N. Mazzocca, P. Tramontana	Propõe um framework baseado em um Processo Hierarquico Analítico (AHP) ((SAATY, 2008)) envolvendo o cliente, fornecendo visões de qualidade	6
Legacy Asset Analysis and Integration in Model-Driven SOA Solution	2010	Nianjun Zhou, Liang-Jie Zhang, Yi-Min Chee, Lei Chen	Avalia a maturidade do software legado para sua transformação em serviços por meio do uso de um método de nome SOMA-ME	6
A preliminary review of legacy information systems evaluation models	2013	Humairath Km Abu Bakar, Rozilawati Razali	Estudo que avalia lacunas em modelos de avaliação de software legado destacando valor de negócio, qualidade técnica e características organizacionais	1

Considerando a aplicação de modelos utilizados no âmbito da economia, buscando pelos termos "decision making; cost-benefit analysis; analytic hierarchy process;", no IEEEXplore e ACM a partir de 2010, considerando artigos acima de dez citações, foi encontrado um artigo que cita a possibilidade no gerenciamento de débitos técnicos [8] por meio de tomada de decisão sugerindo o uso de quatro abordagens (SEAMAN et al., 2012), sendo estas:

- Análise de custo-benefício simples ((QUAH; HALDANE, 2007)), apresenta uma forma simplificada de se avaliar por meio fatores e julgamento humano os impactos de uma determinada ação, é importante o uso de dados históricos para permitir uma

[8] Não-conformidades postegardas devido a fatores externos à qualidade do software. (CUNNINGHAM, 1993)

melhor avaliação dos items. As áreas comuns de aplicação são transações comerciais, decisões de negócio funcionais ou projetos de investimentos.

- Processo Hierarquico Analítico ((SAATY, 2008)), voltado para equipes de pessoas trabalhando em problemas complexos, cujas repercussões são no longo prazo. ùtil para a comparação e a quantificação de elementos difíceis ou onde comunicação de membros da equipe é impedida por suas diferentes especializações.

- Abordagem de Portifólio[9], o objetivo dessa abordagem é priorizar o ativo que possivelmente possa maximizar o retorno do investimento ou minimizar o risco. Esse modelo possui considerações específicas para o modelo de finanças como a dviisão de ativos, no caso, de débitos técnicos essa abordagem não seria possível.

- Opções, permite formular hipóteses sobre o valor presente líquido (VPL)[10], acrescentando variações para atingir o valor positivo.

Do ponto de vista teórico, foi verificado que a maioria dos trabalhos avaliados possuiam similaridades na forma operacional: criando modelos hierárquicos, considerando aspectos negociais e/ou técnicos, associando-lhes características, itens e consequentemente, atribuindo-lhes pesos ou conceitos passiveis de julgamento por especialistas.

Contudo, se considerarmos a dinâmica e o tempo de envolvimento de especialistas nas avaliações, é possível que haja um desincronismo entre as verificações, o *time-to-market* e a agilidade nas entregas dos times. Ainda neste contexto, verificamos que o fator custo e o *time-to-market* nem sempre estão entre os principais fatores na tomada de decisão, pois, em determinadas situações como a descrita por (TRIANTAPHYLLOU et al., 1997), a confiabilidade, a reparabilidade ou a disponibilidade são mais importantes.

Segundo Gorschek et al. (2006), a observação do mundo real antes da formulação das questões de pesquisa é considerado um fator crítico, onde a pesquisa deve se conectar as necessidades e as percepções dos profissionais em suas empresas, caso contrário, a ausência desta conexão poderia comprometer a participação destes profissionais no processo de construção do conhecimento.

Deste modo, partindo do contexto acadêmico analisando os direcionamentos e teorias, para a observação de projetos de software na industria de TI, um cenário comum é o citado por (LUCIA; FASOLINO; POMPELLE, 2001), onde a modificação do software tem sido realizada quase sempre na forma de "consertos rápidos", sem documentação e com poucos recursos possuindo experiência suficiente para implementar tais solicitações. Como consequência, o tempo e o custo necessário para atender as solicitações de mudança alcançam valores inaceitáveis pelos usuários dos sistemas.

Diante deste cenário, este trabalho de tese se iniciou avaliando um projeto de software em execução dentro de uma consultoria de TI com as seguintes características:

[9] http://schultzcollins.com/static/uploads/2014/10/Theory$_a nd_P ractice_0 8_0 4.pdf - acessadoem$10/2018
[10] http://pt.wikipedia.org/wiki/Valor$_p resente_l íquido - acessadoem$10/2018

- Atuação global: presença em mais de 70 países;
- Número de funcionários global: aproximadamente quatrocentros mil;
- Número de funcionários local: aproximadamente dois mil e trezentos;
- Atuação em segmentos de mercado variados, sendo eles, financeiro, telecomunicações, utilities, mídia, saúde e serviços públicos, dentre outros;
- Número de funcionários atuando no projeto: aproximadamente cinquenta dividido em duas localidades (Recife e Curitiba) e três torres: canais, sustentação e portal;

Neste projeto, relacionado à industria de telecomunicações, houve uma avaliação sobre o código produzido e o time de desenvolvimento, considerando os seguintes objetivos:

- **Com relação ao código-fonte**:
  - A avaliação estrutural e estática do código entregue ao cliente;
  - A geração de evidências para compartilhamento com o time sobre os pontos de atenção com relação a entrega, criando ações para mitigação de riscos recorrentes no futuro.
- **Com relação ao time de desenvolvimento**:
  - Experiência técnica em projetos;
  - Participação nos treinamentos ministrados;
  - Domínio das ferramentas de trabalho;
  - Entendimento das regras de negócio e fluxos funcionais;
  - Aderência com as decisões da liderança;
  - Sinergia com outros membros do Time;
  - Gerar insumos para orientar as decisões estratégicas adotadas pela liderança.

O projeto avaliado foi mantido por uma equipe composta por dez profissionais na torre de sustentação, onde o código-fonte foi caracterizado por meio de alguns procedimentos, sendo eles:

- Procedimentos realizados
  - Inspeção manual do código e avaliação estrutural do código;
  - Utilização de ferramentas de análise estática (Sonarqube [11] e STAN [12]), observando os seguintes pontos:

[11] https://www.sonarqube.org/
[12] http://stan4j.com/

    - Violação de boas práticas de programação;
    - Violação de padrões de desenvolvimento;
    - Complexidade do código desenvolvido;
    - Duplicidade de código;
    - Utilização de testes unitários.

- Características
    - Linhas de código: 4015
    - Operações/Métodos: 234
    - Classes: 46
    - Violações: 336

Após a verificação do código-fonte foi elaborado um resumo com os principais problemas encontrados conforme a Tabela 8.

Tabela 8 – Resumos dos principais problemas identificados

Problemas	Descrição
Alta complexidade	Rotinas complexas que dificultam a manutenção e testes do código
Falta de Coesão	Componentes ou métodos que fazem mais que um papel, causando dificuldade na manutenção
Duplicidade de código	Rotinas duplicadas que dificultam a manutenção.
Falta de testes unitários	Dificuldade em garantir a estabilidade da aplicação
Violação de boas práticas	Estes ofensores podem apontar para uma série de problemas potenciais, desde bugs a problemas de performance.
Violação de padrões de projeto	Problemas estruturais identificados na construção da camada de persistência de dados (DAO).

Para a avaliação dos times, que incluiu profissionais de todas as torres em Recife, foi elaborado um questionário descrito no Apêndice B, contendo questões distribuidas em tópicos pertinentes ao cotidiano da operação.

Os resultados apresentados na Tabela 9, identificaram que Experiência técnica em projetos e Treinamentos, obtiveram um percentual bem abaixo dos demais, sendo neste último indicado que *os prazos de entrega atrapalhavam a absorção do conhecimento* em 57% das respostas.

A partir dos resultados obtidos com a avaliação, foi elaborado um plano contendo alguns direcionamentos, sendo eles:

- Novo processo de revisão de código antes da entrega dos pacotes;
- Intercâmbio entre pessoas, dado que o projeto era executado em dois estados;

Tabela 9 – Mapeamento do time

Categoria	%
Experiência técnica em projetos	43%
Treinamentos	55%
Domínio de ferramentas	79%
Entendimento do negócio funcional	77%
Aderência com as decisões da liderança	82%
Sinergia com outros membros do time	82%

- Reuniões diárias;
- Uso de ferramentas automatizadas para análise estática de código;
- Aumento na quantidade de treinamentos técnicos;
- Alocação de profissionais mais experientes;
- Adoção de ferramentas de testes automatizados.

De modo geral, algumas ações surtiram efeito dado que o time absorveu mais conhecimento funcional e técnico, porém, o conhecimento em si permanecia na cabeça do desenvolvedor, o que em caso de acréscimo ou saída de profissionais nos times, deveria haver um novo trabalho de repasse de conhecimento, implicando em mais custos. Ainda nesta avaliação industrial, outros pontos foram considerados, sendo eles:

- A falta de análise de correlação entre as ações e os resultados, ou seja, não foi evidenciado que a alocação de profissionais experientes e/ou o aumento na quantidade de treinamentos, melhoraram a qualidade do código-fonte entregue;
- Sobre o objetivo em se gerar insumos para orientar as decisões estratégicas adotadas pela liderança, não ficou claro quais os insumos e se houve uma aferição se estes de fato apoiaram a tomada de decisão.
- E por fim, a avaliação dos times ocorreu de forma única sem uma re-checagem pós o plano aplicado.

Quando consideramos um cenário onde existe a mudança de fornecedores, é possível que haja a perda de conhecimento, dado que é comum as informações estarem disponíveis somente em forma de código-fonte como artefato.

Para Lanza e Marinescu (2007), uma das possibilidades para a melhoria do entendimento do software seria a coleta de métricas tornando possível sua caracterização.

Historicamente a abordagem com o uso de métricas tais como Linhas de Código (LoC), Complexidade Ciclomática ((MCCABE, 1976)) e Halstead Software Science ((HALSTEAD

et al., 1977)) são adotados por vários anos. A abordagem com o uso de Linhas de Código tem sido utilizada desde o final da década de 60 ((FENTON; NEIL, 1999)), enquanto as métricas de Halstead e McCabe no final de 1970. Tais métricas foram estudadas durante o período entre os anos 70 até o início dos anos 90, quando as métricas orientadas a objetos se tornaram populares graças as pesquisas e métricas propostas por (CHIDAMBER; KEMERER, 1994) e (LI; HENRY, 1993).

Metricas de código-fonte são um componente importante para o processo de medição de software, sendo normalmente utilizadas para a medição e melhoria da qualidade do software em si. Adicionalmente, tais abordagens tem sido utilizadas em uma larga variedade de aplicações e experimentos relacionados de forma geral (predição de falhas, testes, refatoramento) para avaliar a qualidade completa do software. Dado o interesse neste tema, conforme Nuñez-Varela et al. (2017), diversas métricas são propostas constantemente dificultando o monitoramento por parte de pesquisadores sobre o estado atual e tendências.

Relacionando o estado da arte à análise apresentada sobre um projeto na industria de TI, foi possivel identificar alguns pontos:

- Embora existam opções para análise de tomada de decisão, as decisões na avaliação industrial apresentada foram tomadas sem a utilização de algumas dessas abordagens. Neste contexto, as ações foram mais baseadas no *feeling* do gestor, do que em métodos e/ou formulações teóricas, ou seja, sem a utilização de *expert opinions* ou o uso de um conjunto de fatores;

- Considerando o uso de métricas, não houve a interpretação dos resultados como uma abordagem para suporte a uma tomada de decisão. Ocorreu uma ação de treinamentos para melhoria do conhecimento técnico da equipe, porém, não houve uma análise se o estado do projeto em termos de qualidade era o complicador das entregas.

- Outro ponto foi a falta de adoção de uma estratégia para redução do passivo em não-conformidades no código-fonte, além das correções indicadas pelas ferramentas.

Diante destes pontos, esta tese buscou entender se essa prática era algo comum em outros projetos e se a adoção de um método com baixo impacto no modelo de trabalho dos times, poderia contribuir para a melhoria do cenário avaliado.

Sendo assim, durante um período de 2 anos (2015-2016) houve um trabalho de mapeamento da prática adotada em projetos de software com relação a codificação e impactos, com a participação direta do autor desta tese, segmentados em três mercados: financeiro, mídia e telecomunicações. Deste modo, a partir dos resultados obtidos por meio do acompanhamento dos times, questionários e reuniões junto com lideranças técnicas, foi possível direcionar ações estratégicas para a melhoria da qualidade dos projetos.

No decorrer das atividades de mapeamento e avaliações, surgiram três hipóteses para guiar o trabalho de pesquisa, sendo elas:

- H1. Apesar das incertezas causadas por tomadas de decisão não-programadas no contexto de codificação em projetos de software, a aplicação do conhecimento sobre modelos de avaliação de qualidade de software, fornece uma percepção de segurança e resultado nas ações direcionadas.

- H2. A conversão de métricas em indicadores advindas da análise estática de código, contribui como direcionador em ações de melhoria de código em projetos de software, sendo possível simular e projetar custos financeiros.

- H3. Considerando a quantidade de não-conformidades geradas durante a codificação, as atividades de correções direcionadas por meio da análise histórica e um modelo que proponha um indice de priorização, trazem a percepção no qual as atividades executadas apresentam resultados de forma antecipada.

Outro ponto importante no estabelecimento do problema foi a análise de trabalhos considerando o propósito desta tese, cuja a preocupação não está somente na definição de um método, mas, no esclarecimento e direcionamento de ações de melhoria de forma contínua. Sendo assim, procuramos identificar trabalhos com abordagens em mineração em repositórios de software, predição de impacto, métodos e ferramentas, no qual destacamos três dessas verificações abaixo:

- (KAGDI; MALETIC, 2006b) Propõe um framework onde por meio da análise de impactos e mineração de repositório de softwares, consegue prever a mudança com base na análise histórica. **Crítica:** ausência de um direcionamento em termos de ações para os objetos suscetíveis à mudanças; desconsidera o grau de importância dentro do contexto do projeto em termos de design ou financeiro.

- (MUSCO et al., 2015) Verifica o impacto em alterações de operações de classes dentro do paradigma orientado a objetos, considerando classes de testes que falham. Utilizando um algoritmo de aprendizagem de máquina são verificadas as alterações passadas para a predição de futuros impactos. **Crítica:** considerando ponto do item anterior, o fato é que a predição de impacto não vem com uma abordagem proativa no direcionamento de ações.

- (PLöSCH et al., 2009) Propõe um método e uma ferramenta avaliando duas versões de projetos open-source. **Crítica:** não define uma visão de projeção de custos, não observa a questão do design por meio de um modelo de avaliação de qualidade neste sentido.

Até aqui, o que pode ser visto foi o crescimento do mercado brasileiro em TI, o aumento em demandas de desenvolvimento e manutenção de software, abordagens sobre análise decisional e a definição de fatores para a tomada de decisão considerando práticas no contexto econômico, porém, quando ocorreu a avaliação de um projeto real na indústria de TI, foi verificada a ausência na aplicação das abordagens estudadas.

Diante das análises até agora estudadas considerando o estado da arte e a sua aplicação na indústria de TI, verificamos a necessidade em se especificar **um método baseado em métricas para predição de impacto em atividades de codificação em projetos de software**. Deste modo, esta tese buscou como diferencial definir de forma estruturada um roteiro para a avaliação em projetos de software, exercitando os seguintes pontos:

- **Garantir que os resultados sejam entendidos por todos os envolvidos no projeto**, neste aspecto um ponto a se observar é a quantidade de metricas disponíveis: o que segundo Saraiva et al. (2015), atingem uma quantidade de 67 métricas orientadas a aspectos e 575 orientadas a objetos, e para Nuñez-Varela et al. (2017), 300 métricas voltadas para código-fonte. Neste sentido, com aproximadamente 1000 métricas segregadas em uma série de categorias (complexidade, acoplamento, volume, qualidade do design, probabilidade de falhas, dentre outros) e considerando que a maioria dos estudos se relacionam ao estado da pesquisa em âmbito acadêmico, se torna complexo e confuso para um profissional selecionar métricas e definir um roteiro de avaliação;

- **Desmistificar a visão que métricas de software são utilizadas como avaliação do individuo e não do produto**, conforme Umarji e Seaman (2008) desenvolvedores podem se sentir ameaçados dado que métricas podem ser utilizadas contra eles. Neste sentido, o direcionamento é a coleta e a avaliação de forma segura, evitando o enviesamento de informações;

- **Unir visões técnicas e negociais, direcionando ações que contribuem para a tomada de decisão**, como descrito no início desta Seção, conforme as ausências descritas por (KHURUM; GORSCHEK; WILSON, 2013a), não seria adequado considerar somente um conjunto de aspectos seja no âmbito técnico ou financeiro, desta forma, a proposição deste trabalho se utiliza de pontos de vista que subsidiam a tomada de decisão com foco maior de abrangência;

- **Direcionar ações onde equipes de desenvolvimento consigam executar atividades de refatoramento e melhoria do código-fonte de forma gradual**, o interesse neste ponto está em diminuir o custo e trazer visibilidade em ações de melhoria de forma controlada. Neste contexto, Buzluca e Eski (2011) afirma que "classes com baixa qualidade possui uma maior probabilidade de alteração".

Nesta tese, a abordagem consiste em analisar a evolução das classes, associadas a abordagens do mercado financeiro e métricas.

Como resultado, o método proposto foi aplicado em um conjunto de projetos de software em uma consultoria de TI, apoiando a tomada de decisões relativas a atividades de manutenção de software sendo elas corretivas, adaptativas ou perfectivas, bem como, relativas à precificação de atividades de construção em projetos já em fase de desenvolvimento. Contudo, visando expor o entendimento com dados passíveis de publicação por este trabalho, esta mesma abordagem também foi aplicada em projetos de código aberto.

1.2 METODOLOGIA

Motivado pelas hipóteses e problemas levantados, esta tese seguiu o formato proposto por (BASILI; WEISS, 1984), definindo um conjunto de objetivos, perguntas e evidências que guiam o trabalho de pesquisa como um todo. Evidências neste sentido seriam saída e resultados aceitáveis que consigam validar as perguntas estabelecidas. A Figura 2, apresenta um guia de atuação auxiliando como orientador no processo de construção do conhecimento deste trabalho, divididas em 4 fases sendo descritas a seguir:

- **Coletar** visto na Seção 4.2, significa obter os entendimentos necessários sobre os repositorios e métricas, bem como o que faz sentido em termos de uso para a proposição em questão. Desta forma os objetivos desta fase foram dois, como se segue:
  - **Extração de dados históricos dos projetos em repositórios de código-fonte e controle de versão**, o que se procura compreender é se faz sentido utilizar tal abordagem, se perguntando: "É comum o uso de repositórios para se recuperar informações históricas sobre projetos de software, se sim, quais os aspectos são verificados?";
  - **Identificação de um catálogo efetivo de métricas para o mapeamento dos dados obtidos**, dado o número de métricas existentes, quais fazem sentido para o mapeamento em atributos de qualidade, apresentando uma visão sobre o estado atual do projeto.

  Como resultado desta fase, foram esclarecidas quais as métricas e como se dará o seu mapeamento entre os atributos de qualidade do design, respondendo a Hipótese 1.

- **Avaliar** a ser apresentado na Seção 4.3, o que se procura obter como resultados são as visões significativas sobre o estado atual do projeto, onde estas possam subsidiar mesmo de forma inicial alguma tomada de decisão. Desta forma os objetivos desta fase foram três, como se segue:

  - **Identificação e definição de *thresholds***, de acordo com entendimento de métricas obtidas de cada revisão, entender e definir quais os limites minimos e máximos que podem guiar e direcionar um entendimento sobre a qualidade do projeto, estabelecendo indicadores dinâmicos;
  - **Entender a relação entre métricas primitivas e compostas e atributos de qualidade de design** neste contexto, verificar quais os modelos de avaliação de qualidade de software, sua abordagem de mapeamento e quais as métricas básicas que subsidiam tais modelos;
  - **Caracterizar um ou mais indices de manutenibilidade** dados os modelos de referência definidos pela academia para manutenibilidade de software, pesquisar e avaliar uma forma para a definição de métricas que consequentemente gerem indicadores descrevendo a situação do projeto em termos de manutenção.

  Neste sentido, os resultados esperados estão em caracterizar e definir um conjunto de visões com base no histórico de revisões do projeto, exibindo informações que permitem avaliar o estado atual do projeto em termos de: aspectos de design, limites de qualidade de código e indicadores de manutenibilidade.

- **Direcionar** visto na Seção 4.4, quais as ações possíveis orientadas por indicadores de prioridade. Esta fase consiste em definir uma visão onde se torna possível entender os custos associados com o panorama atual do projeto em termos de manutenibilidade, projetando-os por um determinado período, bem como, sugerir ações de melhoria em classes do projeto, baseado em um indicador de priorização definido por este trabalho. Deste modo os objetivos desta fase foram dois, conforme se segue:

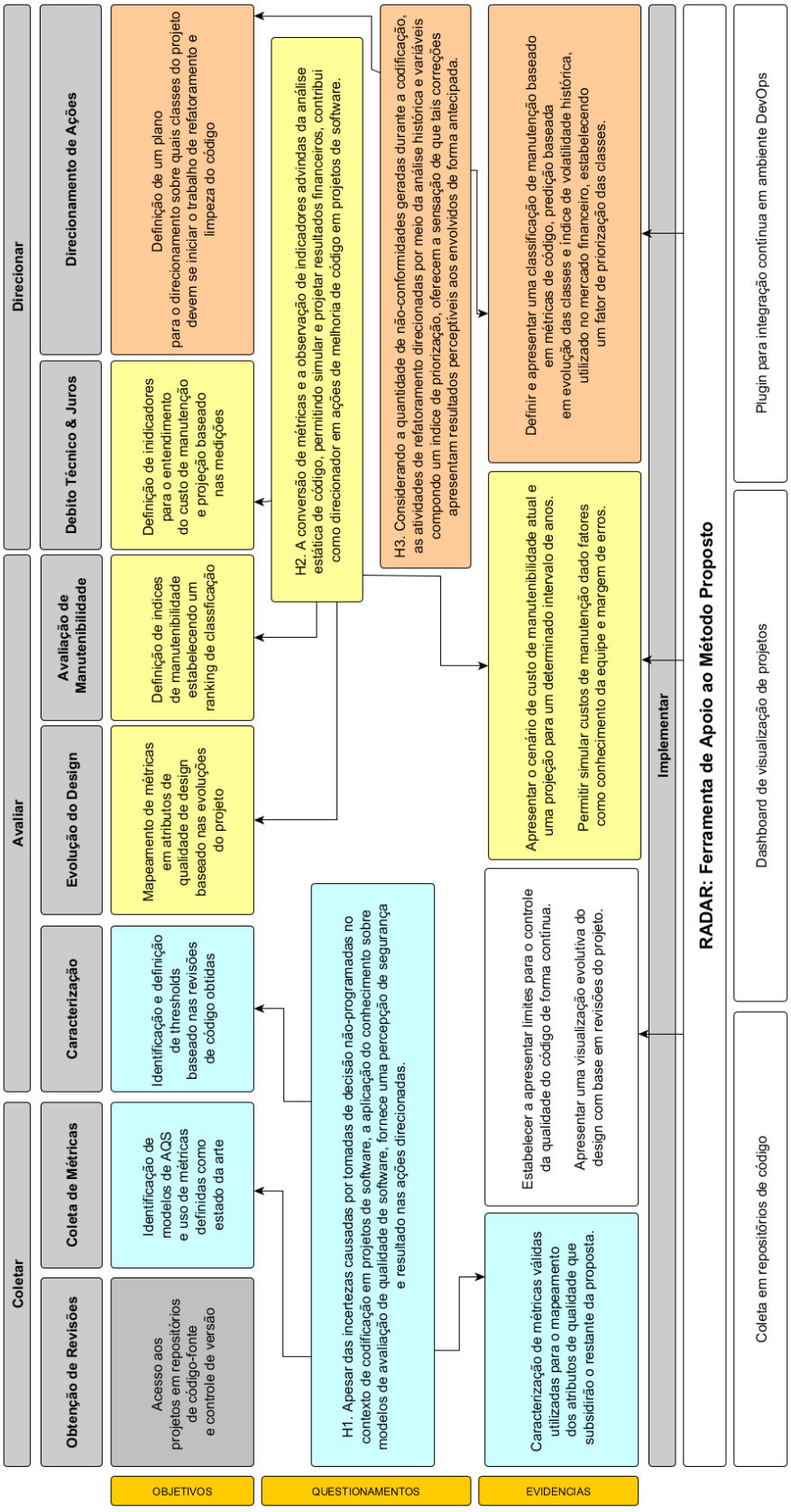
  - **Simular e verificar a condição financeira do projeto** diante de fatores como o entendimento da equipe, percentual de defeitos introduzidos em atividades de manutenção e percentual de manutenção anual, simular e verificar qual o custo financeiro caso o projeto continue seguindo o desenvolvimento conforme o estado atual, bem como, qual o retorno do investimento caso sejam adotadas ações de melhoria;
  - **Direcionamento em ações de melhoria** conforme os resultados fornecidos pelo objetivo anterior, definir um racional de atuação em melhorias do código-fonte considerando a condição de projeto em andamento ou por iniciar.

  Como resultado, esta fase espera definir e apresentar os indicadores necessários para a tomada de decisões sobre o projeto em termos de codificação e evolução de funcionalidades, respondendo assim, as Hipótese 2 e 3.

- **Implementar** direcionar e desenvolver um solução de software aderente ao método proposto, que capture e forneça indicadores e direcionamentos de forma executiva para gestores. Desta forma, a implementação ficou dividida em 3 partes com as seguintes caracteristicas:
  - **Coleta em repositórios de código**, o objetivo deste item está em recuperar o código-fonte bem como seus versionamentos, possibilitando identificar não somente aspectos do código como também: autores, datas e dados que permitem resgatar e mapear informações sobre:
    - Evolução e introdução de defeitos por autor, fornecendo visões sobre as habilidades da equipe;
    - Quantidade de *commits* por autor, permitindo identificar o percentual de atividade de membros de um projeto;
  - ***Dashboard* de visualização de projetos**, considerando:
    - Visão geral de qualidade sobre projetos considerando complexidade e duplicidade;
    - Evolução sobre o design, considerando o mapeamento entre métricas e atributos de qualidade;
    - Categorização e mapeamento sobre manuteniblidade baseados na norma (ISO/IEC 9126, 2001);
    - Priorização na correção de defeitos;
  - **Plugin para integração continua em ambiente DevOps**, permitindo um acompanhamento *on-time* do projeto.

Para alcançar os resultados esperados, foram utilizadas metodologias específicas tais como: brainstorming e discussão em grupo, bem como, estudos de caso.

Figura 2 – Roadmap para guiar o trabalho de pesquisa



1.3 OBJETIVOS GERAIS E ESPECÍFICOS

De forma geral, o objetivo alcançado por esta tese foi em se estabelecer um entendimento comum sobre qualidade de software entre os envolvidos no projeto, através do uso de métricas. Além disso, o entendimento obtido trouxe uma maior segurança na análise decisional, dada a criação de painéis e indicadores. De maneira prática, a abordagem para o monitoramento sobre a qualidade dos projetos de forma contínua, permitiu caracterizar e direcionar ações de melhoria contínua de forma gradual em partes do sistema.

1.3.1 Objetivos específicos

De maneira específica, este trabalho entregou e atingiu os objetivos levantados na seção 1.1, como se segue:

- Mesmo com a existência de uma variedade de modelos de avaliação e métricas de software, foi possível de forma segura avaliar e direcionar ações em nível estratégico, abstraindo a complexidade teórica para o âmbito indutrial por meio de indicadores;
- Considerando o código-fonte como o principal artefato do projeto, o entendimento sobre a qualidade do software tendo como base a análise estática, trouxe a percepção para os times de que métricas são fatores para a avaliação dos projetos e não das pessoas.
- O direcionamento de ações de melhoria de forma gradual, contribuiu para o uso racional do tempo e do esforço dos recursos envolvidos em atividades de construção, trazendo uma percepção de valor para os gestores do projeto em atividades de melhoria e refatoramento.
- O esforço na aplicação do método foi abstraído pela automatização da coleta de dados e o monitoramento dos projetos de forma contínua, sendo possível direcionar ações com base em indicadores, projeções de custos, classificação e melhoria contínua em partes do sistema.

1.3.2 Fora do escopo

Alguns itens devem ser considerados como fora do escopo deste trabalho, sendo estes:

- A criação de um novo modelo de avaliação de qualidade;
- A criação de novas métricas de software, sendo elas primitivas ou compostas;
- A construção de ferramentas de avaliação de propósito comercial;
- Apresentar um modelo comparativo e indicativo entre diversas linguagens de programação ou tecnologias;

- Direcionar o entendimento de qualidade para uma possível migração de partes do projeto para outras tecnologias;
- A análise de outros artefatos além do código-fonte do projeto para subsidiar os resultados das avaliações.

1.4 ORGANIZAÇÃO DA TESE

O restante deste documento, está estruturado da seguinte maneira:

- No Capitulo 2, será apresentada uma Revisão de Literatura, contemplando a obtenção das publicações que embasaram a construção desta tese.
- No Capitulo 3, serão apresentados o estado atual sobre métricas de software, modelos de monitoramento em termos de Avaliação de Qualidade de Software (AQS) e quais as iniciativas voltadas para a classificação e estimativa de custos em termos de débito técnico e juros.
- No Capitulo 4, a apresentação do método proposto, considerando: uma visão geral, a proposta, o resultado da aplicação em projetos reais, validações em 3 projetos reais nos segmentos de mídia, telecomunicações e financeiro, e por fim, os trabalhos relacionados;
- E finalmente, no Capítulo 7, as Considerações finais, contribuições e os trabalhos futuros.

1.5 TRABALHOS APRESENTADOS

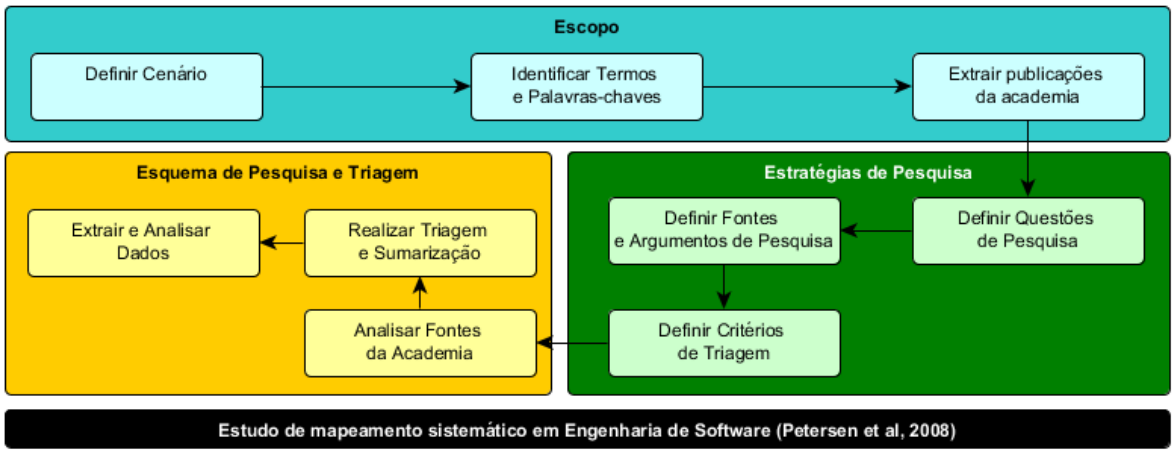
Apresentação Oral. AVALIAÇÃO DE SOFTWARE BASEADO EM MÉTRICAS: UMA ABORDAGEM PRÁTICA. XII CONNEPI - Congresso Norte-Nordeste de Pesquisa e Inovação, Novembro, 2018.

2 REVISÃO DA LITERATURA

Neste Capítulo serão apresentadas a construção, execução e os resultados obtidos através da análise de fontes de dados na academia, buscando situar os objetivos desta tese. Considerando as hipóteses apresentadas na Seção 1.1, o objetivo neste Capítulo está em investigar dentro da etapa de Revisão da Literatura, se os direcionamentos tomados como guia para esta tese são pertinentes.

Sendo essa revisão realizada durante o período de maio a dezembro de 2016, como ponto de partida, foi definida uma investigação na literatura oriunda da Evidence-Based Software Engineering (EBSE) proposta por (PETERSEN et al., 2008) entitulada Systematic Mapping Study (SMS). Essa metodologia é recomendada para a extração com termos abrangentes e sumarização dos resultados.

Figura 3 – Fluxo de Metodologia de Pesquisa



2.1 ESCOPO

A Definição do Cenário permite estabelecer as considerações básicas que subsidiam a construção dos Capítulos 3 e 4 desta tese. Neste passo, analisamos as hipóteses para extrairmos as questões de pesquisa, conforme os três cenários destacados abaixo:

- **Cenário 1 - Avaliação de modelos de qualidade de software**, relativo à Hipótese 1, este ponto busca entender se os modelos de avaliação de qualidade existentes são claros em termos de uso podendo suportar a tomada de decisão, ou seja, dado que um determinado projeto necessita de uma definição quanto à sua parada ou continuidade, quais as características de qualidade deveriam ser consideradas para tal decisão. Neste sentido, as ações para responder a esse item será checar o estado da arte, modelos mais relevantes e características mais referenciadas, permitindo

facilitar o entendimento sobre o que deve merecer uma maior atenção ao se tomar uma decisão não-programada;

- **Cenário 2 - Análise de métricas de software e custo**, relativo a Hipótese 2, o que está sendo investigado é a existencia de estudos sobre os limites pré-definidos para métricas de software, como por exemplo: qual o limite aceitável para complexidade ciclomática, linhas de código, acoplamento que permitam caracterizar a qualidade do software de forma prática. Neste cenário, possivelmente os valores além dos limites pré-definidos se caracterizam como não-conformidades;

- **Cenário 3 - Priorização e ações de refatoramento**, relativo a Hipótese 3, considera que à medida em que se constrói ou se evoluem projetos de software, consequentemente são inseridas não-conformidades que no decorrer do projeto caso não corrigidas podem afetar o desempenho das atividades de desenvolvimento. Deste modo, com o aumento do volume a ser corrigido, por onde começar, o que se priorizar e qual a efetividade dessa ação para as próximas entregas, torna-se uma decisão não-trivial. Neste sentido, a resposta que se busca é como estão sendo abordados esse tema no estado da arte.

Dado os subsídios fornecidos pelo passo anterior, neste segundo passo, Identificar Termos e Palavras-Chaves, o objetivo é a elaboração de argumentos de busca mais precisos para o restante das atividades. E finalizando, o passo de Extrair Publicações da Academia faz uma verificação mais aprofundada e detalhada em fontes acadêmicas.

2.2 ESTRATÉGIA DE PESQUISA

Nesta subseção, serão descritas as estratégias tomadas para a execução da pesquisa nas fontes da academia, conforme as orientações para o desenvolvimento do SMS.

2.2.1 Questões de Pesquisa

O SMS é uma metodologia que se apoia em uma extração de dados de triagem segmentada em Questões de Pesquisa (QPs) com o propósito de identificar um fenômeno registrado na literatura. A proposta deste estudo visa identificar o estado atual no uso de métricas de software, sua propensão ao mapeamento em atributos de qualidade, bem como, o aspecto financeiro relacionado à atividade de melhoria de qualidade de software, além de mapear trabalhos relevantes ao tema que se encontram presentes até o momento na literatura. A revisão pretende responder três questões, a saber:

- (Cenário 1 - QP1) Observando o estado da arte dos modelos de avaliação de qualidade de software, quais os modelos mais relevantes, mais utilizados e quais as características que podem suportar uma tomada de decisão?

- (Cenário 2 - QP2) Considerando métricas como medidas de avaliação de qualidade interna do software, quais os limites definidos no estado da arte que possam direcionar um entendimento de não-conformidades, possivelmente gerando ações de melhoria de código em projetos de software?

- (Cenário 3 - QP3) Dado o volume de não-conformidades em projetos de software, quais as abordagens existentes que predizem ou direcionam a priorização de correções ou refatoramento?

As respostas serão investigadas em fontes acadêmicas disponíveis em repositórios online, sendo descritas em detalhes na próxima subseção.

2.2.2 Definir Fontes e Argumento de Pesquisa

Para esta seção, os argumentos de busca foram elaborados conforme as questões definidas na Seção anterior, sendo assim o direcionamento e os resultados de pesquisa foram mais assertivos. Para cada busca houve uma string elaborada, considerando que as perguntas e os pontos são bastantes conhecidos e estudados pela academia. Sendo assim, seguem como foram executadas as buscas para cada questão:

- Para a QP1, foram pesquisadas revisões sistemáticas relacionadas ao tema modelos de avaliação de qualidade com a seguinte palavra-chave: "'Software quality models' review", no SemanticScholar [1] e no Google Scholar [2] a seguinte string de busca: "Software quality models"+"comparative study". Além disso, foram executadas revisões em referências bibliográficas nos trabalhos obtidos visando obter mais informações de maneira minuciosa sobre o tema investigado.

- Para a QP2, para a análise de limites para métricas de software, foi utilizada a seguinte string de busca: "(((internal quality) AND thresholds) AND software metrics)"no IEEEExplore [3] e no Google Scholar, a seguinte string de busca: "software metrics thresholds". Além disso, foram executadas revisões em referências bibliográficas nos trabalhos obtidos visando obter mais informações de maneira minuciosa sobre o tema investigado.

- Para a QP3, para a análise de de abordagens existentes visando a priorização de correções de não-conformidades, foi utilizada no Google Scholar, a seguinte string de busca: Code smells"+Prioritize+Refactoring". Além disso, foram executadas revisões em referências bibliográficas nos trabalhos obtidos visando obter mais informações de maneira minuciosa sobre o tema investigado.

[1] https://www.semanticscholar.org
[2] https://scholar.google.com.br
[3] https://ieeexplore.ieee.org

2.3 TRIAGEM

Nesta Seção, são apresentadas a triagem das três perguntas descritas na Seção 2.2. Deste modo, para cada pergunta foi definida uma seção indicando os critérios de inclusão, as publicações encontradas e o seu processo de inclusão. De forma geral, para todas as QPs, foram considerados os mesmos critérios de exclusão:

- Publicações duplicadas;
- Publicações sem validação da proposta;
- Publicações não gratuítas e acessíveis de forma online.

2.3.1 Triagem QP1

Para a QP1, foram considerados os seguintes critérios de inclusão:

- Revisões sistemáticas;
- Somente artigos em periódicos a partir de 2014;
- Ordenados por relevância;
- Trabalhos que tratavam exclusivamente 'modelos de qualidade de software' contemplando características de qualidade não limitadas a um unico aspecto, como por exemplo: confiabilidade;
- Maior número de citações;

Os resultados retornados para a QP1 foram 52 (SemanticScholar) e 262 no (Google Scholar) e inclusos artigos de 2010, 2011 e 2013 que atendiam aos critérios de inclusão, com exceção do período. Deste modo, foram selecionados os seguintes artigos, conforme a Tabela 10.

Tabela 10 – Resultados da busca - QP1

No.	Artigo	Ano SS	GS	Observações	Fonte
1.	A Review of Software Quality Models for the Evaluation of Software Products	2014 23	60	Avalia 19 modelos de qualidade de software, categorizando-os em 2 tipos: Básicos e Tailored e comparando 29 características de qualidade	(MIGUEL; MAURICIO; RODRÍGUEZ, 2014)
2.	A comparative study of software quality models	2014 0	9	Avalia 16 modelos de qualidade de software, comparando 28 características	(SUMAN; ROHTAK, 2014)
3.	Comparative Study of Software Quality Models	2013 0	4	Compara 9 modelos de qualidade e 28 características	(YOUNESS et al., 2013)
4.	Software quality models: A comparative study	2011 0	29	Comparando 5 modelos basicos de qualidade e 17 características	(AL-BADAREEN et al., 2011)
5.	Quality models in software engineering literature: an analytical and comparative study	2010 0	127	Comparando 5 modelos de qualidade e 8 características	(AL-QUTAISH, 2010)

Utilizando a técnica apresentada por (WOHLIN, 2014) nos trabalhos selecionados, foram encontrados mais artigos conforme a Tabelas 11.

Tabela 11 – Resultados snowballing - QP1

No.	Artigo	Ano Citações	Fonte
1.	Factors in Software Quality: Preliminary Handbook on Software Quality for an Acquisiton Manager	1977 197	(MCCALL, 1977)
2.	Characteristics of Software Quality	1978 833	(BOEHM, 1978)
3.	ISO-9126	1991 2470	(ISO/IEC 9126, 2001)
4.	Practical Software Metrics for Project Management and Process Improvement	1992 951	(GRADY, 1992)
5.	A Model for Software Product Quality	1995 625	(DROMEY, 1995)
6.	A software quality model and metrics for risk assessment	1996 16	(HYATT; ROSENBERG, 1996)
7.	A hierarchical model for object-oriented design quality assessment	2002 1034	(BANSIYA; DAVIS, 2002)
8.	Quality Attributes for COTS Components	2002 177	(BERTOA; VALLECILLO, 2002)
9.	GEQUAMO—A Generic, Multilayered, Customisable, Software Quality Model	2003 42	(GEORGIADOU, 2003)
10.	Towards a software component quality model	2005 26	(ALVARO; ALMEIDA; MEIRA, 2005)
11.	A new software quality model for evaluating COTS components	2006 117	(RAWASHDEH; MATALKAH, 2006)
12.	ISO-25010	2008 1032	(ISO/IEC 25010, 2011)
13.	DEQUALITE: building design-based software quality models	2008 11	(KHOMH; GUÉHÉNEUC, 2008)
14.	The Squale Model – A Practice -Based Industrial Quality Model	2009 38	(MORDAL-MANET et al., 2009)
15.	Quality Models for Free/Libre Open Source Software Towards the “Silver Bullet”?	2010 2	(GLOTT et al., 2010)
16.	The quamoco product quality modelling and assessment approach	2012 91	(WAGNER et al., 2012)
17.	The SQALE Method Definition Document	2012 92	(LETOUZEY, 2012)
18.	MIDAS: a design quality assessment method for industrial software	2013 21	(SAMARTHYAM et al., 2013)
19.	Measuring, Assessing and Improving Software Quality based on Object-Oriented Design Principles	2016 8	(PLÖSCH et al., 2016)

2.3.2 Triagem QP2

Para a QP2, foram considerados os seguintes critérios de inclusão:

- Artigos a partir de 2014;
- Ordenados por relevância;
- Trabalhos que tratavam exclusivamente 'software metrics thresholds' observando atributos de qualidade internos;
- No mínimo 5 citações;

Os resultados retornados para a QP2 foram seis (IEEXplore), onde somente dois atendiam aos critérios de inclusão, para o Google Scholar foram encontrados trinta e seis artigos, porém, somente dois foram selecionados. Deste modo, foram selecionados os seguintes artigos, conforme a Tabela 12:

Tabela 12 – Resultados da busca - QP2

No.	Artigo	Ano	Cit.	Observações	Fonte
1	Extracting relative thresholds for source code metrics	2014	21	Propõe o conceito de limites relativos para a avaliação de métricas seguindo distribuições heavy-tailed.	(OLIVEIRA; VALENTE; LIMA, 2014)
2	RTTool: A Tool for Extracting Relative Thresholds for Source Code Metrics	2014	6	Propõe uma ferramenta para a coleta de limites relativos proposto no trabalho do item anterior.	(OLIVEIRA et al., 2014)
3	User-perceived source code quality estimation based on static analysis metrics	2016	10	Relaciona qualidade com méticas de software considerando que a popularidade de componentes de software predizem a qualidade de maneira mais simples que o cálculo de limites;	(PAPAMICHAIL; DIAMANTOPOULOS; SYMEONIDIS, 2016)
4	Thresholds for size and complexity metrics: A case study from the perspective of defect density	2016	5	Declara que a pulverização de código em classes menores e menos complexas podem ser contraprodutivo, considerando limites com relação a tamanho, complexidade e sua correlação com densidade de defeitos.	(YAMASHITA et al., 2016)

Utilizando a técnica apresentada por (WOHLIN, 2014) nos trabalhos selecionados, foram encontrados mais artigos conforme a Tabelas 13.

Tabela 13 – Resultados snowballing - QP2

No.	Artigo	Ano	Cit.	Fonte
1.	Validating Metric Thresholds with Developers: An Early Result	2015	0	(OLIVEIRA et al., 2015)
2.	A Catalogue of Thresholds for Object-Oriented Software Metrics	2015	23	(FILÓ; BIGONHA, 2015)
3.	Identifying thresholds for object-oriented software metrics	2012	120	(FERREIRA et al., 2012)
4.	Calculation and optimization of thresholds for sets of software metrics	2011	51	(HERBOLD; GRABOWSKI; WAACK, 2011)
5.	A quantitative investigation of the acceptable risk levels of object-oriented metrics in open-source systems	2010	107	(SHATNAWI, 2010)
6.	Deriving metric thresholds from benchmark data	2010	155	(ALVES; YPMA; VISSER, 2010)
7.	Finding software metrics threshold values using ROC curves	2010	93	(SHATNAWI et al., 2010)
8.	An investigation of ck metrics thresholds	2006	11	(SHATNAWI, 2006)
9.	Thresholds for object-oriented measures	2000	87	(BENLARBI et al., 2000)
10.	Applying and interpreting object oriented metrics	1998	139	(ROSENBERG, 1998)

2.3.3 Triagem QP3

Para a QP3, foram considerados os seguintes critérios de inclusão:

- Somente artigos a partir de 2016;
- Ordenados por relevância;
- Trabalhos que tratavam exclusivamente 'Code smells; Prioritize; Refactoring' observando priorizações em correções de não-conformidades;
- No mínimo 10 citações;

Os resultados retornados para a QP3 foram 193 (Google Scholar), sendo selecionados somente quatro trabalhos. Observando os critérios de inclusão a Tabela 14 destaca os seguintes artigos:

Tabela 14 – Resultados da busca - QP3

No.	Artigo	Ano	Cit.	Observações	Fonte
1.	An approach to prioritize code smells for refactoring	2016	46	Utiliza índices do mercado financeiro como apoio na priorização de correções de não-conformidades.	(VIDAL; MARCOS; DÍAZ-PACE, 2016)
2.	Context-based code smells prioritization for prefactoring	2016	10	Propõe a priorização baseada no contexto do desenvolvedor contidas em um lista de não-conformidades avaliadas por um índice.	(SAE-LIM; HAYASHI; SAEKI, 2016)
3.	Continuous quality assessment with inCode	2017	17	Destaca um ferramenta que oferece avaliação contínua de código permitindo ao desenvolvedor priorizar a inspeção e as correções de fragmentos de design mais críticos	(GANEA; VEREBI; MARINESCU, 2017)
4.	Identifying and quantifying architectural debt	2016	33	Define uma priorização baseado no débito técnico e no mapa de esforço relativo à correção	(XIAO et al., 2016)

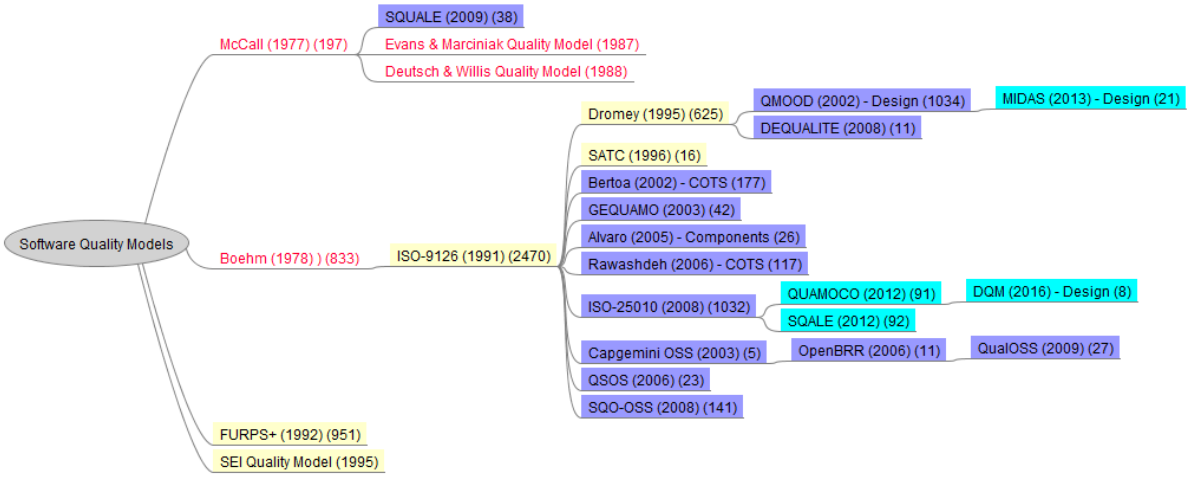
2.4 ANÁLISE E DISCUSSÃO DOS RESULTADOS

Nesta seção serão respondidas as três questões de pesquisas propostas no SMS.

2.4.1 Resultados da QP1

Para a responder o (Cenário 1 - QP1) Observando o estado da arte dos modelos de avaliação de qualidade de software, quais os modelos mais relevantes, mais utilizados e quais as características que podem suportar uma tomada de decisão? Foram mapeados 26 modelos de avaliação de qualidade de software, conforme a Figura 4.

Figura 4 – Modelos para avaliação de qualidade de software



Nesta Figura, podemos observar os modelos associados à quantidade de citações em seus trabalhos de referências, bem como, alguns pontos gerais importantes, sendo eles:

- A maioria dos modelos surgiram com base no trabalho proposto por (BOEHM, 1978);
- A maioria dos modelos propostos surgiram na década de 2000, sendo encontradas 13 proposições;
- A maioria dos modelos herdam características da (ISO/IEC 9126, 2001), totalizando 10 proposições;
- Observando os modelos de qualidade com atenção no Design, o modelo QMOOD (2002) possui o maior número de citações sendo 1034;
- Foram encontrados somente 3 proposições de modelos tendo como base a (ISO/IEC 25010, 2011).

Analisando a quantidade de caraterísticas por modelo de qualidade, a Tabela 15 apresenta os seguintes resultados. Como é possível ver, os modelos de referência como as ISOs

Tabela 15 – Modelos de qualidade de software - qtde de caracteristicas - QP1

Id	Modelo de qualidade	Características
1.	Boehm (1978)	7
2.	McCall (1977)	11
3.	Evans e Marciniak (1987)	11
4.	Deutsch e Willis (1988)	11
5.	FURPS+ (1992)	9
6.	Dromey (1995)	7
7.	SEI Quality Model (1995)	4
8.	SATC (1996)	3
9.	ISO-9126 (2001)	12
10.	QMOOD (2002)	6
11.	Bertoa (2002)	6
12.	GEQUAMO (2003)	2
13.	CapGemini OSMM (2003)	5
14.	Alvaro (2005)	6
15.	Rawashdeh (2006)	6
16.	OpenBRR (2006)	6
17.	QSOS (2006)	12
18.	DEQUALITE (2008)	7
19.	ISO-25010 (2008)	12
20.	SQO-OSS (2008)	5
21.	QualOSS (2009)	5
22.	SQUALE (2009)	6
23.	Quamoco (2012)	8
24.	MIDAS (2013)	6
25.	SQALE (2016)	9
26.	DQM (2016)	8

1996, 25010 e os modelos básicos propostos entre as décadas de 70-80, são os que possuem uma maior quantidade de características.

Para a seleção das características de qualidade, dentro destes modelos foram comparadas e consideradas vinte e uma características tendo como critério a referência de mais de um modelo, abaixo seguem a descrição e o seu propósito conforme a (ISO/IEC 25010, 2011):

- 1. **Confiabilidade** - O grau no qual um produto de software pode manter um nível de desempenho especificado sob determinadas circunstâncias. A habilidade da unidade funcional executar a função solicitada.

- 2. **Manutenibilidade** - O grau no qual um produto de software pode ser modi-

ficado. Incuindo correções, melhorias ou adptações a mudanças no ambiente e em requisitos e especficações funcionais.

- 3. **Eficiencia/Eficiência de desempenho** - Um conjunto de atributos que influenciam o relacionamento entre o nível de desempenho do software e a quantidade de recursos utilizados, sob condições estabelecidas.
- 4. **Portabilidade/Transferibilidade** - Até que ponto um software pode ser transferido de um ambiente a outro.
- 5. **Usabilidade** - O grau no qual os usuários podem alcançar os objetivos no uso com efetividade, eficiencia e satisifação.
- 6. **Funcionalidade/Adequação funcional** - Fornece funções aderentes às necessidades dentro das condições especificadas para o software.
- 7. **Segurança** - Proteção de partes do sistema contra acesso não-autorizado, malicioso, uso, modificação, destruição ou divulgação.
- 8. **Flexibilidade na evolução** - Características que permitem incorporar mudanças no projeto. A habilidade do projeto em ser adaptado para se adequar às mudanças.
- 9. **Reusabilidade** - O grau no qual é permitido o reuso do projeto ou de partes em um novo problema com esforço reduzido.
- 10. **Entendimento do código** - Permite o software ser facilmente compreendido e assimilado. Relacionado diretamente a complexidade do projeto.
- 11. **Testabilidade** - O grau no qual é permitido sob certas condições o software ser validado.
- 12. **Interoperabilidade/Compatibilidade** - A habilidade de dois ou mais componentes de software trocarem informações ou executarem funções compartilhando o mesmo ambiente de hardware/software.
- 13. **Extensibilidade** - Facilidade na incoroporação de novos requisitos no software.
- 14. **Proteção / Integridade** - Define os níveis aceitáveis de risco em causar danos a pessoas, negocios, dados, software, propriedade ou ambiente.
- 15. **Efetividade** - A precisão e a completude no uso do software pelos usuários. Alcançar o comportamento e o resultado esperado.
- 16. **Documentação** - A capacidade de descrever do que deve ser construído.
- 17. **Corretude** - O grau no qual o software atende à sua especificação.

- 18. **Gerenciabilidade** - A capacidade de gerenciamento está relacionada desenvolver e refinar estimativas de esforço e prazos para o projeto como um todo e com a coleta de quaisquer dados que possam ser necessários para tais estimativas.
- 19. **Disponibilidade** - O grau em que um componente de software está operacional e disponível quando necessário para uso.
- 20. **Flexibilidade no uso** - O grau em que o produto é utilizável em todos os contextos potenciais de uso. Ex. Web, smartphone, tablet.
- 21. **Verificabilidade** - Confirmação, através do fornecimento de evidências objetivas se os requisitos especificados tem sido atendidos.

Com base nas ocorrências sobre as caracteristicas mapeadas entres os modelos, a Tabela 16 apresenta os resultados ordenados de forma decrescente, indicando primeiramente aquelas com um maior percentual de ocorrência.

Tabela 16 – Resultados caracteristicas - QP1

1.	Confiabilidade	76%
2.	Manutenibilidade	72%
3.	Eficiencia/Eficiência de desempenho	68%
4.	Portabilidade/Transferibilidade	60%
5.	Usabilidade	60%
6.	Funcionalidade/Aptidão funcional	56%
7.	Segurança	48%
8.	Flexibilidade na evolução	36%
9.	Reusabilidade	32%
10.	Entendimento do código	24%
11.	Testabilidade	20%
12.	Interoperabilidade/Compatibilidade	20%
13.	Extensibilidade	20%
14.	Proteção / Integridade	20%
15.	Efetividade	16%
16.	Documentação	16%
17.	Corretude	8%
18.	Gerenciabilidade	8%
19.	Disponibilidade	8%
20.	Flexbilidade no uso	8%
21.	Verificabilidade	8%

Até este ponto, conseguimos verificar que os modelos básicos e as ISOs são os que mais possuem características, são mais referenciados e utilizados. Que dentre as 5 primeiras ca-

racterísticas mais referenciados pelos modelos estão: a Confiabilidade, Manutenibilidade, Eficiência/Desempenho, Portabilidade e a Usabilidade.

Verificando essas características junto a um mapeamento sistemático sobre aspectos de valor no desenvolvimento de software elaborado por (KHURUM; GORSCHEK; WILSON, 2013b), verificamos que dos seis itens melhores posicionados na Tabela 15, cinco itens aparecem como importantes na perspectiva do cliente com percepção de valor.

Sendo assim, respondemos a QP1 declarando que os modelos mais relevantes são os modelos que originaram os maiores número de referências e características, sendo eles as ISOs e o modelos básicos (70-80). As seis primeiras características classificadas pela recorrência em 25 modelos estudados, aparecem como aspectos que contribuem para a percepção de valor na perspectiva do cliente, com um relação de 83,33%, o que pode sugerí-los como possíveis candidatos para uma tomada de decisão.

2.4.2 Resultados da QP2

Para a responder o (Cenário 2 - QP2) Considerando métricas como medidas de avaliação de qualidade interna do software, quais os limites definidos no estado da arte que possam direcionar ações de melhoria de código em projetos de software?

Analisando as Tabelas 12 e 13, vimos abordagens que se iniciaram a partir de (ROSENBERG, 1998) com definições de limites a partir da análise de um conjunto de projetos baseados na experiência dos autores. Observando os estudos seguintes, as abordagens também possuiam a mesma dinâmica: avaliando um conjunto de projetos sendo de código aberto ou industrial de tamanho e complexidade distintas, associados a avaliações por meio de alguma análise estatística.

Ainda neste contexto, foram consideradas as opiniões de especialistas, a análise de correlação entre limites e defeitos, e a utilização de algoritmos de aprendizagem de máquina.

Foi também avaliada uma ferramenta (RTTool) com o objetivo de definir limites com base em um conjunto de amostras, onde cada tipo de métrica possui uma distribuição estatística associada. Conforme Figura 5, é possível identificar os limites para dezessete métricas,

Durante a análise dos artigos foi elaborada um mapeamento com as métricas definidas por cada trabalho, objetivando apresentar as diferenças entre os estudos, dado que foram utlizadas definições com base em experiências e outras, com abordagens estatísticas, conforme a Tabela 17. Ainda neste sentido, foram apresentadas somente métricas que possuiam mais de um estudo relacionado. Sendo assim, foram checadas as seguintes métricas:

- CA (Robert C. Martin) - Acoplamento Aferente, um número de classes fora de um pacote que dependem de classes dentro do pacote;

Figura 5 – Limites identificados (FILÓ; BIGONHA, 2015)

Metric	Good/Common	Regular/Casual	Bad/Uncommon
CA	$m \leq 7$	$7 < m \leq 39$	$m > 39$
CE	$m \leq 6$	$6 < m \leq 16$	$m > 16$
DIT	$m \leq 2$	$2 < m \leq 4$	$m > 4$
LCOM	$m \leq 0,167$	$0,167 < m \leq 0,725$	$m > 0,725$
MLOC	$m \leq 10$	$10 < m \leq 30$	$m > 30$
NBD	$m \leq 1$	$1 < m \leq 3$	$m > 3$
NOC	$m \leq 11$	$11 < m \leq 28$	$m > 28$
NOF	$m \leq 3$	$3 < m \leq 8$	$m > 8$
NOM	$m \leq 6$	$6 < m \leq 14$	$m > 14$
NORM	$m \leq 2$	$2 < m \leq 4$	$m > 4$
NSC	$m \leq 1$	$1 < m \leq 3$	$m > 3$
NSF	$m \leq 1$	$1 < m \leq 5$	$m > 5$
NSM	$m \leq 1$	$1 < m \leq 3$	$m > 3$
PAR	$m \leq 2$	$2 < m \leq 4$	$m > 4$
SIX	$m \leq 0,019$	$0,019 < m \leq 1,333$	$m > 1,333$
VG	$m \leq 2$	$2 < m \leq 4$	$m > 4$
WMC	$m \leq 11$	$11 < m \leq 34$	$m > 34$

- CBO (CK) - Acoplamento entre objetos, mede o número de acoplamento não herdado com outras classes, onde o acoplamento é a medida de interação entre dois objetos;
- CE (Robert Martin) - Acoplamento Eferente, um número de classes dentro de um pacote que dependem de classes foram do pacote;
- DIT (CK) - Profundidade da árvore de herança, verifica a distância do comprimento das classes-filhas para a classe raiz;
- LCOM (CK) Ausência de coesão entre métodos, conta o número de métodos que compartilham atributos da classe;
- LOC - Linhas de Còdigo, tamanho da classe em linhas de código;
- MLOC - Tamanho do método em linhas de código;
- NOC (CK) - Número de classes filhas, número de subclasses imediatas;
- NOM - Número de métodos;
- NORM (Robert C. Martin) - Números de operações sobrescritas
- NSM - Número de métodos estáticos;
- RFC (CK) - Resposta para a classe, refere-se a quantidade de acessos que uma operação necessita para a realização de uma ação;
- WMC (CK) - Métodos ponderados por classe, representa a soma das complexidades dos métodos de uma classe;

Dentre as métricas observadas os estudos em sua maioria avaliaram limites para as seis métricas propostas por (CHIDAMBER; KEMERER, 1994). Verificando as diferenças entre limites conforme a Tabela 17, vemos que existem variações significativas como por

exemplo: o estudo 1 que apresenta o valor de CA/F-OUT de 39 e, o estudo 7, apresenta o valor de 10; outro exemplo seria o de LOC de 222 para o estudo 2 e, 500 para o estudo 5.

Tabela 17 – Métricas de Software - Limites - QP2

Métricas	1	2	3	4	5	6	7	8	9	10	11
CA/F-IN	39			20			10				
CBO					5	9		11	10	8	5
CE/F-OUT	16	15									
DIT	4		2				1		1		
LCOM	0,7	33		20				18			
LOC		222	178		500						
MLOC	30							30			
NOC	28									2	
NOM	14	16	22	40	20	19					
NORM	4				3						
NSM	3				4						
RFC		49			100	40		38	55	25	100
WMC	34	32			100	20		18	33	31	100

Um ponto importante foi a dificuldade em se aplicar as abordagens em âmbito industrial dada a inexistência de ferramentas que ofereça um suporte para tal atividade.

Sendo assim, respondemos a QP2 declarando que não é possível selecionar uma única metodologia como a mais adequada para a definição de limites, somente pela quantidade de referências ou pela cobertura de métricas avaliadas. O que pode se concluir é que dada a quantidade de abordagens na definição de limites de métricas, cada uma propondo uma abordagem específica, a atividade de definição de limites por meio da análise estática se torna algo passível de mais experimentação e adequação, conforme o contexto do projeto a ser avaliado.

Deve se avaliar o quanto a métrica está relacionada a outros fatores como: quantidade de defeitos, produtividade do time, a natureza do projeto, opinião de especialistas, número de defeitos, dentre outros.

2.4.3 Resultados da QP3

Para a responder o (Cenário 3 - QP3) Dado o volume de não-conformidades em projetos de software, quais as abordagens existentes predizem ou direcionam a priorização de correções ou refatoramento?

Quando observamos os artigos encontrados na QP3, identificamos quatro abordagens distintas onde criticamos os seguintes pontos:

- Utiliza índices do mercado financeiro como apoio na priorização de correções de não-conformidades - Neste sentido é utilizado o índice de volatilidade histórica, sendo a abordagem aplicada em um único tipo de aplicação.

- Propõe a priorização baseada no conexto do desenvolvedor contidas em lista de não-conformidades, avaliadas por um Ìndice de Relevância de Contexto, sendo calculado a partir a média ponderada de módulos impactados pela não-conformidade. Não foi encontrado no trabalho a indicação sobre a necessidade de uma correlação entre módulos e não-conformidade para a execução da abordagem.

- Destaca um ferramenta que oferece avaliação contínua de código permitindo ao desenvolvedor priorizar a inspeção e as correções de fragmentos de design mais críticos. Neste sentido, o desenvolvedor fica com a responsabilidade na priorização das correções.

- Define uma priorização baseado no débito técnico e no mapa de esforço relativo à correção. Neste contexto é necessário que haja não-conformidades de formas variadas dado que o esforço é variável conforme o tipo de incidente.

Sendo assim, respondemos a QP3 encontrando quatro abordagens que possivelmente atingem o objetivo de predição em correções de defeitos, porém, elas não consideram métricas de código na composição dos índices de priorização propostos. Durante as pesquisas foram encontradas abordagens sobre priorização na correção de bugs, porém, o propósito desta questão seria a priorização de possíveis não-conformidades, dado fatores como complexidade e acoplamento.

2.5 ANÁLISE GERAL

Por fim, a atividade de Revisão da Literatura esclareceu as três questões derivadas das hipóteses descritas na Seção 1.1 esclarecendo modelos de avaliação de qualidade de software, limites para métricas de software e técnicas de priorização de correção de não-conformidades.

O que pode ser concluido nesta pesquisa é que na QP1 podemos considerar que os modelos existentes conseguem apoiar uma tomada de decisão, porém, para as outras duas QPs, vimos que existe uma variedade nas abordagens tanto na definição de limites quanto na forma de se priorizar correções, trazendo uma visão não-conclusiva para estes dois questionamentos.

Enfim, com os resultados dos levantamentos deste Capítulo, foi possível obter como contribuição ao autor, outros formatos sobre a apresentação de informações relativas às métricas, abrangência dos modelos de qualidade de software e a possíveis não-conformidades em código-fonte em niveis mais estratégicos.

2.6 SUMÁRIO

Este Capítulo apresentou a abordagem para a obtenção de uma contextualização inicial sobre o tema proposto por esta tese, utilizando um critério de seleção explícito, onde extraiu os principais pontos a serem considerados quanto à avaliação de métricas, modelos e custos relacionados à construção e manutenção em projetos de software. No Capítulo 3 serão verificados os seguintes pontos:

- A discussão sobre temas que contribuiram com esta proposta de tese, utilizando as publicações extraídas neste Capítulo associadas as fases do método proposto.
- A análise de outras publicações relativas à proposta de análise estática de código, juntamente com uma discussão sobre a contribuição de cada seção e sua importância para o trabalho como um todo.

3 FUNDAMENTAÇÃO TÉORICA

O objetivo deste capítulo consiste em apresentar os principais conceitos que fundamentaram a construção desta tese, utilizando como guia a Figura 2. Como tais itens são historicamente extensos, o objetivo principal neste Capítulo está em avaliar a aderência de tais técnicas no método proposto. Considerando as 3 fases descritas na proposta, **Coletar**, **Avaliar** e **Direcionar**, a organização deste Capítulo está disposta da seguinte forma:

- **Coletar**, na Seção 3.1 apresentamos um breve histórico sobre a prática de mineração de código-fonte em repositórios de código e controle de versão, e na Seção 3.2, verificamos o estado atual sobre pesquisas relativas a métricas de código.
- **Avaliar**, na Seção 3.3, são discutidos os modelos de avaliação de qualidade de software, com foco em qualidade a nível de design, e na Seção 3.4, será analisado como o monitoramento contínuo é utilizado na coleta de dados para a análise e aferição de métricas em projetos.
- **Direcionar**, a Seção 3.5 apresenta porque a ausência de controle de qualidade em projetos de software se tornam débitos técnicos a serem pagos posteriormente.

3.1 REPOSITÓRIOS DE CÓDIGO E CONTROLE DE VERSÃO

Históricamente, o uso de Repositório de Código e Controle de Versão (RCCV) se consolidou como um fator essencial para o desenvolvimento de software de forma distribuída e em equipe efetivamente a partir de 1972. Segundo Koc, Tansel e Bicer (2012), o entendimento sobre RCCV não deveria ser simplesmente sobre mudanças, mas também sobre entender o racional por trás de tais mudanças, onde frequentemente é interessante entender não só a evolução dos dados, mas também a razão por trás de tal evolução.

Neste sentido, (HASSAN, 2008) define a Mineração em Repositórios de Software como um campo de pesquisa que posiciona repositórios estáticos somente guardando arquivos, para, uma posição ativa podendo guiar decisões em projetos de software atuais. Como por exemplo, dados de código-fonte armazenados em repositório poderiam ser cruzados com dados contidos em repositórios de bugs permitindo avaliar a quantidade de não-conformidades ocorridas em determinados objetos do projeto, considerando suas características (linhas de código, complexidade, operações, dentre outros).

Inicialmente, foram analisadas quais práticas, técnicas e dinâmicas de mineração de código em repositórios atenderiam o propósito de geração e criação de indicadores de qualidade. No entanto o que foi verificado é que a mineração em RCCV abrange diversificados direcionamentos de pesquisa além de simplesmente a guarda, busca e compartilhamento

de código-fonte. Deste modo, as verificações identificaram algumas técnicas e abordagens que contribuiram para a fundamentação desta tese, sendo elas:

- **A análise de frequência de refatoramento, granularidade e escopo em repositórios**. Esta verificação trouxe o entendimento de que são mais frequentes refatoramento em um escopo local, normalmente dentro do corpo das operações sem a alteração das assinaturas de acesso, por outro lado, o refatoramentos em escopo global são menos frequentes e mais difíceis de se controlar e preservar o comportamento;

- **A detecção de oportunidades de refatoramento**. Neste sentido foi verificada a falta da análise de impacto em tais sugestões com relação a preservação do comportamento, se limitando à visualização das oportunidades e estruturas complexas do projeto.

- **A análise sobre o comportamento dos desenvolvedores com relação a hábitos de refatoramento de código**. O que pode ser verificado nesta abordagem é que mensagens de commit não são confiáveis como direcionadores para atividades de refatoramento, e que refatoramentos em baixo e médio nível são bem mais frequentes, como também, aparentemente as ferramentas atuais são sub-utilizadas.

- **Em que momento do projeto existe uma maior incidência de code smells (FOWLER et al., 1999)**. Foram constatados que para novos projetos, o surgimento de code smells ocorrem, em sua maioria, no ínicio da codificação, contradizendo a hipótese em que code smells surgem somente com a evolução do projeto. Além disso, atividades de refatoramento geram a aparição de novos smells, denotando a ausência de ferramentas que avaliem os impactos em tais atividades. E por fim, desenvolvedores com alta carga de trabalho são mais suscetíveis a introdução de smells do que desenvolvedores iniciantes.

- **O entendimento do código de acordo com a evolução do projeto**. Neste contexto, foi verificado que o surgimento de smells perduram durante todo o projeto e que o desaparecimento de tais smells se dão mais por atividades de manutenção e melhorias do que por atividades de refatoramento de fato.

- **A efetividade da análise estática na avaliação de falhas e predição de refatoramento**. Como conclusão foi avaliado que ferramentas disponíveis, levantam uma grande quantidade de falsos positivos, nâo existindo uma detecção consistente de falhas por meio destas.

- **O surgimento de débito técnico por meio de métricas de modularidade de acordo com a evolução do projeto**. Conclui-se que é possível a utilização

de métricas de modularidade mais simples desprezando o registro de commits, algo eventualmente indisponível para alguns projetos.

- **O impacto do refatoramento na qualidade do software e suas evoluções**. Dentro do trabalho proposto foi verificado que nem todas as atividades de refatoramento se traduzem em melhoria de qualidade, sendo possível até uma redução de tal qualidade considerando diferentes aspectos, como por exemplo, refatoramentos para melhoria de segurança podem impactar na reusabilidade, o que deixa a cargo do desenvolvedor avaliar impactos e *trade-offs*.

Neste contexto, foi avaliado um conjunto não extenso de artigos com base em uma pesquisa exploratória, no qual são citados o uso de mineração de código, categorizados conforme a Tabela 18.

Tabela 18 – Mineração em repositórios de código

Categoria	Trabalhos Relacionados
Análise de refatoramento	(SOARES et al., 2011) (GÖG; WEISSGERBER, 2005) (MURPHY-HILL; PARNIN; BLACK, 2012)
Evolução de *code smells*	(CHATZIGEORGIOU; MANAKOS, 2014) (TUFANO et al., 2015)
Efetividade na análise estática	(BUSE; PL.; R., 2010) (WEDYAN; ALRMUNY; BIEMAN, 2009)
Análise de métricas e qualidade	(LI et al., 2014) (SHATNAWI; LI, 2011)
Visualização e evolução de software	(WETTEL, 2010) (CHAIKALIS et al., 2014)

Embora industrialmente RCCVs sejam considerados nada mais que locais para compartilhamento de código-fonte, os estudos relacionados apresentaram muito mais que o objetivo proposto, direcionando os caminhos para a proposição de coleta de dados deste trabalho.

Neste sentido, é importante entender que a forma na qual os objetos são versionados, bem como a definição de acessos, podem se tornar uma ameaça à verificação e obtenção de dados de forma histórica em RCCV. Porém, é possível considerar a validade desta abordagem, caso sejam consideradas algumas premissas tornando possível direcionar o uso de mineração de código um insumo para a execução das atividades previstas no guia desta tese.

Industrialmente foram verificadas que práticas como compartilhamento de um único usuário entre membros do projeto era algo bastante comum, devido ao custo de licenciamento com base em número de usuários, bem como, a remoção e subida de arquivos

desconsiderando histórico do controle de versão da ferramenta em uso, o que poderia inviabilizar a avaliação de desempenho dos membros dos times por exemplo.

Concluindo esta Seção, os temas de pesquisa utilizando RCCV trouxeram um apoio substancial à esta pesquisa, dado que, forneceram direcionamentos com relação à periodicidade de busca e avaliação do código-fonte, evolução de code smells, entendimento dos dados obtidos em termos de manutenibilidade, predição de custos e visualização de resultados, conforme o estado atual do projeto.

Apresentando aos envolvidos nos projetos os levantamentos sobre as diversas abordagens de pesquisa no uso de RCCV, bem como, o real valor dos dados históricos contidos em tais repositórios, foi possível obter uma maior aceitação sobre a necessidade de acessos individuais em RCCV, tornando-se possível direcionar um modelo mais adequado para o versionamento dos projetos. Na Seção 3.2, serão apresentadas de forma sucinta uma visão sobre métricas de software, que conforme a Figura 2 consiste em um passo de suporte à coleta de código.

3.2 MÉTRICAS DE SOFTWARE

As métricas de software que surgiram por volta da década de 70, foram definidas para se tornarem um método de quantificação de atributos em processos, produtos e projetos de software. Segundo Daskalantonakis (1992), métrica pode ser entendida como um valor aplicado a um atributo de um produto de software, deste modo, tais valores podem ser comparados entre si com padrões aplicáveis em organizações. Esta seção não pretende ser uma revisão exaustiva sobre o tema, dado que, o objetivo seria esclarecer quais as métricas são tidas como as mais citadas durante a elaboração desta tese.

Métricas podem ser divididas em duas principais épocas: antes de 1992 onde o foco principal eram medidas de tamanho e complexidade (MCCABE, 1976) e, Halstead Software Science Metrics (HALSTEAD et al., 1977), largamente conhecidas e estudadas por vários anos. Medidas de tamanho como Linhas de Código (LOC) comumente referenciadas tem sido utilizadas desde o final da década de 60, conforme (FENTON; NEIL, 2000), e, McCabe e Halstead no final da década de 70. Tais métricas foram estudadas dos anos 70 ao início dos ano 90, quando sugiram as métricas orientadas a objetos (OO) propostas por (CHIDAMBER; KEMERER, 1994) e (LI; HENRY, 1993).

Devido a sua aplicação em uma larga variedade de aplicações e experimentos relacionados a código-fonte de forma geral, possuem um papel importante na predição de falhas, complexidade de testes, atividades de refatoramento, dentre outros. Com o objetivo de se aperfeiçoar as diferentes necessidades de medidas, mais e mais métricas são elaboradas, estudadas e validadas através dos anos, com novas métricas sendo propostas de forma constante.

Considerando o ritmo na elaboração de novas métricas de software juntamente com o corpo de pesquisa já produzido, torna-se difícil para pesquisadores e profissionais acom-

panharem o estado atual e as tendências em métricas de código-fonte. Neste sentido, podemos citar os seguintes trabalhos em termos de análise do estado da pesquisa recente que visam auxiliar no mapeamento e na tomada de decisão, sendo eles:

- O levantamento elaborado por (NUÑEZ-VARELA et al., 2017), analisou 226 estudos publicados entre os anos de 2010 a 2015, encontrando aproximadamente 300 métricas.

- A investigação apresentada por (SANTOS et al., 2016), onde existem por volta de 79 trabalhos relacionados a qualidade interna de software orientado a objetos, ou seja, passível de serem medidos de forma estática. Neste sentido, foram encontradas 265 métricas, porém, somente 15 métricas foram consideradas como as principais utilizadas para avaliação de qualidade interna;

- A catalogação e categorização elaborada por (SARAIVA et al., 2015), constando por volta de 600 métricas permitindo facilitar a seleção e a tomada de decisão por parte de acadêmicos e profissionais da indústria;

- A análise de (TIMÓTEO et al., 2008) sobre o estado da arte em métricas de software, verificando sua evolução na área e informando as métricas mais referenciadas, sendo estas candidatas para o mapeamento em atributos de qualidade de código.

Analisando os quatro trabalhos citados, procuramos identificar quais as métricas foram mais estudadas ou recomendadas em cada trabalho.

Observando o trabalho elaborado por (NUÑEZ-VARELA et al., 2017), onde podemos destacar a existências das seis métricas propostas por (CHIDAMBER; KEMERER, 1994), a Tabela 19, destaca um total de dez métricas OO com um maior número de ocorrências.

Tabela 19 – Métricas com mais alto número de ocorrências (NUÑEZ-VARELA et al., 2017)

Métrica	Ocorrências
Weighted Methods per Class (WMC)	89
Coupling Between Objects (CBO)	89
Lack of Cohesion in Methods (LCOM)	86
Depth of Inheritance Tree (DIT)	81
Lines of Code (LOC)	79
Number of Children (NOC)	77
Response for a Class (RFC)	72
Number of Methods (NOM)	57
Cyclomatic Complexity (V(G))	55
Number of Attributes (NOA)	43

Para o trabalho elaborado por (SANTOS et al., 2016) as métricas com maior número de citações são destacadas na a Tabela 20, onde também estão inclusas as seis métricas citadas no trabalho anterior.

Tabela 20 – Métricas com mais alto número de citações (SANTOS et al., 2016)

Métrica	Citações
Lack of Cohesion in Methods (LCOM)	40
Depth of Inheritance Tree (DIT)	37
Response for a Class (RFC)	33
Coupling Between Objects (CBO)	32
Number of Children (NOC)	30
Weighted Methods per Class (WMC)	28
Lines of Code (LOC)	20
Number of Methods (NOM)	18
Cyclomatic Complexity (V(G))	13
Number of Attributes (NOA)	11
Lack of Cohesion in Methods-2 (LCOM2)	10
Lack of Cohesion in Methods-4 (LCOM4)	10
Loose Class Cohesion (LCC)	10
Tight Class Cohesion (TCC)	10
Fan-out (FOUT)	10

Conforme o trabalho elaborado por (SARAIVA et al., 2015), este indica as seguintes métricas mais utilizadas em sua revisão:

- Linhas de Código (LoC);
- e Métricas de (CHIDAMBER; KEMERER, 1994).

Como resultado do trabalho proposto por (TIMÓTEO et al., 2008), as métricas mais recomendadas foram:

- Linhas de Código (LoC) para medir documentação;
- Complexidade ciclomática, para medir complexidade e manutenibilidade;
- Métricas de (CHIDAMBER; KEMERER, 1994) para medir reusabilidade, acoplamento e coesão;
- e por fim, (LORENZ; KIDD, 1994) para calcular modularidade.

A partir de 2000, foram identificados pouco trabalhos relacionados a criação de novas métricas, contudo, métricas possuem um papel crucial em modelos contínuos de avaliação de software, principalmente com a adoção de metodologias ágeis [1]. Neste contexto,

[1] http://agilemanifesto.org/

podemos entender que métricas de software ainda continuam sendo a base fundamental a avaliação de qualidade de software, e que, coletar métricas sem uma justificativa de interpretação adequada, tornam os números meramente dados sem um valor efetivo, podendo ainda a levar a tomadas de decisão equivocadas.

Ainda conforme (NUÑEZ-VARELA et al., 2017) as métricas OO são consideradas como essencial para o estudo de medições, trazendo consigo uma evolução para os paradigmas orientado a aspectos e orientado a características.

Conforme a evolução da pesquisa nessa tema, esta tese identificou que métricas associadas ao tamanho da aplicação, complexidade e acoplamento foram as mais utilizadas em técnicas de predição de manutenção. Embora haja um alto volume de métricas propostas, os trabalhos apresentaram alguns questionamentos que restringem o uso de métricas de forma geral, sendo eles:

- Dificuldade em se obter ferramentas que extraiam tais métricas;
- Problemas de validação empírica e teórica;
- Métricas não possuem um processo claro de definição;
- Métricas com o mesmo propósito;
- Definições de métricas advindas de outras métricas com composições téóricas não correlacionadas.

Na Seção 3.3, serão discutidos modelos de avaliação que permitem o uso de métricas discutidos nesta Seção, deste modo, é possível direcionar o mapeamento de métricas simples para um formato em mais alto nível.

3.3 MODELOS PARA AVALIAÇÃO DE QUALIDADE DE SOFTWARE

Considerando o aspecto intangível e a natureza abstrata do software, pesquisadores tem procurado por maneiras de caracterizar software de modo a entender e tornar os benefícios e custos mais visíveis. Esta jornada continua até hoje, porém, iniciado por dois notáveis modelos, sendo eles: (MCCALL, 1977) e (BOEHM, 1978).

No contexto desta tese, normas e modelos de qualidade de software possuem como objetivo ***direcionar atributos de qualidade mensuráveis para que de forma automatizada seja possível coletar e mapear atributos internos de qualidade por meio da análise estática de código***.

Embora tais modelos direcionem quais as características em termos de qualidade devem ser avaliadas, o desafio de se caracterizar e mapear métricas de código em itens de qualidade, se torna algo não trivial. Neste sentido, esta Seção relaciona os principais modelos e normas para a Avaliação de Qualidade de Software (AQS) que guiaram este trabalho, tomando como base o levantamento executado no Capítulo 2.

O modelo proposto por (DROMEY, 1995), empírico, fornece um processo para a construção de um mapeamento sistemático entre propriedades e atributos de qualidade, fornecendo uma ligação entre características tangíveis e menos tangíveis com relação a atributos de qualidade. Porém não existe uma indicação de como métricas simples se adequariam a cada item de qualidade proposto.

Conforme (KAN, 2014), Grady e Caswell em 1986 apresentaram uma descrição sobre o software para avaliação de métricas em um programa da Hewllet-Packard (HP), o qual incluia métricas primitivas e métricas computadas, sendo largamente utilizadas pela própria companhia. Diante de um conjunto de métricas, os autores afirmaram que embora as métricas computadas fossem largamente utilizadas pela HP, talvez as mesmas não fossem compreensíveis para cobrir outros aspectos de qualidade. Deste modo (GRADY, 1992) desenvolveu um modelo baseado em classificação de atributos de qualidade de software nomeado FURPS. Segundo (AL-QUTAISH, 2010), este modelo foi desenvolvido tendo como propósito atender somente a IBM Rational Software Company, proponente do modelo.

O modelo SQALE de acordo com (LETOUZEY, 2012), se constitui em um método para a avaliação de qualidade do software, cujo o objetivo está em se tornar aplicável para todos os entregáveis, bem como, todas as representações de software tal como modelos UML. Deste modo, uma porção deste método se aplica a avaliação do código-fonte, definindo 8 características. Cada característica é segmentada em subcaracterísticas, divididas em dois tipos: atividades de ciclo de vida, como testes unitários e de integração, otimização do uso de processadores e do código gerado; e o outro tipo, em termos de taxonomia avaliando boas e más práticas relacionadas a arquitetura e codificação. Na busca por implementações deste modelo encontramos o SonarQube[2], uma ferramenta líder de mercado em detecção de *code smells*, o Squoring [3], o NDepend [4] e o Mia-Quality [5]. Como o próprio modelo descreve, o mapeamento de métricas de código por ferramentas não é o escopo da proposta, deste modo, o SonarQube, dentre outras ferramenttas possui um modo próprio de mapeamento.

O modelo SQUALE proposto por (MORDAL-MANET et al., 2009), sendo inspirado pelo modelo FCM (Factors-Criteria-Metrics) proposto por (MCCALL, 1977), introduz um novo nível de práticas entre critérios e métricas. Neste modelo é descrita uma decomposição hierárquica dos atributos de qualidade conforme a (ISO/IEC 9126, 2001) extendendo o modelo com fórmulas para agregar e normalizar os resultados mensurados. Não possui uma ferramenta comercial, sendo uma proposta aplicada em um consórcio composto por 6 integrantes.

A definição do QMOOD, proposto por (BANSIYA; DAVIS, 2002) tem em sua dinâmica um modelo hierárquico para a avaliação de atributos de qualidade do design em alto

[2] https://www.sonarqube.org/
[3] http://www.squoring.com/
[4] http://www.ndepend.com/
[5] http://www.mia-software.com/produits/mia-quality/

nível. Neste modelo, propriedades estruturais e comportamentais de classes e objetos, bem como, seus relacionamentos são avaliados por meio de um conjunto de métricas de design orientada a objetos. Tal modelo relaciona propriedades de design tais como: encapsulamento, modularidade, acoplamento e coesão, para atributos de qualidade em alto nível tais como: reusabilidade, flexibilidade e complexidade utilizando informações empíricas.

O modelo DQM proposto por (PLÖSCH et al., 2016), direciona para uma abordagem baseada em mapeamentos de qualidade do design, verificando aspectos de design, sem considerar métricas primitivas.

Nesta Seção vimos uma diversidade de abordagens para AQS, porém, o que pôde ser identificado é que tais modelos se apresentaram sendo de forma proprietária, sem considerar métricas primitivas e não apresentando de forma clara a dinâmica de mapeamento entre métrica e propriedades do design.

3.4 MONITORAMENTO EM QUALIDADE DE SOFTWARE

O processo de monitoramento contínuo de métricas não é algo novo. Os primeiros sistemas que automatizavam a coleta de métricas surgiram quase que imediatamente após os sistemas de controle de revisão e banco de dados de gerenciamento de defeitos, serem integrados aos processos de desenvolvimento de software. Esforços iniciais concentraram em pequena escala, equipes centralizadas e métricas de produto, conforme (BASILI; ROMBACH, 1988) e (KOMI-SIRVIÖ; PARVIAINEN; RONKAINEN, 2001), com o objetivo de suportar modelos de qualidade e gestão definidos por abordagens como a Goal-Question-Metrics proposta por (BASILI; WEISS, 1984).

Trabalhos como o de (GOUSIOS; SPINELLIS, 2009) propõe uma plataforma para o monitoramento de maneira plugável, avaliando um conjunto de métricas de código-fonte, porém, sem citar uma previsão de custos relacionados aos incidentes encontrados. Em (CHAIKALIS et al., 2014) a abordagem proposta consiste em uma ferramenta para acesso ao código-fonte contido em repositório, avaliando e exibindo as métricas coletadas.

Seguindo a mesma linha, uma ferramenta para o monitoramento contínuo proposta por (KOTHAPALLI et al., 2011), utiliza como base a norma (ISO/IEC 9126, 2001), buscando como diferencial, integrar soluções de mercado disponibilizando gráficos de tendências em termos de atributos de qualidade. Conforme o trabalho apresentado por (OUHBI et al., 2014) sobre o tema monitoramento e AQS, foi possível se constatar os seguintes pontos:

- Continua sendo abordado de forma séria e crescente por pesquisadores, dado o aumento significativo no número de publicações e periódicos em conferências estáveis, conforme verificado em 57 artigos selecionados pelo estudo apresentado, onde dentre eles, 33 foram publicações em periódicos e 12 em conferências;

- A quantidade de novos métodos de avaliação, inclusive o uso de técnicas de mineração de dados tem contribuido para o aumento no interesse em AQS;
- Os mais frequentes tipos de pesquisa em AQS demonstram um numero considerável de propostas de solução, o que evidencia que pesquisas ainda estão procurando por boas abordagens como melhoria em AQS. Poucos pesquisadores escolheram avaliar técnicas de AQS existentes;
- A maioria das abordagens são baseadas em técnicas de mineração de dados, lógica fuzzy, técnicas de regressão e agrupamento;
- Código-fonte foi o principal artefato utilizado pelas abordagens de AQS, indicando que pesquisadores estão mais interessados em avaliar a qualidade de software de forma interna;
- Mais da metade dos estudos em termos de AQS baseiam-se particularmente no uso da (ISO/IEC 9126, 2001). Embora o modelo (ISO/IEC 25010, 2011) tenha surgido em 2011, dentre 15 trabalhos publicados após 2011, somente 1 citou esta última norma.

Com o processo de desenvolvimento e entrega seguindo um modelo ágil com equipes multifuncionais, bem como, a utilização de dinâmicas de integração e entregas contínuas, o monitoramento de projetos por meio de ferramentas de automação e geração de pacotes, agregaram funcionalidades tais como: testes automatizados, análise estática de código e implantação de pacotes de forma a reduzir os custos do projeto, diminuindo a necessidade do papel do integrador de software. Considerando somente o item qualidade de software, este trabalho entende que a criação de mecanismos como *plugins* agregam um valor significativo para a medição e obtenção dos dados do projeto de forma eficiente.

Até este ponto, o método proposto por este trabalho procurou entender se práticas de: coleta de dados em RCCV, extração de métricas de software, adequação de tais métricas de software em um modelo de avaliação de qualidade de alto nível e a avaliação contínua desta abordagem faria sentido. O fato é que embora tais técnicas não sejam novidade, o ponto em questão é a composição de tais técnicas em um formato claro e de fácil entendimento não somente baseado em ferramentas de software, mas estruturado de maneira a ser controlado e replicado independente de tecnologia. Na próxima e última Seção deste Capítulo, veremos o conceito de débito técnico, este ponto trás uma visão sobre o custo em se postergar ou se relegar a análise da qualidade para um segundo plano.

3.5 DÉBITO TÉCNICO

O termo Débito Técnico (DT), uma métafora definida por (CUNNINGHAM, 1993), significa problemas de qualidade no software, ou seja, quando padrões, técnicas e boas práticas não são aplicadas de forma adequada, em um certo momento essas ausências de aplicabilidade

podem retornar como incidentes de qualidade. Segundo (BARTON; STERLING, 2010), o termo foi inicialmente concebido para a implementação de software (código-fonte), porém, com o passar do tempo se estendeu para arquitetura, detalhamento técnico do projeto, documentação, requisitos e testes.

Segundo (LI; AVGERIOU; LIANG, 2015), podemos classificar DT em 10 tipos, sendo eles:

- **Requisitos**, refere-se a distância entre a especificação e o que de fato foi construído;
- **Arquitetura**, causada por decisões que podem comprometer aspectos internos de qualidade, como por exemplo, a manutenibilidade;
- **Projeto (Design)**, refere-se aos atalhos técnicos definidos no detalhamento do projeto técnico;
- **Código**, um código com escrita pobre, violando as melhores práticas e regras, Exemplo: duplicação e alta complexidade de código;
- **Teste**, refere-se aos atalhos definidos para os teste, como por exemplo, a ausência de testes unitários, de integração e de aceitação;
- **Build (Empacotamento)**, refere-se às falhas no processo de geração do pacote, gerando complexidade ou dificuldade na entrega do produto de software;
- **Documentação**, observa a insuficiência, incompletude ou desatualização da documentação em qualquer aspecto do desenvolvimento do software. Exemplos: documentação de arquitetura desatualizada ou ausência de comentário no código-fonte;
- **Infraestrutura**, compreende uma configuração de baixa qualidade com relação ao processo de desenvolvimento, tecnologias, ferramentas de apoio. Exemplo: uma má definição de ferramentas e ambientes de desenvolvimento afetam o desempenho e a qualidade dos produtos entregues pelas equipes;
- **Versionamento**, verifica problemas relacionados ao versionamento do código-fonte com branches e forks desnecessários;
- **Defeitos**, refere-se a bugs ou falhas encontradas no software.

Neste contexto, foi identificado que DT de Código foi o mais estudado, correspondendo a 40% dos artigos avaliados. Sobre os termos mais mencionados dentro do contexto de DT, *principal, juros e riscos* foram os mais recorrentes. Quando avaliados os atributos de qualidade, estes foram mapeados conforme a ISO 25010 que é uma revisão da ISO 9126, sendo estes:

- **Adequação funcional** - 2 estudos

- **Performance eficiência** - 3 estudos
- **Compatibilidade** - 3 estudos
- **Usabiiidade** - 2 estudos
- **Confiabilidade** - 5 estudos
- **Segurança** - 4 estudos
- **Manutenibilidade** - 79 estudos
- **Portabilidade** - 7 estudos

Deste modo, podemos perceber que a Manutenibilidade é o atributo de qualidade com mais estudos e mais relacionado ao tema de DT. Quando observados a forma mais recorrente sobre a verificação de DT, a Tabela 21 apresenta o seguinte resultado:

Tabela 21 – Abordagem de verificação e estudos relacionados (LI; AVGERIOU; LIANG, 2015)

Categoria	Descrição	Estudos
Análise estática de código	Analisa o código-fonte para identificar violações de regras de codificação, ausêncis de testes; calcula métricas de software baseada no código-fonte para identificar não conformidades em projeto técnico e arquitetura.	19
Análise de dependências	Analisa as dependências entre os diferentes tipos de elementos do sofware (componentes, módulos)	6
Checklist	Verifica uma lista de cenários pré-definidos onde incorrem em DT	1
Comparação de soluções	Compara a solução atual com a solução ótima em alguma dimensão, tal como a taxa de custo/benefício. Caso a solução atual não seja a ótima incorrerá em DT	2

Até aqui, podemos concluir que a **análise estática de código** se torna a via mais utilizada para a identificação de DT considerando principalmente a **manutenibilidade** como o principal fator de qualidade a ser verificado.

Neste contexto, observamos ainda o modelo proposto por (LETOUZEY, 2012), que procura mapear DT estabelecendo níveis de conformidade variando entre A e F. Neste modelo, os valores dos débitos são mapeados de acordo com as não-conformidades encontradas no projeto, ou seja, é necessário definir um mapeamento próprio ou utilizar o

A2DAM [6] como referência ao qual contém uma lista com 32 boas práticas. Neste sentindo, os aspectos impactados são seis, sendo eles: manutenibilidade, confiabilidade, eficiência, segurança, testabilidade e modificabilidade.

Analisando esta abordagem, vimos que propriedades como **confiabilidade** associadas a análise estáitca de código, não se adequa ao conceito do tempo em que a aplicação não apresentou falhas durante a sua execução, podendo gerar uma ambuiguidade na interpretação dos dados. Foi visto também que, o termo **segurança** não apresenta boas práticas associadas na planilha disponibilizada pelo A2DAM para a análise de DT.

Segundo (IZURIETA et al., 2012), existem diversas técnicas e ferramentas para a determinar DT, porém indica que existem lacunas e sobreposições entre classes e DT, que não tem sido rigosamente demonstrados. Os autores também indicam que relacionamentos entre anomalias no código e DT não tem sido demonstradas, e que o uso de técnicas para a organizar, visualizar, identificar e gerenciar DT não tem sido fornecidas de forma facilmente integradas para software na prática.

Para contornar tais problemas os autores definem um DT Landscape conforme a Figura 6. Como dinâmica tal abordagem se divide em 2 estágios, onde no primeiro, procura-se caracterizar e extrair os tipos de débito técnico, considerando as seguintes informações:

- Ferramentas de análise estática de código automatizadas, verificação de violações de modularidade e não-conformidades, bem como, verificação de padrões de projeto;
- Experiências em projetos reportadas por profissionais de forma qualitiativa, por meio de entrevistas, o que permite avaliar quais os tipos de débito técnico geraram um maior impacto nas atividades em execução.

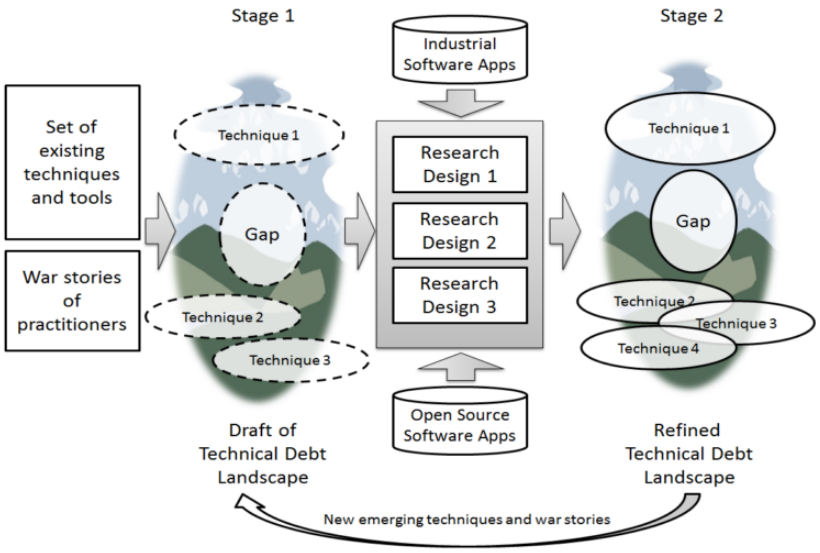
Após os levantamentos iniciais, o segundo estágio tenta refinar e validar as extrações obtidas, considerando 3 tipos de estudo:

- Comparação direta entre técnicas de identificação de DT, uso de mais de uma ferramenta de análise estática de código, aplicadas ao mesmo projeto;
- Avaliação das técnicas de DT para identificar débitos reais, por meio da participação de grupos avaliando sobre o impacto em corrigir ou não um determinado DT, sob pontos de vista distintos;
- Avaliação do relacionamento entre tipos de DT e manutenções futuras, avaliar uma tarefa de manutenção do software, considerando uma versão do projeto com DT e outra sem DT, validando qual o impacto dos DTs na manutenção do projeto.

Neste contexto, o que se procurou entender sobre Débito Técnico seria como mensurá-lo tendo como entrada o código-fonte de um projeto, bem como, um conjunto de métricas

[6] https://www.agilealliance.org/the-agile-alliance-debt-analysis-model/ - Acessado em 13/08/2018.

Figura 6 – Landscape Débito Técnico (IZURIETA et al., 2012)



extraidas. Vimos que análise estática e manutenibilidade, principal e juros são abordagens, atributos de qualidade e termos alinhados com o propósito desta tese.

Verificamos a ferramenta SonarQube analisando sua abordagem para mapeamento de DT e o custo de correção, sendo este relacionado com não-conformidades encontrados na análise do código-fonte, que por sua vez pode ter seu custo valorado em horas ou dias com base em uma tabela. De certo modo, esse formato de avaliação exige uma manutenção no mapeamento entre as não-conformidades e o seu tempo de correção. Além disso, não foi encontrado qual o entendimento utilizado para atribuir um custo para cada defeito mapeado.

Mais adiante será visto como esta tese procurou identificar DT e estimar seu custo por meio de métricas e uso de técnicas que consiste em derivar pontos de função a partir da quantidade de linhas de código, utilizando um fator de conversão conforme a linguagem de programação adotada.

3.6 DISCUSSÃO

Mesmo considerando que os tópicos tratados neste Capítulo se iniciaram entre as décadas de 70 e 90, ou seja, são temas bastante estudados e conhecidos, a investigação de tais assuntos se propôs a analisar as lacunas, as recomendações e como seria iniciada a ligação entre cada parte da fundamentação para a composição do método proposto. Neste sentido, este Capítulo apresentou os entendimentos necessários para a construção da proposta, observando os objetivos presentes nas fases prescritas pelo guia. Após identificados os pontos fortes e fracos nos tópicos abordados em cada Seção, tornou-se possível direcionar as perguntas que guiam este trabalho, sendo possível assumir como direcionadores os seguintes pontos:

- A abordagem no uso de mineração de dados em repositório de código, como forma de se avaliar o histórico do projeto, é algo factível, dado o conjunto de pesquisas que se utilizam de mineração e obtenção de código-fonte em repositórios, conforme apresentado na Seção 3.1;

- A extração de métricas automatizadas carecem de mapeamento entre os resultados coletados em código e o seu mapeamento em atributos de qualidade de forma automática, segundo a Seção 3.2, embora a abordagem proposta por (PLÖSCH et al., 2016) conteste a efetividade deste procedimento;

- O modelo que possui mais clareza e flexibilidade com relação a métricas e atributos de qualidade, dentre os apresentados na seção 3.3 foi o proposto por (BANSIYA; DAVIS, 2002). Para (PARIZI; GHANI, 2010) o modelo de AQS deveria seguir um modelo probabilistico baseado em séries temporais, dado que, testes convencionais não cobrem atributos de qualidade como: performance, confiabilidade e disponibilidade, algo possivel considerando que os projetos já estejam em execução, porém, esta abordagem não permite uma avaliação rápida de maneira estática para a coleta de indícios de forma preliminar.

- O monitoramento contínuo em qualidade é algo comum e que continua sendo abordado de forma séria e crescente por pesquisadores, conforme a Seção 3.4;

- Por fim, uso de técnicas para organizar, visualizar, identificar e gerenciar Débito Técnico não tem sido fornecidas de forma facilmente integradas para software na prática, de acordo com a Seção 3.5.

3.7 SUMÁRIO

Vimos que este Capítulo embasa esta tese, considerando os seguintes temas: repositórios de código, métricas, modelos para a avaliação de qualidade de software, monitoramento em qualidade de software e débito técnico. Deste modo, se torna possível visualizar a junção entre o estado da arte e as fases propostas neste contexto, permitindo a especificação de um método sistemático e repetível. No Capítulo 4, serão verificados os seguintes pontos:

- A elaboração do método proposto por esta tese;

- A avaliação do método em um projeto de código-aberto;

4 UM MÉTODO BASEADO EM MÉTRICAS PARA PREDIÇÃO DE IMPACTO EM PROJETOS DE SOFTWARE

Como primeiros passos, iniciamos este Capítulo alinhando os entendimentos necessários, subsidiados pelas fundamentações descritas nos Capítulos 2 e 3. O conceito da palavra **método**: substantivo masculino, que significa emprego de procedimentos ou meios para a realização de algo, seguindo um planejamento; rumo; conforme o Michaelis[1], direciona esta tese como um método para predição de impactos em atividades de codificação possibilitando o entendimento sobre qualidade do projeto por parte de todos os envolvidos, e, consequentemente auxliando na tomada de decisões. A seguir, serão descritos os passos conforme o guia da Figura 2.

4.1 VISÃO GERAL

Além da discussão apresentada na Seção 1.1, foram identificadas forças e motivadores que contribuem para a perda da qualidade nos projetos durante o trabalho desenvolvido dentro de uma consultoria de TI, sendo eles:

- Múltiplos fornecedores trabalhando nos mesmos projetos com o passar do tempo, término de contrato e desenvolvedores sem a obrigação de manter a qualidade do código por conta própria;
- Consultoria de TI que recebe esses projetos para manutenção, já o fazem com uma quantidade de não-conformidades legadas considerável.

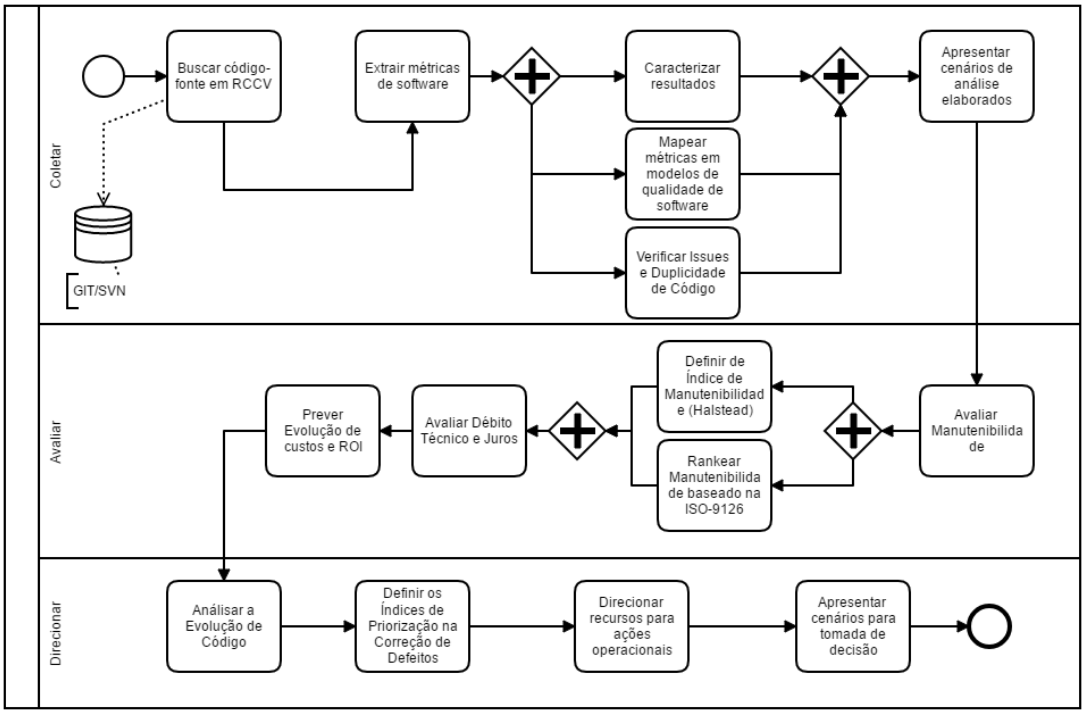
Problemas arquiteturais, que cuida da estrutura e aspectos não-funcionais, de design e trabalham o detalhamento técnico da especificação funcional de forma a atender as diretrizes definidas pela arquitetura, afetam constantemente a evolução de projetos de software.

Neste sentido, foi elaborado um roteiro contendo os passos necessários para o entendimento e execução desta proposta considerando três fases, sendo elas: **Coletar**, **Avaliar** e **Direcionar**, bem como, um conjunto de ações sendo suportadas por direcionamentos teóricos e técnicos.

Para exemplificar o método proposto por esta tese, a Figura 7, apresenta uma visão do fluxo de execução e as responsabilidades de cada fase. Neste fluxo as atividades estão agrupadas dentro de raias relativas a cada fase de execução do método. Nestas raias, as atividades podem ser executadas de forma sequencial ou paralela (representada pela estrutura de decisão com o símbolo +), porém, sem partir para a execução de outra fase sem o término da fase atual por completo.

[1] http://michaelis.uol.com.br/moderno-portugues/busca/portugues-brasileiro/método/

Figura 7 – Fluxo de execução do método proposto



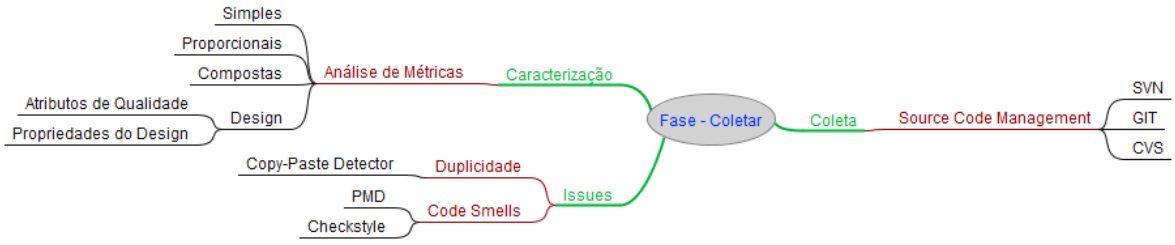
4.2 COLETAR

Como primeira fase esta tese se inicia como a atividade de **Coleta**, onde são previstas a **Caracterização de informações** e a análise de **Duplicidade código** em RCCV, seguindo o entendimento apresentado em 3.1.

Conforme a Figura 8, **Coletar** prescreve o obtenção do código-fonte por meio da busca em repositórios de projetos, porém, acrescido de verificações adicionais: como a análise do histórico de commits e as evoluções baseadas em métricas.

Caracterizar, verifica o tamanho e a complexidade de classes do projeto, executa o mapeamento de métricas para propriedades do design, e por fim, analisa a evolução de **Não-conformidades e Duplicidade de código**, observando o quanto tais itens contribuem para o aumento da dificuldade de entendimento em projetos de software. No restante desta Seção, será descrita a dinâmica desta fase.

Figura 8 – Fase - Coletar



4.2.1 Seleção de métricas

Conforme apresentado no Capítulo 2 e na Seção 3.2, a utilização de métricas constitui a base para a avaliação da qualidade interna do software por meio da análise estática do código, sendo este um dos objetivos trabalhados dentro desta tese. Embora exista um volume considerável com relação o estado da arte métricas, um conjunto de quinze métricas são destacados como os mais citados dentro dos trabalhos analisados. Nesta Seção, são propostas duas visões para a fase Coletar, sendo elas:

- A caracterização do software por meio da pirâmide de visão geral proposta por (LANZA; MARINESCU, 2007);
- O mapeamento de métricas de código em atributos de qualidade proposto por (BANSIYA; DAVIS, 2002).

Em ambos trabalhos já são sugeridas métricas para a execução das propostas, porém, foram verificadas se estas métricas estavam dentro daquelas citadas nos trabalhos avaliados. Sendo assim, as Tabelas 33 e 23, destacam as métricas sugeridas pelos trabalhos, o propósito e se estas métricas estão relacionadas dentre aquelas de maior estudo.

Tabela 22 – Métricas propostas por (LANZA; MARINESCU, 2007)

Métrica (Lanza)	Propósito	Relacionada
Número de classes (NoC)	Número de classes definidas no sistema, exceto classes contidas em bibliotecas	
Número de métodos (NoM)	Operações definidas pelo usuário, incluindo construtores	X
Número de pacotes (NoP)	Número de mecanismos de empacotamento em alto nivel	
Linhas de código (LoC)	Linhas de todas as operações definidas pelo usuário	X
Complexidade ciclomática (CYCLO)	Número de possiveis caminhos somados de todas as operações do sistema	X
Número de chamadas da operação (CALLS)	Número de operações chamadas por outras operações no projeto	X
Número de classes chamadas (FANOUT)	Classes das quais as operações chamam métodos	X

Para o segundo propósito: o mapeamento de métricas em atributos de qualidade, foram consideradas as seguintes métricas conforme proposto por (BANSIYA; DAVIS, 2002), destacado na Tabela 23 foram informadas as métricas utilizadas considerando a análise do propósito e a relevância da métrica no estado da arte.

Tabela 23 – Métricas propostas por (BANSIYA; DAVIS, 2002)

Propriedade do Design	Métrica (Bansiya)	Métrica utilizada	Propósito	Relac.
Tamanho do Design	(NoC)		número total de classes no design	
Complexidade	(NoM)	WMC	conta todas as operações definidas na classe	X
Hierarquias	(NOH)	DIT	número de hierarquia de classes no design	X
Abstração	(ANA)		número médio de classes do qual uma classes herda informações	
Encapsulamento	(DAM)		a taxa entre atributos privados com relação aos total de atributos de uma classe	
Acoplamento	(DCC)	CBO	o número total de classes do qual uma classes é diretamente relacionada	X
Coesão	(CAM)	LCOM4	computa o relacionamento de uma classes baseado na lista de parâmetros das operações	X
Composição	(MoA)		conta os relacionamentos entre classes por variáveis	
Herança	(MFA)	SI	procura identificar se de fato os métodos herdados são utiilzados pela classe-filha	X
Polimorfismo	(NoP)		número de operações em interfaces e métodos abstratos	
Mensagens	(CIS)	PM	número de métodos públicos em uma dada classe	X

Para considerar uma métrica relacionada, verificamos o propósito da mesma e se alguma dentro dos estudos possuiam semelhanças. Ao todo 16 métricas foram identificadas com duas ocorrências identicas em ambos os trabalhos, sendo encontradas cinco métricas de maior relevância.

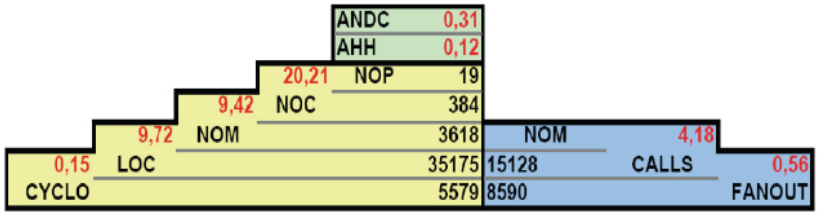
Por fim, para a seleção das métricas foram escolhidas àquelas constantes nas abordagens já propostas pelos autores, sendo avaliada a sua relevância como métrica dentro do estado da arte. Deste modo, este entendimento pode esclarecer quais as métricas estarão envolvidas na caracterização e no mapeamento para propriedades do design.

4.2.2 Caracterização

Um importante fator para o entendimento sobre a qualidade do código é o esclarecimento das métricas diretas obtidas, neste sentido, (LANZA; MARINESCU, 2007) propõe uma dinâmica de avaliação que permite de forma simples uma visão inicial da condição do projeto, observando 3 fatores: Tamanho e Complexidade, Acoplamento e Herança.

- **Tamanho e Complexidade** procura entender o quão grande e complexo um sistema é;
- **Acoplamento** procura demonstrar até que ponto classes (que criam objetos) estão acopladas umas as outras;
- **Herança** procura entender o quanto do conceito de herança é aplicado e se da melhor forma.

Figura 9 – Pirâmide de Visão Geral (LANZA; MARINESCU, 2007)



A Figura 9, apresenta a caracterização de um projeto selecionado por (LANZA; MARINESCU, 2007), apresentando os valores obtidos e passíveis de interpretação em alto nível. Neste sentido, podemos verificar que por meio de métricas diretas e proporcionais torna-se possível verificar a distribuição dos objetos dentro do projeto.

Considerando por exemplo, a divisão do número de pacotes (NOP) pelo número de classes (NOC), é possível visualizar se o empacotamento está bem distribuído, indicando se há uma alta ou baixa granularidade. Este mesmo entendimento se aplica para a quantidade de operações (NOM) por classes, bem como, o número de linhas de código (LOC) por operações.

Na porção à direita da Figura 9 está representada a visão sobre a quantidade de chamadas tanto de entrada (CALLS) quanto de saída (FANOUT) em que uma operação está envolvida, onde neste último, é possível ter uma visão sobre o quão dispersa estão as chamadas das operações entre as classes.

Seguindo este raciocínio, tais métricas associadas ao NOM tornam possível avaliar o nível de acoplamento entre as operações, bem como, quantas classes estão envolvidas na chamada de uma determinada operação. Por fim, as duas métricas no topo da pirâmide, apresentam a largura da árvore de herança (ANDC) e a profundidade de tais

heranças (AHH), permitindo verificar o uso intensivo de herança e a profundidade de tais hierarquias.

Além da coleta das métricas, foi necessário definir limites seguindo a abordagem proposta pelo (LANZA; MARINESCU, 2007), identificando em que estado se encontra o projeto comparado a outros projetos similares, e que, por meio de um cálculo simples se torna possível definir tais limites, sendo eles: **Baixo, Médio, Alto e Muito Alto**, calculando-os da seguinte forma:

- **Baixo**, valor da média - desvio padrão;
- **Média**, determina a tendencia central de um conjunto de dados;
- **Alto**, valor da média + desvio padrão;
- **Muito Alto**, (valor da média + desvio padrão) * 1.5.

A abordagem para a definição de limites apresentada por (LANZA; MARINESCU, 2007), se baseou na coleta de projetos com características variadas observando somente a tecnologia.

Para auxiliar no trabalho de definição de limites, vimos que existem as abordagens descritas no Capítulo 2, onde cada proposta apresenta uma dinâmica diferente de aferição.

Neste contexto, este tese trabalhou a definição de limites baseada no cálculo de desvio-padrão, ou seja, são retiradas as médias em termos de quantidade de operações, classes, complexidades e acoplamento, utilizando os objetos do próprio projeto. Considerando um intervalo de confiança de 95%, em seguida, são definidos os intervalos dos limites.

Desta forma, a extração de limites com base no histórico de revisões não foi considerada, porém, as abordagens de limites definidas no estado da arte podem ser considerados como referências caso hajam grandes diferenças entre os valores calculados e os valores indicados no estado da arte.

Um ponto positivo desta abordagem consiste na forma contínua de avaliação e indicação de limites ainda menores à medida em que as atividades de melhoria são executadas, outro fator positivo está em evitar a comparação com projetos que embora utilizem a mesma tecnologia, tenham outro propósito.

4.2.2.1 Considerações

Até este ponto, conforme os direcionamentos das subseções 4.2.1 e 4.2.2, se torna possível obter métricas e caracterizar o código-fonte obtido, utilizando-se de médias e desvio-padrão. Um dos principais pontos desta tese está em transpor métricas simples para um nível mais executivo, conforme será visto a partir da Subseção 4.2.3.

4.2.3 Propriedades do design

Além da coleta de métricas básicas e sua apresentação de forma evolutiva, outro fator importante está em caracterizar a qualidade do design e sua evolução histórica, para isto, o modelo proposto por (BANSIYA; DAVIS, 2002) apresentou-se como o modelo com o maior número de referências (1034) conforme apresentado no Capítulo 2. Dado que, a abordagem proposta por este trabalho prescreve a coleta e verificação de forma contínua, a Seção 3.4 discorreu sobre o uso de monitoramento contínuo como suporte a tal prática, e nesta Seção apresentamos e analisamos as seis propriedades propostas por (BANSIYA; DAVIS, 2002).

- **Reusabilidade**, permite que o design seja reaplicado em um novo problema com um baixo esforço;
- **Flexibilidade**, permite incorporar mudanças no design;
- **Entendimento**, permite o design ser facilmente entendido e compreendido;
- **Funcionalidade**, as responsabilidades das classes do design são expostas por meio de interfaces públicas;
- **Extensibilidade**, permite adicionar novos requisitos no design;
- **Efetividade**, permite se atingir a funcionalidade e o comportamento desejado por meio do uso de conceitos e técnicas orientadas a objetos.

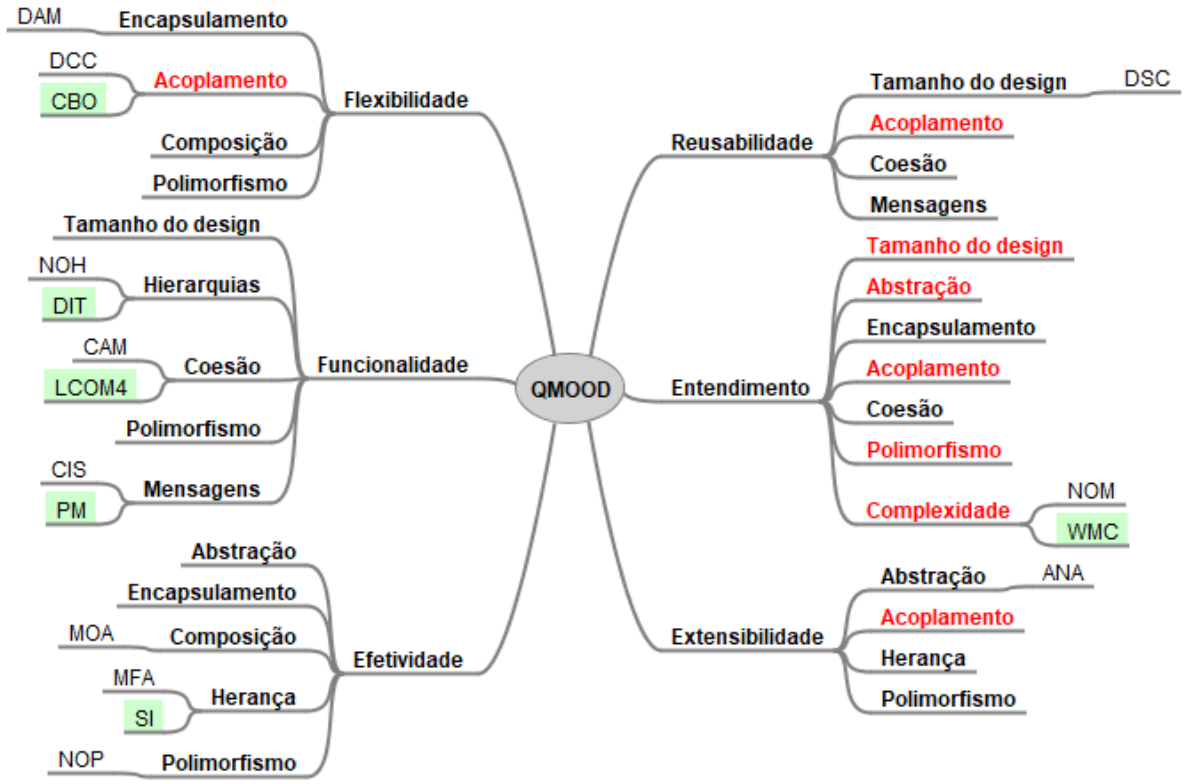
Conforme a Figura 10, é possivel verificar a composição de cada fórmula, sendo possível calibrar os pesos conforme a necessidade de cada projeto avaliado.

Figura 10 – Fórmula para a validação dos atributos de qualidade - (BANSIYA; DAVIS, 2002)

Quality Attribute	Index Computation Equation
Reusability	-0.25 * Coupling + 0.25 * Cohesion + 0.5 * Messaging + 0.5 * Design Size
Flexibility	0.25 * Encapsulation - 0.25 * Coupling + 0.5 * Composition + 0.5 * Polymorphism
Understandability	-0.33 * Abstraction + 0.33 * Encapsulation - 0.33 * Coupling + 0.33 * Cohesion - 0.33 * Polymorphism - 0.33 * Complexity - 0.33 * Design Size
Functionality	0.12 * Cohesion + 0.22 * Polymorphism + 0.22 Messaging + 0.22 * Design Size + 0.22 * Hierarchies
Extendibility	0.5 * Abstraction - 0.5 * Coupling + 0.5 * Inheritance + 0.5 * Polymorphism
Effectiveness	0.2 * Abstraction + 0.2 * Encapsulation + 0.2 * Composition + 0.2 * Inheritance + 0.2 * Polymorphism

De acordo com a Figura 10 elaboramos um quadro para esclarecer como as propriedades afetam os atributos de qualidade conforme a Figura 11, onde são apresentados o mapeamento entre métricas, atributos e propriedades de qualidade, onde os pontos em verde foram as substituições indicadas na Seção 4.2.1 considerando o estado da arte descrito na Seção 3.2.

Figura 11 – Propriedades x atributos de qualidade



Objetivando encontrar se fatores de qualidade apresentados por (BANSIYA; DAVIS, 2002) possuiam uma correlação, foi executado uma investigação com dados obtidos em mais de 12 projetos dentro da consultoria com 30 revisões para cada um deles. Utilizando a análise do coeficiente de correlação linear entre duas variáveis, conhecida como Pearson's r, a Figura 12, apresenta os resultados das correlações.

Figura 12 – Análise de correlação linear

	complexity	coupling	cohesion	messaging	encapsulation	composition	polymorphism	abstraction	hierarquies	inheritance	designsize
complexity	1	**0.993283377**	0.348647316	**0.944432048**	**-0.730883375**	**0.940465656**	**0.970143924**	-0.571286757	**-0.836736634**	**-0.912997999**	**0.97146167**
coupling		1	0.324898776	**0.957925374**	**-0.738488771**	**0.963530428**	**0.983286703**	-0.516398348	**-0.818466052**	**-0.874731706**	**0.983388306**
cohesion			1	0.099533305	**-0.726165217**	0.21236932	0.356709079	-0.70018202	-0.412337413	-0.522115352	0.178261204
messaging				1	-0.601041256	**0.918138194**	**0.938711817**	-0.290725561	-0.672459092	-0.774049854	**0.992898815**
encapsulation					1	-0.578484391	**-0.73429433**	**0.753466168**	**0.701882512**	**0.861573101**	-0.65664698
composition						1	**0.943759964**	-0.472925183	**-0.828636881**	-0.789577597	**0.947687835**
polymorphism							1	-0.450153747	-0.780427088	-0.812045002	**0.965808801**
abstraction								1	**0.805495548**	**0.790273792**	-0.381667688
hierarquies									1	**0.869820792**	-0.740703141
inheritance										1	**-0.816222907**
designsize											1

Após tal análise, foi possível constatar dois pontos importantes:

- Que os atributos de Encapsulamento, Hierarquias e Herança possuiam uma relação inversa com Complexidade;
- Que os atributos de Encapsulamento, Hierarquias e Herança possuiam uma relação inversa com Acoplamento;
- Que o Polimorfismo possuia uma relação inversa com o Hierarquias e Herança;

Mesmo considerando que as atividades de tradução de métricas, caracterização e mapeamento para propriedades do design, trazem uma outra visão do projeto, não podemos descartar a análise clássica da evolução de não-conformidades a ser vista na Seção 4.2.4, sendo este considerado como mais um fator que corrobora com os dados levantados nesta Seção.

4.2.4 Evolução de Não-Conformidades

A verificação da evolução de não-conformidades, auxilia em esclarecer o quanto as habilidades técnicas das equipes estão de acordo com as boas praticas definidas para o projeto.

Na abordagem proposta por (TUFANO et al., 2015), foram avaliados 200 commits de projetos executando uma verificação de não-conformidades em cada arquivo e suas evoluções subsequentes. Neste contexto, se torna possível identificar em que momento são introduzidos mais não-conformidades, comparando por exemplo se em tempos de entrega os desenvolvedores produzem mais ou menos não-conformidades no projeto.

Deste modo, o objetivo desta Seção está em definir que o acompanhamento sobre a evolução de não-conformidades possui um papel importante no direcionamento sobre o que deve ser refatorado. Neste sentido, devem ser observados os seguintes aspectos:

- As não-conformidades devem ser consolidadas por níveis de severidade, sendo eles: Alto, Médio, Médio Alto e Medio Baixo;
- Devem ser classificadas por categorias onde estas cobrem regras que variam de não-conformidades em termos de operações chegando ao nivel de classes entre pacotes;
- E por fim, devem ser permitir a análise de evolução por meio de gráficos, facilitando o monitoramento à medida em que mais revisões são acrescentadas ao projeto.

Por fim, esta visão subsidia um pouco mais a audiência com informações para a Tomada de Decisão, demonstrando que a evolução de não-conformidades possui relação com as propriedades de design apresentadas nesta abordagem, pois, conforme evoluem as violações de boas práticas, as propriedades do design podem sofrer um decréscimo de qualidade conforme os gráficos de Flexibilidade, Entendimento, Extensibilidade e Efetividade apresentados.

4.2.5 Discussão

Quando observamos a fase de Coleta como uma oportunidade de se obter evidências por meio da extração de dados e a análise de métricas dentro do método proposto por este trabalho, conseguimos então ter uma visão inicial sobre o quanto o projeto está teoricamente 'saudável' ou carecendo de direcionamentos em termos de melhoria. Embora a precisão das informações obtidas inicialmente possam gerar uma visão com indícios inexatos, torna-se possível entender ou tirar algumas conclusões sobre a qualidade do projeto avaliado.

Segundo (NUÑEZ-VARELA et al., 2017), dentre 226 estudos analisados em termos do estado de pesquisa em métricas de software, existem por volta de cento e quatorze estudos utilizando ferramentas comerciais e/ou desenvolvidas exclusivamente para o estudo desenvolvido. Neste sentido, o posicionamento desta tese não está em criticar a cobertura ou abrangência de cada ferramenta, mas, em indicar que é importante avaliar mais que somente a caracterização e a evolução do design para a tomada de decisão e entendimento dos impactos no projeto.

O que motivou a construção da dinâmica desta fase foi em facilitar a interpretação por parte dos *stakeholders* sobre as métricas extraídas em repositórios de projetos. De forma recorrente, a maioria dos projetos avaliados trabalharam na redução primeiramente de não-conformidades com maior grau de severidade, já que este serviria como argumento suficiente para o convencimento de algumas lideranças sobre uma abordagem razoável de qualidade de software dentro do projeto.

4.3 AVALIAR

Após os direcionamentos definidos na Seção 4.2 sobre como a atividade de caracterização se torna um importante meio para a obtenção de indícios de qualidade, o propósito desta fase está em verificar como associar os dados obtidos na fase anterior, observando aspectos de manutenibilidade, projeções de esforço e retorno do invesmento.

Durante as avaliações dos projetos executadas na consultoria de TI já citada neste trabalho, ficou constatada que a única abordagem de avaliação sobre a qualidade do código se restringia ao uso de uma ferramenta para a detecção de não-conformidades. Algumas outras atividades de engenharia reversa, com o objetivo de repasse do entendimento do software para novos membros dos times; e a criação de artefatos de solução técnica, nada mais eram do que entregáveis meramente formais para o antendimento a clásulas contratuais.

Diante deste cenário, as documentações produzidas e os resultados das não-conformidades coletadas por meio de ferramentas de análise estática, não estavam sendo utilizados como direcionadores para atividades de melhoria nos projetos, como visto na Seção 1.1.

No decorrer das avaliações dos times, foi verificado que não havia uma priorização

na correção das não-conformidades, e que, o percentual de correção dependeria dos limites pré-estabelecidos em contrato por solicitação do cliente, onde caso não houvesse tal formalização, o objetivo estaria em corrigir as não-conformidades partindo do nivel de severidade mais crítico, ou seja, a abordagem adotada não se preocupava de fato com priorização em termos de correções, bem como, quais os custos e retorno dos investimentos previstos com tais atividades.

Sendo assim, esta Seção tem como principal objetivo, definir o uso de uma abordagem para a medição de manutenibilidade, analisar os possíveis fatores que possivelmente afetam as atividades de manutenção, o custo relacionado ao esforço para se executar atividades de refatoramento, e, em que momento se dará o retorno do investimento.

4.3.1 Manutenibilidade

Como descrito no Capítulo 1, com o passar dos anos foi visto um volume considerável de serviços de manutenção de software forma geral demandadas à consultorias de TI. Sendo assim, algo comum nesta modalidade de serviços é o estabelecimento de prazos para a entrega das atividades, registrando cada solicitação seja de evolução ou correção como *tickets*, valorados com um prazo para a resolução e adotados como o acordo de nível de serviço entre as partes contratante e contratada.

Neste sentido, os prazos para a resolução são acordados conforme a definição de algum especialista do projeto, com o apoio do time de precificação, que se utiliza de uma comparação com projetos similares. Porém, durante as avaliações dos projetos foi possível verificar um desconforto por parte do time de desenvolvimento, dado que, a complexidade do código não era avaliada no estabelecimento dos prazos de entrega, gerando estimativas de resolução aquém do previsto.

Neste contexto, esta Seção avalia o estado atual do projeto considerando sua manutenibilidade por meio de duas abordagens:

- Utilizando o cálculo do Índice de Manutenibilidade (IM) definido por (OMAN; HAGEMEISTER, 1992);
- Utilizando um mapeamento entre métricas e características de qualidade descritas na (ISO/IEC 9126, 2001), seguindo a abordagem proposta por (HEITLAGER; KUIPERS; VISSER, 2007).

O objetivo da primeira abordagem é definir um IM por classes, fornecendo uma visão da qualidade de forma granular, ou seja, quais as classes são mais críticas e merecem uma maior atenção no projeto. Na segunda abordagem, o objetivo é definir um *ranking* de manutenibilidade do projeto para a sua utilização posteriormente na Seção 4.3.2 como insumo para a estimativa de esforço e projeção de custo de manutenção.

4.3.1.1 Índice de manutenibilidade

O ìndice de manutenibilidade (IM) definido por (OMAN; HAGEMEISTER, 1992), utiliza como base as métricas definidas por Halstead (DINARI, 2015), que possui como dinâmica contar a quantidade de operandos e operadores para avaliar a complexidade, a probabilidade de defeitos e a facilidade de entendimento. Segundo (NUÑEZ-VARELA et al., 2017), tais métricas tem sido largamente utilizadas e estudadas desde o final da década de 1970, estando dentre as métricas procedurais mais utilizadas. Ferramentas como Borland Togheter, Rational Software Analyzer e Visual Studio, também embarcaram tais métricas em seus produtos.

Deste modo, este método direciona o uso do IM para o cálculo da manutenibilidade, com intuíto de se obter um indício de tempo em termos de entendimento sobre os objetos do projeto, utilizando a seguinte fórmula:

$$\begin{aligned} IM = \max(0, (171 - 5.2 * \log(HalsteadVolume) - 0.23* \\ (CyclomaticComplexity) - 16.2 * \log(LineofCode)) * 100/171) \end{aligned} \tag{4.1}$$

A Tabela 24 abaixo, apresenta os limites para os resultados obtidos, categorizando-os em 3 níveis de manutenibilidade.

Tabela 24 – Resultados do Índice de Manutenibilidade (OMAN; HAGEMEISTER, 1992)

Intervalo	Manutenibilidade
0-9	Baixa
10-19	Moderada
20-100	Boa

Na Seção 4.5.2 verificaremos que os resultados em percentuais apresentaram valores mínimos considerando o número de classes com baixa qualidade de manutenção com relação ao número total de classes do projeto, porém, após uma investigação com relação ao tamanho das classes em termos de linha de código e tempos de entendimento, se constatou que o IM não deveria ser considerado o único fator para a avaliação de qualidade do projeto.

4.3.1.2 Ranking de Manutenibilidade

Dada a discussão apresentada na Seção 4.3.1.1, vimos que o uso do IM gera indícios sobre a qualidade do projeto em termos de manutenção por classes, e que embora haja a consideração do volume em termos de linha de código para a composição da sua fórmula , torna-se importante verificar o quanto tal volume corresponde percentualmente ao volume do projeto como um todo em termos de complexidade e operações.

Neste sentido, observamos que se tornou importante avaliar outras métricas que faziam sentido para a questão da manutenibilidade, o qual propusemos duas opções:

- O mapeamento entre métricas de manutenibilidade, associando os artigos conforme (NUÑEZ-VARELA et al., 2017) dentro da área de manutenção e o modelo da ISO 9126, deste modo, esta tese mapeou 31 métricas associadas às sub-características de manutenibilidade, descritas no Apêndice C. Nesta abordagem teríamos que definir os limites para cada métrica selecionada, se utilizando de abordagens descritas na Seção 2.2.1, e estabelecendo uma análise qualitativa dos resultados e consequentemente uma caracterização baseado em indicadores para um melhor entendimento;
- O modelo de avaliação de manutenibilidade seguindo as orientações descritas por (HEITLAGER; KUIPERS; VISSER, 2007), que também prescreve o uso da ISO 9126, porém, com uma abordagem utilizando cinco métricas para as mesmas sub-características da norma.

Diante deste cenário, dentre a duas opções selecionamos o segundo modelo de avaliação de manutenibilidade por considerar que as métricas para as três primeiras sub-característica são as mesmas, e que a diferença está na sub-característica de **testability** com uma quantidade maior de métricas passíveis de utilização.

Deste modo, de acordo com o segundo item como uma das opções complementares ao IM, este considera os seguintes fatores em sua proposta:

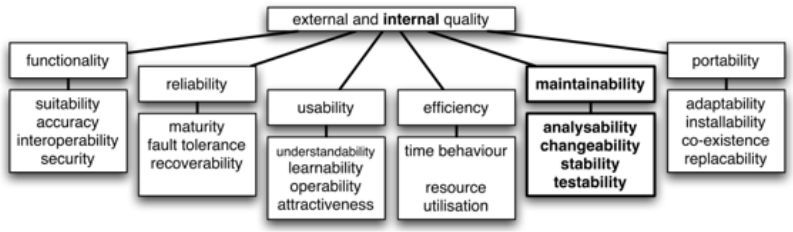
- *Volume* - a quantidade de linhas de código do sistema influencia na **analysability** do sistema;
- *Complexity per unit* - a complexidade influencia na **changeability** do sistema;
- *Duplication* - influencia na **analysability** e **changeability** do sistema;
- *Unit size* - influencia na **analysability** e **testability** do sistema;
- *Unit testing* - influencia na **analysability**, **stability** e **testability** do sistema;

Observando a Figura 13, podemos identificar a relação entre as características de manutenibilidade definidas pela ISO-9126 e o seu mapeamento com métricas obtidas na avaliação do código-fonte, sendo possível obter um entendimento sobre a qualidade de manutenção de projeto de forma direta.

Ainda segundo o modelo para a definição do *ranking* de manutenibilidade proposto (HEITLAGER; KUIPERS; VISSER, 2007), devemos considerar três verificações adicionais, sendo elas:

- *Duplicated code status* - o percentual relativo à duplicidade;
- *Complexity risk status* - como a complexidade das classes influenciam o sistema de forma geral;

Figura 13 – Visão da ISO-9126 mapeadas com propriedades do código-fonte

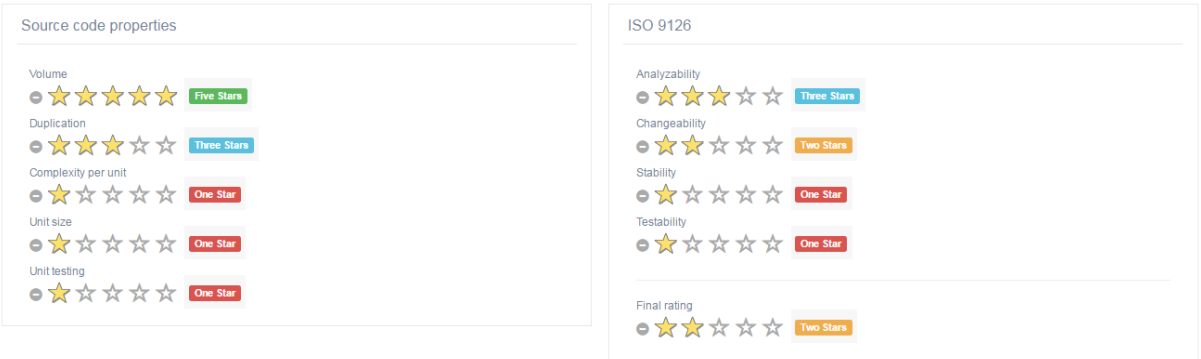


ISO 9126 maintainability	source code properties: volume	complexity per unit	duplication	unit size	unit testing
analysability	x		x	x	x
changeability		x	x		
stability					x
testability		x		x	x

- *Operations risk status* - como a complexidade está distribuída em operações/métodos das classes, considerando o tamanho das operações e seu percentual sobre as linhas de código do projeto.

Conforme a Figura 14(de forma ilustrativa), no lado esquerdo são apresentadas as propriedades do código-fonte de acordo com a definição da Figura 13, e no lado direito, as sub-características de manutenibilidade, onde ao final é exibida a classificação de manutenibilidade do projeto com uma qualificação de duas estrelas.

Figura 14 – Ranking de manuteniblidade conforme a ISO-9126



4.3.1.3 Discussão

Concluindo esta Seção, observamos que o tema manutenibilidade pode ser avaliado por meio do uso de métricas diretas e proporcionais, de forma prática e eficiente. Além disso, vimos que a utilização de um único índice poderia comprometer a assertividade da avaliação, deste modo, propomos dois tipos de avaliação.

O que deve ficar claro é que as teorias apresentadas neste contexto não foram criadas pelo autor desta tese, porém, foram estruturadas e adaptadas com relação a limites, bem como, o refinamento dos detalhes técnicos do ponto de vista de adequação ao método proposto e a construção de uma ferramenta de suporte.

A partir deste ponto se torna possível identificar informações que podem esclarecer o nível de qualidade em termos de manutenção do projeto, considerando evoluções e melhorias futuras. Ainda neste contexto, as abordagens apresentadas permitem identificar pontos de perda de qualidade em um formato mais granular, com indicadores que facilitam o entendimento considerando uma audiência mais estratégica.

4.3.2 Débito técnico e Juros

Algo bastante comum em empresas de desenvolvimento de software é a construção sem o devido acompanhamento em termos de qualidade do código produzido, onde por vezes, os problemas só começam a receberem a devida atenção quando já estão gerando transtornos em termos de prazo e custo, levantando questionamentos sobre a viabilidade em se manter ou reconstruir o projeto, e qual o retorno do investimento conforme a tomada de decisão.

Neste contexto, somado ao conceito de débito técnico descrito na Seção 3.5, (NUGROHO; VISSER; KUIPERS, 2011) define o conceito de **juros** como um esforço extra para a adequação de pontos de correção.

Tido como um dos principais fatores que influenciam a tomada de decisão sobre a continuidade de um projeto, a visão financeira deve considerar qual o custo em se manter ou refatorar o projeto atual, bem como, sobre quando ocorrerá o Retorno do Investimento (RoI).

Deste modo, verificaremos como avaliar em termos financeiros a situação do projeto, adaptando o modelo proposto por (NUGROHO; VISSER; KUIPERS, 2011), para projetar e obter uma visão de retorno do investimento de tal decisão. Para iniciar a construção dessa abordagem, podemos entender o restante dessa Seção como um manual prático, considerando os seguintes pontos:

- Conversão de Linhas de Código do projeto para Pontos de Função (PF), utilizando a técnica de *Backfiring* definida por (JONES, 1995);
- O modelo de conversão de Pontos de Função para Pontos de Produtividade (PP) proposto por (LANZA, 2008), conforme a Tabela 25;
- O cálculo do Valor de Reconstrução (VR), onde este se dá pela divisão da quantidade de Linhas de Còdigo (LoC) pela quantidade de instruções conforme a linguagem de programação, segundo a tabela definida por (JONES, 1995), divididos pelos PP, ou seja, VR = (LoC/Backfiring(Linguagem))/PP;

Tabela 25 – Tabela de conversão de pontos de função em pontos de produtividade (LANZA, 2008)

Tamanho do projeto	PF	PP (Java)
Pequeno	50-350	18
Medio-Pequeno	350-650	16
Medio	650-1100	14
Medio-Grande	1100-2000	12
Grande	>2000	10

Para a composição do cálculo do débito técnico e juros, devemos considerar a escala de rankeamento proposta na Seção 4.3.1.2, e, conforme a Figura 15, para a melhoria da qualidade do software é necessário definir o esforço de evolução em níveis, sendo este calculado com base nos percentuais definidos para cada nível que se deseja atingir.

Figura 15 – Tabela de conversão entre níveis (NUGROHO; VISSER; KUIPERS, 2011)

Target/Source	1-star	2-star	3-star	4-star	5-star
1-star					
2-star	60%				
3-star	100%	40%			
4-star	135%	75%	35%		
5-star	175%	115%	75%	40%	

Conforme a Figura 15, para passar um determinado projeto de nível 2-estrelas para o próximo nível, deverá ser aplicado um percentual de esforço em torno de 40%.

Além do cálculo do Débito técnico, devem ser considerados os Juros que incorrem com o passar do tempo, e que por sua vez, depende do cálculo do Esforço de Manutenção (EM). Para o cálculo do EM são necessários 3 fatores:

- A Fração de Manutenibilidade (FM), que define o percentual previsto de manutenção de um projeto ao longo do ano, ou seja, um determinado projeto pode possuir um percentual anual de 15% relativos à manutenção;
- o Fator de Qualidade (FQ) descrito na Tabela 26 de acordo com o ranking do projeto;
- e o Valor de Reconstrução (VR).

Neste contexto, o EM e os Juros são calculados de seguinte forma:

$$EM(Ranking) = \frac{FM * VR}{FQ} \tag{4.2}$$

$$Juros = EM(RankingAtual) - EM(5 - Estrelas) \tag{4.3}$$

Tabela 26 – Fator de qualidade (NUGROHO; VISSER; KUIPERS, 2011)

Ranking	Fator
1 Estrela	0.5
2 Estrelas	0.707
3 Estrelas	1.0
4 Estrelas	1.414
5 Estrelas	2.0

Deste modo, torna-se possível projetar para os próximos 10 anos a evolução do EM e Juros, conforme a Figura 16.

Figura 16 – Projeçao de custos em manutenção (NUGROHO; VISSER; KUIPERS, 2011)

Technical debt on a 10-years horizon (2 stars)

Year	Technical Debit	Interest	Maintenance Effort	Days/person
1	55.50532345013477	6.620292407080524	10.240204806002357	215
2	58.28058962264151	6.95130702743455	10.752215046302474	226
3	61.19461910377358	7.298872378806276	11.289825798617597	237
4	64.25435005896226	7.663815997746591	11.854317088548477	249
5	67.46706756191038	8.047006797633923	12.447032942975904	261
6	70.8404209400059	8.44935713751562	13.0693845901247	274
7	74.3824419870062	8.8718249943914	13.722853819630936	288
8	78.10156408635653	9.31541624411097	14.408996510612484	303
9	82.00664229067435	9.781187056316519	15.129446336143108	318
10	86.10697440520806	10.270246409132344	15.885918652950263	334

Technical debt on a 10-years horizon (Next level)

Year	Technical Debit	Interest	Maintenance Effort	Days/person
1	36.19912398921833	3.6199123989218327	7.239824797843665	152
2	38.00908018867925	3.8009080188679247	7.6018160377358495	160
3	39.90953419811321	3.9909534198113206	7.981906839622641	168
4	41.905010908018866	4.190501090801886	8.381002181603773	176
5	44.00026145341982	4.4000261453419816	8.800052290683963	185
6	46.20027452609081	4.620027452609081	9.240054905218162	194
7	48.510288252395355	4.851028825239536	9.702057650479071	204
8	50.93580266501513	5.093580266501513	10.187160533003025	214
9	53.482592798265884	5.348259279826588	10.696518559653176	225
10	56.15672243817918	5.615672243817918	11.231344487635836	236

Embora esteja claro os benefícios em termos de custo de manutenção, caso seja tomada a decisão de passar o projeto para o nível 3, se torna importante visualizar a partir de que momento ocorrerá o retorno do investimento aplicado nas atividades de melhoria. Para isto se faz necessário o seguinte cálculo:

$$DiferencaAcumuladaDeEM = \sum EM(RankingAtual) - EM(ProximoRanking) \tag{4.4}$$

$$RoI = \frac{DiferencaAcumuladaDeEM - DebitoTecnico}{DebitoTecnico} * 100 \tag{4.5}$$

Na Fórmula 4.4, o cálculo da diferença acumulada considera a soma do esforço de manutenção no ranking atual, subtraindo-o do esforço de manutenção para o próximo ranking.

Na Fórmula 4.5, o retorno do investimento é calculado pela fração entre a diferença acumulada sobre o esforço de manutenção e o valor do débito técnico, sobre o valor do débito técnico multiplicado por cem.

4.3.3 Discussão

Encerramos aqui a fase **Avaliar** do método, apresentando um modelo de verificação de qualidade baseado em mapeamento, *rankeamento* e direcionamentos do ponto de vista de manuntenibilidade e de custos, permitindo a tomada de decisão com relação à melhorias futuras do projeto.

O que podemos verificar na argumentação apresentada na Seção 4.3, é que dada as condições para o entendimento sobre os custos associados à manutenibilidade do projeto por meio de indicadores, tais resultados podem agregar valor na ánalise de novos contratos de manutenção, tendo como justificativa, os futuros direcionamentos com relação a redimensionamento de equipes e o retorno do investimento.

4.4 DIRECIONAR

O direcionamento sobre como proceder melhorias em projetos de software se torna um importante ponto de atenção quando são considerados fatores como: *time-to-market*, disponibilidade de recursos e custo.

Neste sentido, esta fase pode ser analisada sob dois aspectos: onde o primeiro deve verificar quais classes do projeto merecem um trabalho de refatoramento e melhoria, e o segundo, analisar como o time de desenvolvimento esta apto a atuar em melhorias, observando autores e membros que mais contribuíram para a evolução do projeto, mais especificamente nos pontos sugeridos para a manutenção elencados pela primeiro ponto.

Deste modo, podemos dividir esta Seção em duas partes: a definição de um modo de priorização e correção de defeitos e, a indicação de quais recursos estão mais aptos a atuar em uma determinada ação de melhoria, o que pode ser chamado de designação de membros do time.

Observando o primeiro aspecto, durante os levantamentos que subsidiaram esta tese, foram verificados que mesmo utilizando ferramentas para a detecção de não-conformidades, estes eram priorizados e corrigidos com relação à sua severidade, ou seja, considerando uma escala com 4 níveis de criticidade: *Blocker*, *Critical*, *Major* e *Minor*.

Embora a abordagem de correção baseada em níveis de criticidade faça sentido, na prática, a alocação de equipes especificamente para a correção de defeitos é escassa tanto no que diz respeito a quantidade de profissionais, bem como, quanto a experiência dos membros do time.

Contudo, durante as avaliações dentro da consultoria foi bastante comum os projetos alocarem mais recursos voltados para a construção e entrega, tornando a tarefa de correção uma atividade menor dentro do projeto. Um ponto de atenção nesta abordagem foi quando a quantidade de não-conformidades por criticidade atingiram volumes altos, o que gerou nos times a necessidade de se entender que existiam duas situações com relação às atividades de priorização e correção de defeitos:

- Correção de não-conformidades para a entrega do produto, onde visivelmente o cliente, dado o seu nível técnico e politico, está preocupado em comprovar e apresentar que a entrega recebida está em boas condições;

- Correção de não-conformidades mantendo o projeto com um grau de qualidade aceitável, facilitando a manutenção e evolução do projeto de maneira menos custosa.

Neste sentido, o que foi identificado nesta fase é que as equipes não possuíam outra abordagem para a correção de defeitos além do argumento da criticidade. e que algumas classes obrigatoriamente eram alteradas devido ao seu grau de acoplamento, porém, não havia um trabalho de melhoria destas, somente o acréscimo ou retirada de alguma operação.

Diante disto, foi investigado como seria possível direcionar melhorias no código, dada as evoluções das classes com relação a algum critério, desta maneira, seria possível identificar o grau de mudanças que justificariam a melhoria daquela parte do código naquele momento. Contudo, a presença somente de alterações em uma determinada classe não a tornaria candidata a uma atividade de melhoria sem identificar qual a qualidade desta em termos de complexidade e acoplamento. Além disso, outro ponto importante foi o de buscar algum fator externo à contagem de alterações que pudesse corroborar com a avaliação baseada em evoluções das classes.

Neste contexto, a pergunta principal como justificativa para esta fase do método é: ***Por que é necessária a priorização e correção desta classe nesta fase do projeto?*** Deste modo, esta tese concluiu após algumas investigações, que seria importante o direcionamento e a criação de um Índice de Priorização na Correção de Não-Conformidades (IPCNC), para subsidiar a seleção do que deveria ser corrigido com base em três itens:

- A análise de evolução de métodos proposta por (GÎRBA; DUCASSE; LANZA, 2004), neste, são avaliadas as evoluções das classes baseadas no acréscimo e retirada de operações, classificando-as em alterações mais antigas ou mais recentes;

- O cálculo de Volatilidade Histórica[2], aplicado na previsão de ativos do mercado financeiro, deste modo, se torna possível avaliar a previsibilidade de mudanças de uma classe, observando uma abordagem similar a adotada por (VIDAL; MARCOS; DÍAZ-PACE, 2016);

- A complexidade e o acoplamento da classe, bem como, suas atualizações conforme as revisões do projeto, onde, os primeiros são classicamente problemas de qualidade, e o último, verifica que não somente acréscimo e retirada de operações são importantes fatores, mas qualquer alteração dentro da classes deveria ser computada, porém com um peso menor.

[2] http://www.investopedia.com/terms/h/historicalvolatility.asp

4.4.1 Composição do IPCNC

A motivação na criação do Índice de Priorização de Correção de Não-Conformidades (IPCNC), surgiu da necessidade em se apoiar as atividades de melhoria de código em termos de refatoramento, dado o volume de não-conformidades inseridas nos projetos. Como dinâmica, o processo de definição do índice considerou a variação e a quantidade de modificações de atributos e operações em classes do projeto durante as atividades de codificação.

Para este fim, consideramos uma técnica de avaliação sobre a evolução de métodos proposta por (GÎRBA; DUCASSE; LANZA, 2004), sendo avaliados três fatores:

- ENOM (Evolution of Number of Methods), contabilizando o acréscimo e a retirada de métodos para cada classe do sistema em cada versão;
- LENOM (Latest Evolution Number of Methods), as alterações mais recentes como peso maior na previsão de alterações futuras;
- EENOM (Earliest Evolution Number of Methods), as alterações mais antigas como peso maior na previsão de alterações futuras;

Deste modo, a utilização destes três fatores permite avaliar e priorizar a correção de defeitos considerando componentes que estão sendo utilizados mais recentemente, permitindo que os efeitos da melhoria seja refletida de forma imediata, dado que o objeto possui uma probabilidade maior em ser acionado durante a construção de uma nova funcionalidade do sistema.

Somada a essa abordagem, trouxemos uma abordagem aplicada no mercado financeiro, utilizando o conceito de análise de Volatilidade Histórica (HV), que verifica por um período de tempo a variação de um ativo financeiro. Neste sentido, foi considerado como período a quantidade de revisões avaliadas ao invés do período de 252 dias utilizados no mercado financeiro definido pelo Banco Central. Deste modo, foi avaliado o quanto cada classe variou em termos de adição e retirada de operações durante uma determinada quantidade de revisões.

A quantidade de atualizações sofridas pela classe (UPD) que considera qualquer tipo de alteração além do acréscimo e retirada de operações com base na verificação em algoritmos de somas de verificação (*checksum*), também foi considerada.

Além disto, esta tese considerou fatores relativos à qualidade do código e que possívelmente se tornam ofensores do design. Sendo assim, foram selecionados dois fatores, sendo eles:

- **Acoplamento - DCC**, que caracteriza o grau de dependencia entre pacotes, módulos e objetos no projeto. Um alto grau de acoplamento reduz a flexibilidade, aumenta

proabilidade de propagação de erros, podendo causar efeitos colaterais à medida em que são trabalhadas novas modificações. (CHIDAMBER; KEMERER, 1994)

- **Complexidade - WMC**, uma medida quantitativa do número de caminhos independentes através de código-fonte de um programa. Utilizando um grafo de fluxo de controle, os nós representam os comandos de forma isolada e as linhas conectam dois nós, quando o segundo comando é executado imediatamente após o primeiro. (MCCABE, 1976).

Conforme a Equação 4.6, podemos identificar que para as evoluções mais recentes foi atribuído um peso 3, para as alterações mais antigas atribuido o peso 3 negativo, para a volatilidade histórica e as atualizações o peso 1, e por fim, para o acoplamento e a complexidade o peso 2.

$$IPCNC = (LENOM * 3) + (EENOM * (-3)) + HV + UPD + (DCC * 2) + (WMC * 2) \tag{4.6}$$

A partir da definição da Equação 4.8, foi elaborado um modelo por meio da execução de uma análise de regressão linear múltipla para investigar a dependência entre as variáveis e elaborar estimativas sobre os parâmetros definidos para o IPCNC.

Sendo assim, o intervalo de confiança correspondeu a 95%, e, a massa de dados utilizada para a elaboração da análise constava de 49 observações, ou seja, as classes foram analisadas considerando o histórico de revisões (por volta de trinta) no repositório de código-fonte de projetos dentro da consultoria estudada.

Conforme os resultados apresentados na Figura 17, é possível destacar os seguintes pontos:

- O R-multiplo indica o Coeficiente de Correlação Linear de Pearson em 100%;
- O coeficiente de determinação também conhecido como R-Quadrado contém o valor igual a um, o que indica que o modelo consegue explicar os valores observados em 100%, deste modo, 100% da variação de y é explicada pela variação de x;
- R-quadrado ajustado que indica o Coeficiente de Determinação levando em conta a quantidade de variáveis e observações;
- Erro de estimativa da média girando em torno de 6,8251E-16.
- Teste ANOVA, testa a hipótese de que existe relação linear entre as variáveis. Quando o F de significação for menor que 0,05, existe relação linear entre as variáveis.

Figura 17 – Análise de regressão

RESUMO DOS RESULTADOS

Estatística de regressão	
R múltiplo	1
R-Quadrado	1
R-quadrado ajustado	1
Erro padrão	0.00000000000000068251
Observações	49

ANOVA

	gl	*SQ*	*MQ*	*F*	*F de significação*
Regressão	6	4.832364244	0.805394	1.73E+30	0
Resíduo	42	1.95647E-29	4.66E-31		
Total	48	4.832364244			

	Coeficientes	*Erro padrão*	*Stat t*	*valor-P*	*95% inferiores*	*95% superiores*	*Inferior 95.0%*	*Superior 95.0%*
Interseção	2.498E-16	1.89236E-16	1.320045	0.193969	-1.32094E-16	6.31694E-16	-1.32094E-16	6.31694E-16
LENOM	0.5	8.45085E-16	5.92E+14	0	0.5	0.5	0.5	0.5
EENOM	-0.5	2.18141E-16	-2.3E+15	0	-0.5	-0.5	-0.5	-0.5
HV	0.166666667	4.16511E-16	4E+14	0	0.166666667	0.166666667	0.166666667	0.166666667
UPD	0.166666667	5.67762E-16	2.94E+14	0	0.166666667	0.166666667	0.166666667	0.166666667
DCC	0.333333333	6.02524E-16	5.53E+14	0	0.333333333	0.333333333	0.333333333	0.333333333
WMC	0.333333333	1.03687E-15	3.21E+14	0	0.333333333	0.333333333	0.333333333	0.333333333

Com base no modelo de regressão criado, podemos simular a priorização por meio da seguinte Equação 4.7.

$$\begin{aligned} RANKING = 2,498E-16 + (VALOR1 * 0,5) + (VALOR2 * -0,5) + \\ (VALOR3 * 0,166666667) + (VALOR4 * 0,166666667) + \\ (VALOR5 * 0,33333333) + (VALOR6 * 0,33333333) \end{aligned} \tag{4.7}$$

Utilizando o modelo descrito, a Figura 18 apresenta uma simulação para exemplificar uma estimativa dos parâmetros em termos de priorização de classes. Neste contexto, podemos verificar que a quantidade de alterações mais recentes (LENOM) embora possua um peso maior, não necessariamente se torna um fator decisivo para a priorização de correções, dado que a CLASSE D possui um histórico de alterações mais recentes que a CLASSE F. Contudo, fatores de qualidade como Complexidade e Acoplamento contribuíram para que esta última estivesse acima na ordem de priorização das correções.

Figura 18 – Priorização de classes - simulação de estimativa de parâmetros do modelo

	RANK DE PRIORIZAÇÃO	LENOM	EENOM	HV	UPD	DCC	WMC
CLASSE F	11,66666653	2	6	1	1	10	30
CLASSE D	11,16666657	4	1	1	1	8	20
CLASSE G	10,38333324	5	6	4	1	20	10
CLASSE E	10,16666657	2	1	1	1	8	20
CLASSE C	8,333333261	3	2	1	2	7	15
CLASSE A	4,499999969	1	1	3	4	5	5
CLASSE B	4,333333283	2	7	2	7	6	10

Além da abordagem com relação a relevância das variáveis conforme a Figura 17, elaboramos uma segunda análise por meio do Processo de Hierarquia Analítica (AHP)

(SAATY, 2008), que se caracteriza por um processo de tomada de decisão, onde os decisores escolhem opções dentre algumas alternativas.

Como decisores foi solicitada a participação de dois profissionais da consultoria, sendo um analista e um programador com experiência em métricas de código, onde e após a explicação das variáveis foi solicitado que estes apoiassem no julgamento.

Neste sentido, foram elaborados quatro passos para a utilização do método, sendo eles: requisitos do índice, alternativas, cálculo do peso e elaboração da tabela de decisão.

- Requisitos, conforme os índices selecionados quais os mais importantes para a predição de alteração em classes do projeto;
- Alternativas, no nosso modelo definimos o uso das varíáveis definidas na formula proposta;
- Cálculo dos pesos, onde foram atribuidos: 0,5 para fraca importância, 1 para importância igual e 9 para extrema importância;
- Tabela de decisão, elaborada conforme os pesos calculados no item anterior. Cada linha representa os parâmetros e as colunas as alternativas.

Figura 19 – AHP - IPCNC

	EENOM	LENOM	HV	UPD	DCC	WMC
EENOM	1	0,5	0,5	0,5	0,5	0,5
LENOM	2	1	9	9	9	9
HV	2	0,111111	1	1	0,5	0,5
UPD	2	0,111111	1	1	0,5	0,5
DCC	2	0,111111	2	2	1	9
WMC	2	0,111111	2	2	0,111111	1
	11	1,944444	15,5	15,5	11,61111	20,5

EENOM	0,090909	0,257143	0,032258	0,032258	0,043062	0,45563	**9,1%**
LENOM	0,181818	0,514286	0,580645	0,580645	0,77512	2,632514	**52,7%**
HV	0,181818	0,057143	0,064516	0,064516	0,043062	0,411055	**8,2%**
UPD	0,181818	0,057143	0,064516	0,064516	0,043062	0,411055	**8,2%**
DCC	0,181818	0,057143	0,129032	0,129032	0,086124	0,58315	**11,7%**
WMC	0,181818	0,057143	0,129032	0,129032	0,009569	0,506595	**10,1%**
	1	1	1	1	1	5	

Na Figura 19, foram atribuídas as pontuações conforme a relevância entre os atributos e os parâmetros, obtendo como resultados os percentuais de relevância conforme apresentado na coluna em verde.

Enfim, tanto a análise de regressão que se baseia nos valores coletados nos repositórios dos projetos, quanto na utilização do AHP que consiste em uma avaliação por especialista e nesse caso atribuímos os valores conforme os direcionamentos obtidos na literatura e

na experiência dos participantes, podemos ter um entendimento sobre a composição das variáveis da fórmula.

Embora a proposição nesta Seção seja voltada para a criação de um índice facilitador para predição de possíveis correções, e não a proposição de uma métrica, foram pesquisados quais os critérios necessários para a validação de uma métrica com objetivo de avaliar se existia a aderência em alguns desses critérios pelo índice proposto.

Segundo (MENEELY; SMITH; WILLIAMS, 2013), existem 47 critérios para a validação de uma métrica, e não necessariamente tal métrica deva atender a todos os critérios para ser considerada válida. Neste sentido, destacamos alguns pontos de aderência do IPCNC a alguns desses critérios.

- *Actionability* permite que um gerente de software tome uma decisão empiricamente tendo como base o índice apresentado;
- *Empirical validity* tem validade se a observação dos resultados do índice corroboram com a medida pretendida de uma métrica. Neste sentido foi apresentado um modelo de regressão linear com um R-Quadrado igual a 1;
- *Improvement validity* possui uma validade de melhoria, caso a métrica seja uma melhoria de métricas existentes. Neste sentido o índice se utiliza de propostas relacionadas a três fatores, com o acréscimo de fatores do mercado financeiro e métricas de complexidade e acoplamento em sua composição;
- *Usability* para sua viabilidade é necessário que haja a facilidade na coleta da métrica, deste modo, foi proposta a utilização métricas academicamente conhecidas cuja a coleta pode ser executada por meio de ferramentas de análise estática e análise do histórico de commits no repositório de código-fonte.

4.4.2 Designação de membros do time

Um dos principais problemas encontrados em atividades de melhoria e refatoramento de software, está em identificar quais os membros do time possuem um maior conhecimento sobre o objeto a ser trabalhado. Obviamente, é bastante comum identificar que o autor da classe é o principal responsável pelas informações e que este possui o maior conhecimento sobre o papel desta, porém, o que deve ser avaliado além da autoria, são as contribuições a tal classe à medida em que projeto evolui.

Segundo (WILLIAMS; KESSLER, 2002), o conceito de *Truck Factor* designa um número mínimo de desenvolvedores que podem ser atingidos por um caminhão antes que um projeto se torne incapacitado. De acordo com a Wikipedia, que conceitua o termo *Bus Factor* com o mesmo propósito, este define como *"uma medida de risco resultante da falta de compartilhamento de informações e capacidades entre os membros de um time".*

Neste sentido, verificando a abordagem proposta por (AVELINO; VALENTE; HORA, 2015), esta fase de direcionamento deve propor quais os membros estão aptos a orientar ou atuar na melhoria das classes apresentadas na Seção 4.4.1. Para este fim, este método analisou o grau de autoria proposto por (FRITZ et al., 2014), onde o autor é um desenvolvedor capaz de influenciar ou comandar a construção ou implementação de uma classe. Deste modo, se estabelece a medida de grau de autoria por meio da seguinte fórmula:

$$DOA = 3.293 + 1.098 * FA + 0.164 * DL - 0.321 * ln(1 + AC) \tag{4.8}$$

Deste modo, o grau de autoria de um desenvolvedor d e uma classe c depende de 3 fatores: primeira autoria (FA), número de entregas ou *commits* (DL), e o número de aceitações, sendo assim, se d é o autor de c, FA é igual a 1; senão é igual a 0; DL corresponde ao número de mudanças em c feitos por d; e AC é o número de modificações em c feitas por outros desenvolvedores. Conforme (FRITZ et al., 2010), os pesos utilizados na Equação 4.8, foram derivados de um experimento envolvendo 7 profissionais em desenvolvimento Java. Os autores mostraram que o modelo é robusto o suficiente para ser utilizado em diferentes ambientes e projetos.

Ainda nesta equação, podemos verificar que a autoria da classe possui um peso maior se comparada às contribuições de outros desenvolvedores e que o valor destas contribuições diminui minimamente o peso do grau de autoria da classe.

Por fim, é possível avaliar que tanto o autor como outros membros do time, através do percentual de contribuição sobre cada classe, podem atuar como direcionadores nas atividades de melhorias prescritas na Seção anterior.

4.4.3 Discussão

Para a fase Direcionar, foram apresentadas duas abordagens: a primeira sugerindo ações em partes do projeto onde possivelmente ocorrerão novas alterações, e a segunda, direcionando quais os recursos do time que históricamente possui conhecimentos em tais partes. Como resultado esperado, a abordagem proposta buscou a designação de tarefas de forma mais direcionada, gerando uma maior economia de esforço.

Se considerarmos somente o ponto de vista de designação de membros do time visto na Seção 4.4.2 quanto a efetividade dos profissionais baseados em *commits*, é possível ainda, extrair métricas para avaliar outros aspectos do ponto de vista de desempenho da equipe, como por exemplo, o quanto cada componente insere defeitos no projeto e qual o grau de qualidade do código de cada membro do time.

Segundo (TUFANO et al., 2015), utilizar as revisões sob a perspectiva de quando ocorrem o surgimento de não-conformidades no decorrer das atividades de codificação, permite analisar em que momento do projeto isto ocorre mais frequentemente. Deste modo, esta abordagem vinculada ao cronograma do projeto do projeto permitiria avaliar se em períodos de entrega tais ocorrências são mais frequentes.

4.5 APLICAÇÃO DO MÉTODO

O objetivo desta Seção consiste em exemplificar a utilização do método proposto para fins de avaliação, considerando as fases apresentadas por esta tese. Para isto, foi selecionado um projeto de código aberto permitindo a exibição de dados passíveis de verificação.

Para este fim, o projeto JHotDraw foi selecionado pelos seguintes motivos:

- Dado que esta tese propôs um método de avaliação observando o paradigma orientado a objetos, o projeto selecionado foi desenvolvido na linguagem Java;
- Observando o quesito popularidade, foram investigadas a quantidade de citações em sites acadêmicos sobre experimentos utilizando o projeto. Utilizando a string de busca "jhotdraw open source project", no Google Scholar foram retornados 1.410 resultados, procurando por somente a palavra "jhotdraw"no SemanticScholar foram retornados 1.100 resultados, no IEEEXplore utilizando a mesma sentença, foram retornados 57 resultados;
- Observando o aspecto qualidade do projeto, o JHotDraw foi desenvolvido com o objetivo de exercitar padrões de projeto e concebido por um dos autores do livro Design Patterns (GAMMA et al., 1996) - Erich Gamma. Possui uma pesquisa online para a avaliação da ferramenta [3], porém, embora antigo os resultados indicam uma boa percepção de qualidade por parte dos usuários.

Enfim, esta Seção através das fases e atividades, procurou avaliar se de fato existiam indícios de qualidade no projeto de código aberto selecionado, e clareza na aplicação e entendimento nos resultados obtidos. Para esta dinâmica de avaliação, levantamos algumas questões que devem foram verificadas durante a execução das fases, sendo elas:

- **Questão 1**: É possível visualizar, caracterizar e avaliar o design do projeto, quando o único artefato disponível é o código-fonte?
- **Questão 2**: Como verificar a viabilidade em se manter, recuperar ou reconstruir o projeto do zero, visando futuras manutenções?
- **Questão 3**: É possível simular o custo associado ao estado atual do sistema e a economia caso sejam aplicadas ações de melhoria?
- **Questão 4**: É possível priorizar a correção de partes do sistema que contribuem para a melhoria do design, com base em aspectos de qualidade, observando as prováveis evoluções do projeto?

[3] https://www.jhotdraw.org/survey/survey$_r esults.html - em$11/2018

4.5.1 Coletar

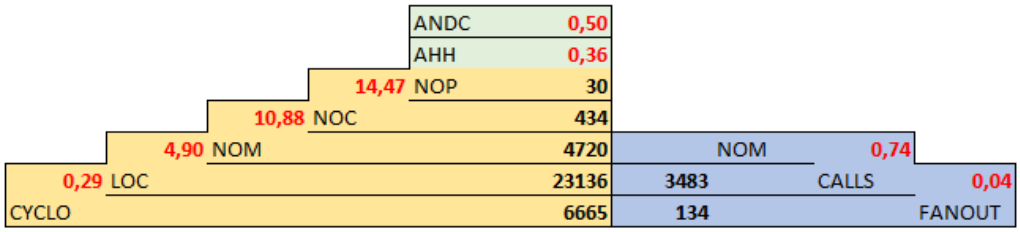
Para a execução desta fase, iniciamos a obtenção do código-fonte do projeto no endereço do repositório: svn://svn.code.sf.net/p/jhotdraw/svn/trunk/JHotDraw, com acesso de forma anônima. Nesta foram coletadas revisões dentre um conjunto de 104 partindo da revisão 2 a revisão 180, o número se justifica para facilitar a análise do aumento do volume de código-fonte construído e a evolução dos objetos do projeto.

Para exemplificar a aplicação do método, dependendo do cenário a ser verificado, será indicada a versão de revisão para fins de esclarecimentos.

4.5.1.1 Caracterização

Para a atividade de Caracterização, iniciamos com a elaboração da Pirâmide de Visão Geral, descrita na Seção 4.2.2, selecionando a revisão 180, conforme a apresentado na Figura 20.

Figura 20 – Pirâmide de Visão Geral - JHotDraw



A partir da observação da Figura 20, podemos afirmar os seguintes pontos:

- Observando o cálculo de produtividade conforme a Tabela 25, o projeto se enquadra em sendo do tipo Médio-Pequeno equivalendo a 16 pontos de produtividade;
- Verificando a Complexidade (CYCLO) temos um valor médio de 16 pontos de complexidade por classe, se considerarmos como referencial o levantamento de limites conforme a Tabela 17, o valor (WMC) está abaixo do menor valor máximo apresentado;
- Considerando o numero de LoC/método (MLOC), sendo cinco por classe, e utilizando a mesma Tabela do item anterior, podemos verificar que este valor também se encontra abaixo do menor valor ideal máximo sendo este o valor de trinta/método;
- Com relação ao número de operações/classe, sendo onze, e seguindo a mesma dinâmica de comparação, temos o valor de NOM em catorze/classe;
- Observando o número de classes/pacote, sendo este quinze, temos o valor de NOC considerado como regular, pois abaixo de onze seria o valor tido como bom/ideal;

- Avaliando outros três itens: AHH (DIT), FUN-OUT (CE) E CALLS (CA) sendo calculados como valores médios, estes apresentaram valores bem abaixo dos limites máximos da Tabela 17, para o ANDC não fizemos uma comparação de limites para esta métrica, por não constar mais de uma indicação para efeito de comparação na Tabela 17.

Enfim, observando somente esta Pirâmide de Visão Geral podemos concluir que todas as métricas proporcionais estão sendo consideradas abaixo do menor valor máximo permitido, considerando o comparativo de limites descritos na Tabela 17.

4.5.1.2 Propriedades do design

Na Figura 21, são apresentadas duas revisões a primeira e a última dentre o conjunto de 104 revisões, o que pode ser visto é que a médida em que a quantidade classes/interfaces (DesignSize) cresceram, proporcionalmente cresceram algumas outras propriedades, sendo elas: coesão, composição, polimorfismo mensagem, acoplamento e complexidade, corroborando assim, com a análise de correlação executada conforme a Figura 12.

Figura 21 – Atributos de qualidade por revisão - JHotDraw

Revision	Hierarchies	Abstraction	Encapsulation	Cohesion	Composition	Polymorphism	Messaging	Inheritance	DesignSize	Coupling	Complexity
2719	0.7015	0.8421	0.6327	0.343	88.0	183.0	958.0	0.8728	153.0	634.0	1902.0
2720	0.5046	0.3625	0.5244	0.2182	187.0	338.0	3697.0	0.7991	481.0	1766.0	6665.0

Observando a evolução do design do projeto na Figura 22, o gráfico apresenta no eixo X, os valores referentes às revisões, e no eixo Y, os valores referentes ao valor da métrica de qualidade calculada. Neste sentido, os valores foram normalizados para efeito de comparação.

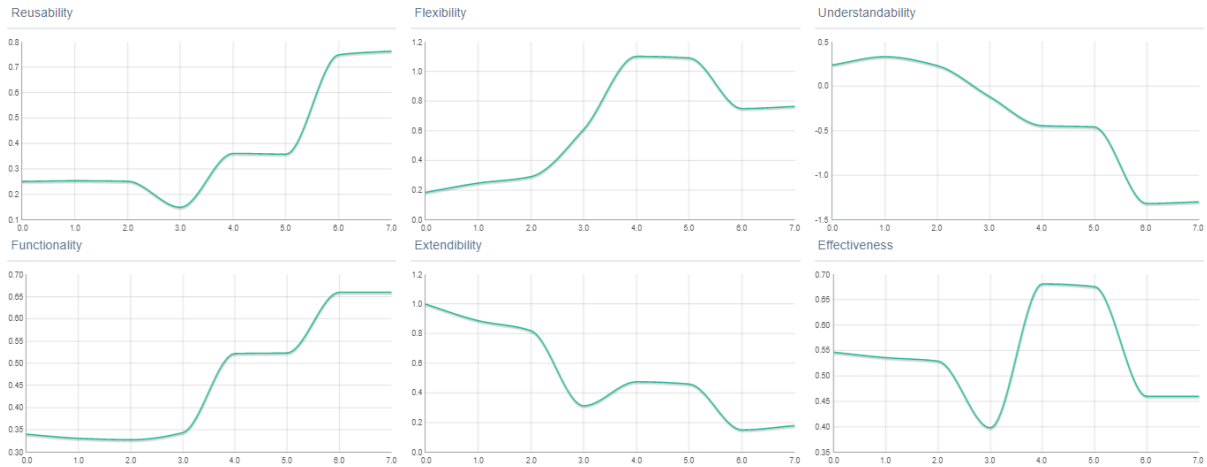
Para este projeto de código aberto, utilizando sete revisões com um intervalo de dez revisões entre elas, podemos verificar que conforme a evolução do projeto, as métricas de qualidade de Reusabilidade e Funcionalidade cresceram, enquanto o Entendimento e Extendibilidade reduziram, isto também pode ser verificado observando os resultados da Figura 21, onde o uso de hierarquias, abstração, encapsulamento e herança, fatores que contribuiam para uma melhora na qualidade do design, ficaram menores à medida em que o projeto evoluiu.

Já sobre as medidas de Flexibilidade, esta obteve um grau maior em revisões intermediárias e se estabilizou nas duas últimas revisões, para a Efetividade, esta ficou com o valor abaixo do calculo da primeira revisão do projeto.

4.5.1.3 Evolução de Não-Conformidades

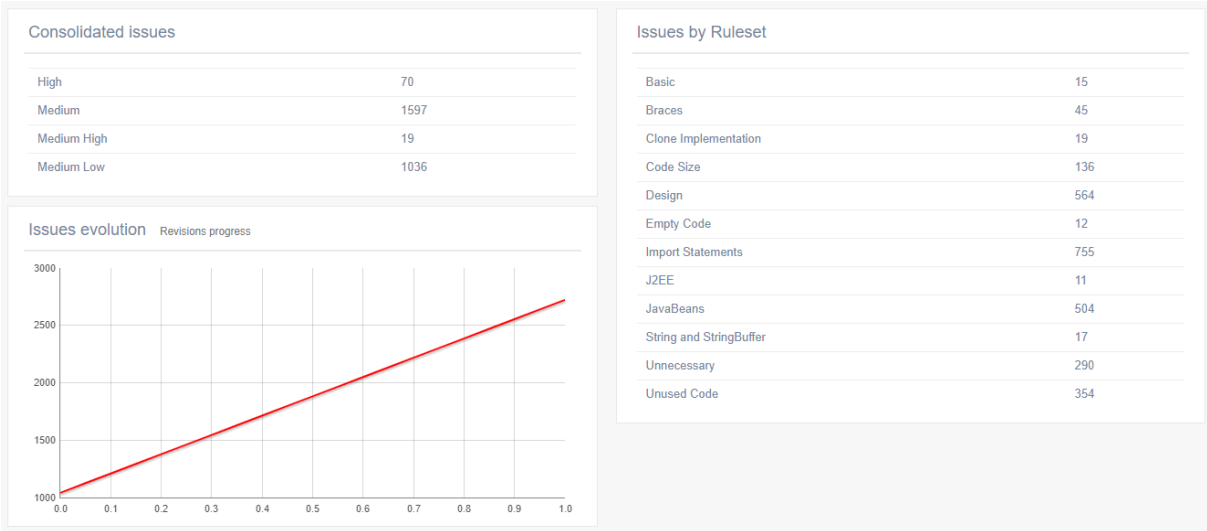
Para a análise de evolução de não-conformidades, na Figura 23 é exibida as informações sobre a última revisão obtida. Para efeito de comparação foi elaborado um comparativo

Figura 22 – Evolução das propriedades do design



de evolução considerando a primeira revisão do projeto.

Figura 23 – Evolução de não-conformidades do projeto - JHotDraw



Na Figura 24 é possível identificar que houveram evoluções de não-conformidades por categoria consideráveis, onde algumas afetam a qualidade do design como por exemplo o numero de declarações de import. Para as severidades, também pode ser visto uma variação nos número de médio-baixo e alta severidade.

4.5.1.4 Discussão

Encerramos a fase Coletar observando que os valores referentes à proporcionalidade descrita na Pirâmide de Visão Geral está conforme os limites descritos no estado da arte, as propriedades do design apresentaram uma variação proporcional ao crescimento dos objetos do projeto, porém, para a coleta de não-conformidades vimos que o crescimento denotou que não havia um ação contínua para a contenção no surgimento de novas não-conformidades para se manter o quantitavo estável.

Figura 24 – Comparação de evolução de não-conformidades do projeto - JHotDraw

Categoria

Basic	14	15	7%
Braces	312	45	-86%
Clone Implementation	8	19	138%
Code Size	32	136	325%
Design	281	564	101%
Empty Code	3	12	300%
Import Statements	26	755	2804%
J2EE	6	11	83%
JavaBeans	132	504	282%
String and StringBuffer	4	17	325%
Unnecessary	47	290	517%
Unused Code	176	354	101%

Severidade

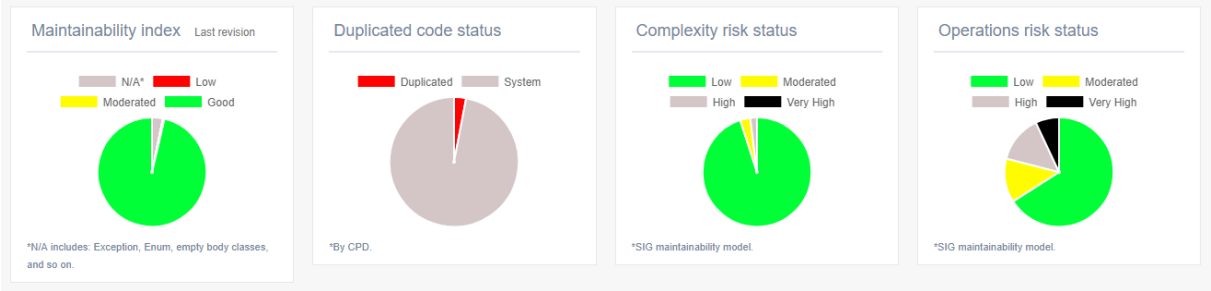
High	23	70	204%
Medium	928	1597	72%
Medium High	12	19	58%
Medium Low	78	1036	1228%

4.5.2 Avaliar

4.5.2.1 Manutenibilidade

A visão de manuteniblidade permite verificar o projeto considerando o índice de manutenibilidade proposto por (OMAN; HAGEMEISTER, 1992), onde foram identificadas duas classes com manutenbilidade classificadas como média, ou seja, se considerarmos 2 classes em um universo de 434 equivaleria a a 0,5% do total. Porém, se considerarmos o volume das classes em termos de linhas de código, ambas equivalem a 1184 Loc correspondendo a 5,12% do tamanho total do projeto.

Figura 25 – Visão da manuteniblidade - JHotDraw



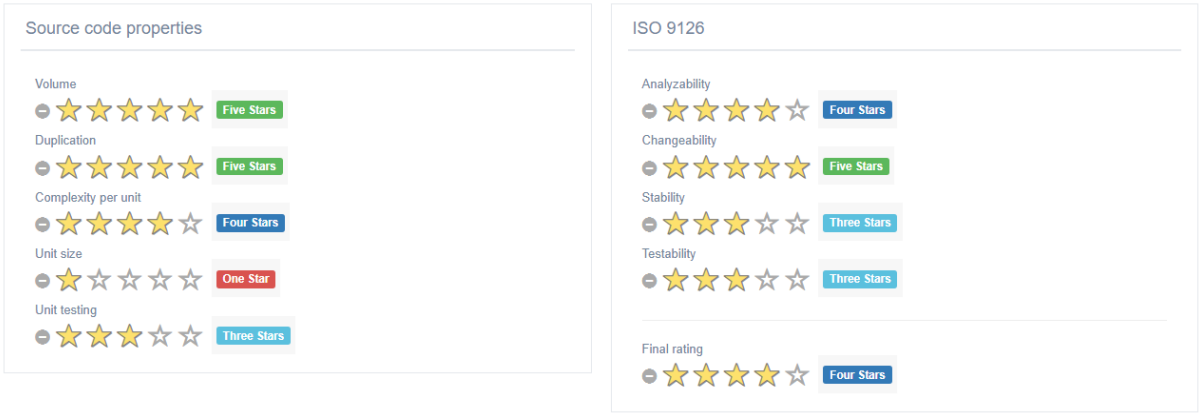
Além da visão clássica do parágrafo anterior, torna-se importante verificar a quantidade de duplicações (3%), tidos como ideal segundo (HEITLAGER; KUIPERS; VISSER, 2007).

Observando o risco de complexidade, vimos que existem somente duas classes com alta complexidade, tornando o trabalho de construção de testes mais complexos.

Observando o risco de complexidade das operações, vimos que existem trinta e quatro operações com muito alta, alta e moderada complexidade, tornando o trabalho de construção de testes mais complexos.

Para a definição do rating da manutenibilidade a Figura 26 apresenta do lado esquerdo, as propriedades do código-fonte, e do lado direito, o seu mapeamento conforme a ISO 9126, onde foi possível verifcar que o projeto possui uma avaliação de 4-estrelas, ou seja, uma avaliação próxima do ideal.

Figura 26 – Ranking da manutenibilidade - JHotDraw



4.5.2.2 Débito tecnico e juros

Ainda de forma prática, o projeto avaliado como 4 estrelas descrito na Seção 4.5.2.1, será utilizado como exemplo, considerando as seguintes informações:

- Tecnologia: Java;
- Linhas de Código: 23.136;
- Classificação do projeto: 4 estrelas;
- Como se trata de um projeto Java, o valor do PF equivale a 53 (número médio de instruções por ponto de função) segundo a tabela definido por (JONES, 1996);
- **Fator de Produtividade: 16**, dado que 23.136 LoC / 53 = 436, o que equivale a um projeto Médio-Pequeno conforme a tabela 25;
- O **Valor de Reconstrução** (VR) do projeto corresponde a 436 / 16 = 27 (homem/mês), o que equivaleria se consideramos somente uma pessoa a 27 x 21 dias, ou seja, uma pessoa reconstruiria o projeto em 567 dias.

Neste contexto, o Débito técnico da passagem para 5 estrelas seria de 27 (Valor de Reconstrução) * .40 = 10.8 (homem/mês), neste caso seria mais interessante manter o sistema, dado que, o Valor de Reconstrução é maior que o Débito técnico calculado.

Para o projeto em questão considerando uma taxa de manutenção anual de 15%, o valor do Juros seria a diferença entre 2.89 - 2.04 = **0.85 (homem/mes)**, conforme o cálculo abaixo:

$$EM(4 - Estrelas) = \frac{.15 * 27}{1.414} = 2.89 \tag{4.9}$$

$$EM(5 - Estrelas) = \frac{.15 * 27}{2} = 2.04 \tag{4.10}$$

Na Figura 27, verificamos a evolução do esforço em 10 anos onde estão informados os anos, a taxa de crescimento anual de 5%, o fator de manutenção de 15%, os fatores para o nivel 4 e 5 estrelas, sendo eles: o Valor de Reconstrução (VR), o Fator de Qualidade (FQ), o Esforço de Manutenção (EM), o Débito Técnico (DT) e o Juros. Ainda na Figura, dado o esforço em se passar o projeto de 4 para 5 estrelas o EM inicial só seria percebido no situação 5 estrelas em oito anos, e o Retorno do Investimento se daria no primeiro ano a uma taxa de 2%. Contudo, o que deve ser observado nesse cenário é se o projeto terá uma durabilidade que justifique uma ação com um retorno menor de 50% em 10 anos.

Figura 27 – Projeção de débito técnico e juros - JHotDraw

			4-ESTRELAS					5-ESTRELAS				
ANOS	TX. CRESC	FM	VR	FQ	EM	DT	JUROS	VR	FQ	EM	JUROS	ROI
1		0,15	27,28	1,414	2,893917963	10,912	0,847917963	27,28	2	2,046	0	0,022633704
2	0,05	0,15	28,644	1,414	3,038613861	11,4576	0,890313861	28,644	2	2,1483	0	0,069174956
3	0,05	0,15	30,0762	1,414	3,190544554	12,03048	0,934829554	30,0762	2	2,255715	0	0,113499958
4	0,05	0,15	31,58001	1,414	3,350071782	12,632004	0,981571032	31,58001	2	2,36850075	0	0,155714246
5	0,05	0,15	33,1590105	1,414	3,517575371	13,2636042	1,030649584	33,1590105	2	2,486925788	0	0,19591833
6	0,05	0,15	34,81696103	1,414	3,69345414	13,92678441	1,082182063	34,81696103	2	2,611272077	0	0,234207933
7	0,05	0,15	36,55780908	1,414	3,878126847	14,62312363	1,136291166	36,55780908	2	2,741835681	0	0,270674222
8	0,05	0,15	38,38569953	1,414	4,072033189	15,35427981	1,193105724	38,38569953	2	2,878927465	0	0,305404021
9	0,05	0,15	40,30498451	1,414	4,275634849	16,1219938	1,252761011	40,30498451	2	3,022873838	0	0,33848002
10	0,05	0,15	42,32023373	1,414	4,489416591	16,92809349	1,315399061	42,32023373	2	3,17401753	0	0,369980971

4.5.2.3 Discussão

Encerramos a fase de Avaliar verificando a manutenibilidade do projeto e analisando uma projeção de 10 anos para a manutenção do projeto verificando a evolução do débito ténico e do juros. Neste sentido, podemos verificar que o valor de reconstrução é maior que o valor do débito técnico e que o retorno do investimento para a passagem do projeto de 4 para 5 estrelas é baixo, devendo ser considerada a longevidade do projeto para justificar uma ação de melhoria com um custo de 10.8 homem/mês.

4.5.3 Direcionar

4.5.3.1 Priorização de correções - IPCNC

Como ponto inicial, para a seleção das classes a serem priorizadas são consideradas aquelas que possuem alguma evolucão em termos de acréscimo ou retirada de operações.

Tomando como exemplo a revisão 20, verificamos que está possui 147 classes, porém, considerando somente o número de classes que sofreram alterações em termos de operações, o quantitativo de classes recomendadas se reduz a treze.

A Tabela 27, exemplifica a quantidade de classes por revisão, bem como, a quantidade de recomendações de priorização. Vale ressaltar que as revisões foram selecionadas em intervalos que pudessem representar melhor o aumento no volume de classes do projeto.

Tabela 27 – Revisão - numero de classes x recomendações de priorização

Revisão	Número de classes	Recomendações de priorização
20	147	13
40	170	60
60	249	78
80	252	81
160	434	89

Para a verificação do índice de priorização de não-conformidades, a avaliação foi executada da seguinte forma:

- Definição do intervalo entre as revisões, neste sentido, foram selecionadas duas revisões base, onde estas são a base de comparação de modificações de classes em revisões futuras;
- Revisões de verificação, foram selecionadas as revisões que continham classes modificadas, verificando o histórico do repositório de controle de versão, até a próxima revisão tida como base, ou seja, a partir da revisão base 20, foram selecionadas todas as revisões que continham classes modificadas até a revisão base 40;
- Verificação da quantidade de classes modificadas na revisão versus a quantidade de classes recomendades para priorização, considerando o percentual de recomendação sobre o total modificado, conforme a Tabela 28.

Para um melhor esclarecimento sobre os resultados na Tabela 28, os títulos na colunas definem os seguintes pontos:

- Rev-B Revisão base, classes recomendadas para verificação nas revisões seguintes;
- NCl Número de classes constantes na revisão base;
- Rec Número de recomendações direcionadas pelo IPCNC;
- Rev-A Revisão avaliada;
- NCl-M Número de classes modificadas na revisão avaliada;

Tabela 28 – Recomendação x revisões

Rev-B	NCl	Rec	Rev-A	NCl-M	NCl-R	%Cl-R	%Cl-M
20	147	13					
			26	43	2	15%	5%
			32	43	2	15%	5%
			33	96	8	62%	8%
			35	87	7	54%	8%
			36	110	10	77%	9%
			40	20	3	23%	15%
40	170	60					
			42	49	23	38%	47%
			43	27	9	15%	33%
			44	2	2	3%	100%
			45	165	52	87%	32%
			46	3	2	3%	67%
			47	4	3	5%	75%
			48	10	1	2%	10%
			49	16	4	7%	25%

- NCl-R Número de classes recomendadas encontradas no conjunto de classes modificadas na revisão avaliada;
- %Cl-R Percentual de classes recomendadas encontradas dentre as classes modificadas na revisão avaliada;
- %Cl-M Percentual representativo de classes recomendadas dentro das classes modificadas na revisão avaliada;

Ainda neste contexto, é importante verificar se as classes recomendadas pelo IPCNC estiveram presentes em algumas das revisões avaliadas, e, se a ordem de priorização traria algum indício de que a classe teria um maior número de modificações com o passar das revisões.

Deste modo, a Figura 28 apresenta a ordem de priorização para a revisão base 20, onde constam 13 classes cuja a primeira classe na ordem aparece modificada em cinco revisões dentre a seis revisões avaliadas, conforme apresentado na Tabela 28.

O que podemos concluir é que a ordem de priorização não indica que a classe aparecerá de forma mais recorrente no conjunto de classes modificadas, porém, vimos que todas as classes priorizadas independentemente do número de ocorrências, constaram nas reviões avaliadas.

Figura 28 – Priorização x número de ocorrências

	RANK DE PRIORIZAÇÃO	LENOM	EENOM	HV	UPD	DCC	WMC	OCORRÊNCIAS	
DrawApplication	3,194541591	3,5	3,5	0,126293725	1	9	0,020478	5	83%
FigureAttributes	2,541885715	1	1	0,251314428	1	6	1	2	33%
StandardDrawingView	2,535668063	0,5	0,5	0,016260521	1	7	0,098874	5	83%
PolyLineFigure	2,306486199	2	2	0,09237332	1	6	0	1	17%
AttributeFigure	2,232638764	1	1	0,080042708	1	6	0,157895	1	17%
DecoratorFigure	2,056016098	1	1	0,031748698	1	5	0,652174	2	33%
CompositeFigure	1,960513330	3	3	0,167054085	1	5	0,298013	2	33%
ArrowTip	1,931795286	3	3	0,485507816	1	5	0,052632	1	17%
TextTool	1,899798996	2	2	0,287682072	1	5	0,055556	2	33%
AbstractFigure	1,870813625	1	1	0,055569851	1	5	0,084656	3	50%
ToolButton	1,677560071	0,5	0,5	0,105360516	1	4	0,48	4	67%
BorderDecorator	1,589504725	1	1	0,251314428	1	4	0,142857	2	33%
PaletteButton	1,424720507	3	3	0,405465108	1	3	0,571429	2	33%

4.5.3.2 Designação de membros do time

Para a análise desta Seção, selecionamos todas as revisões do projeto de código aberto partindo da revisão número 2 a 180, totalizando 104 revisões. Neste contexto, associamos as priorizações levantadas na Seção 4.5.3.1 para a revisão de número 20 com os membros do times utilizando o conceito de truck factor descritos na Seção 4.4.2.

Conforme a Figura 29, podemos verificar o autor de cada arquivo, a quantidade de mudanças na classe pelo autor e por outros membros do time, o grau de autoria (DOA) onde à medida em que surgem mais contribuições externas, o DOA é reduzido, e, os co-autores onde são indicados o nome e a quantidade de contribuições.

Sendo assim, vimos que todos os arquivos priorizados possui um único autor, porém, caso não seja possível a atuação do autor nas atividades de melhoria, na coluna co-autores é possível identificar quem mais contribuiu com a classe, podendo este ser um candidato no apoio do entendimento ou na atividade de refatoramento.

Figura 29 – Priorização x Truck Factor

		AUTOR	MUDANÇAS		DOA	CO-AUTORES
	RANK DE PRIORIZAÇÃO		AUTOR	OUTROS		
DrawApplication	3,194541591	jeckel	0	26	3,333036366	ricardo_padilha;2;dnoyeb;7;mrfloppy;17;
FigureAttributes	2,541885715	jeckel	0	6	3,766362842	mrfloppy;6;
StandardDrawingView	2,535668063	jeckel	1	18	3,609835088	ricardo_padilha;1;dnoyeb;4;mrfloppy;13;
PolyLineFigure	2,306486199	jeckel	0	6	3,766362842	mrfloppy;6;
AttributeFigure	2,232638764	jeckel	0	5	3,815845210	ricardo_padilha;1;mrfloppy;4;
DecoratorFigure	2,056016098	jeckel	0	7	3,723499265	mrfloppy;7;
CompositeFigure	1,960513330	jeckel	1	9	3,815870185	dnoyeb;1;mrfloppy;8;
ArrowTip	1,931795286	jeckel	0	3	3,945999510	mrfloppy;3;
TextTool	1,899798996	jeckel	0	7	3,723499265	mrfloppy;7;
AbstractFigure	1,870813625	jeckel	1	9	3,815870185	ricardo_padilha;1;mrfloppy;8;
ToolButton	1,677560071	jeckel	0	7	3,723499265	mrfloppy;7;
BorderDecorator	1,589504725	jeckel	0	5	3,815845210	mrfloppy;5;
PaletteButton	1,424720507	jeckel	0	5	3,815845210	mrfloppy;5;

4.5.3.3 Discussão

Encerramos a fase Direcionar apresentando um modelo para a priorização na correção de não-conformidades, avaliando se as classes indicadas foram trabalhadas ao longo das revisões seguintes. Neste sentido, vimos que todas as classes foram modificadas pelo menos por uma vez e que a ordem na priorização das classes não indicaria um percentual maior em numero do modificações.

Para a designação de membros do time para a atuação em atividades de melhoria, apresentamos o uso do truck factor, onde associamos à classes aos seus respectivos autores, indicando possíveis membros do time que possa substituir o autor baseado no número de alteração por classe, ou seja, na primeira classe priorizada, DrawApplication, o autor jeckel poderia ser substituído por mffloppy, dnoyeb e/ou ricardo_padilha nessa ordem de importância.

4.5.4 Implementar

Para esta fase, dado que esta tese desenvolveu uma ferramenta de suporte ao uso do método conforme o Apêndice A, foi elaborado um comparativo com a ferramenta SonarQube versão 7.3. Para isto, foi selecionada a revisão do JHotDraw 180 conforme a Figura 30.

Figura 30 – Comparativo Sonar x RADAR

JHotDraw	
Revisão	180

	SONAR	RADAR	
LoC	28145	23136	
Complexidade	6671	6665	
Issues	2836	2722	
Duplicação	5,60%	3%	
Debito Técnico (EM)	78	61	
Juros	0	17,64	
Classificação	A	Quatro	
Priorização de correção	0	71	
Criticidade - Blocker	21	70	Alta
Criticidade - Critical	136	19	Média-alta
Criticidade - Major	1700	1597	Média

Neste contexto, destacamos os itens conforme se segue:

- A ferramenta SonarQube não apresentou algo similar à Pirâmide de visão geral proposta nesta tese, como um meio de caracterização considerando limites e métricas proporcionais;

- Observamos uma diferença no número de Linhas de Código por volta de 5000 linhas, isso se justifica pois a RADAR utiliza como contagem somente as linhas de código dento das operações;
- Para o item complexidade, existe uma pequena diferença de 0,05 em termos de proporcionalidade entre Complexidade/LoC;
- Para o item issues, a diferença se apresenta em 0,01 novamente, considerando a divisão entre Issues/LoC e sua difenreça entre os projetos;
- A duplicação verificada pelo Sonar se apresentou em quase o dobro com 5,6%;
- O Dèbito técnico (Esforço de Manutenção - EM) apresenta uma diferença de 3 dias, porém, o Juros não é calculado pelo SonarQube;
- O SonarQube classificou o projeto com o nota máxima "A"segundo o SQALE, na RADAR a classificação atingiu o valor de 4 estrelas, ou seja, ainda existe mais um nível a se atingir;
- Para a priorização de correção, a RADAR apresentou 71 classes como priorizadas em um universo de 481 classes;
- Sobre a criticidade das não-conformidades encontradas em ambas as ferramentas, o objetivo não foi compará-las dado que a categorização se apresenta de forma diferente com relação às issues, mas, apresentar que tal categorização se econtravam presentes em ambas;
- A ferramenta SonarQube não apresentou uma projeção de custos de débito técnico e juros, permitindo simular taxa de crescimento, manutenções e conhecimento técnico do time.

Por fim, encerramos esta Seção apresentando a possiblidade de uso da RADAR, a qual apresentou mais informações agregadas para o apoio à tomada decisão que o SonarQube neste comparativo.

4.6 SUMÁRIO

Este Capítulo apresentou o método proposto por esta tese, seguindo fluxo de atividades apresentado na Figura 7, tendo como apoio os estudos definidos pelo estado da arte e da pesquisa levantados nos Capítulos 2 e 3. Após a definição foi avaliado um projeto de código aberto para efeito de aplicação do método, com uma breve discussão sobre os resultados obtidos. Por fim, o que se tentou confirmar foram as declarações descritas na Seção 1.3.1, sendo algumas delas respondidas pela aplicação do método, como se segue:

- Objetivo. Mesmo com a existência de uma variedade de modelos de avaliação e métricas de software, foi possível de forma segura avaliar e direcionar ações em nível estratégico, abstraindo a complexidade teórica para o âmbito indutrial por meio de indicadores;

- **Discussão**. Neste contexto esta tese verificou o estado da arte em métricas e mapeou o maior quantitativo de ocorrências considerando a popularidade de cada métrica, verificou o estado da arte em limites de métricas de software, permitindo que o uso de referenciais facilitem o entendimento sobre a situaçao do código-fonte para fins de caracterização, mapeou métricas clássicas em substituição a algumas definidas pelo modelo de avaliação de qualidade do design OO selecionado.

- Objetivo. Considerando o código-fonte como o principal artefato do projeto, o entendimento sobre a qualidade do software tendo como base a análise estática, trouxe a percepção para os times de que métricas são fatores para a avaliação dos projetos e não das pessoas.

- **Discussão**. Para este fim, o objetivo consistiu em verificar a evolução do código-fonte em termos de caracterização, não-conformidades, design e situação de manutenibilidade. Além do indicativo sobre possíveis membros do time capazes de atuar em melhorias do projeto com base em seu histórico de autoria e contribuição para as classes. Neste contexto, não houveram verificações sobre a quantidade de não-conformidades geradas por indivíduos em atividade de codificação.

- Objetivo. O direcionamento de ações de melhoria de forma gradual, contribuiu para o uso racional do tempo e do esforço dos recursos envolvidos em atividades de construção, trazendo uma percepção de valor para os gestores do projeto em atividades de melhoria e refatoramento.

- **Discussão**. Isto pode ser visto por meio da proposição de um índice para a priorização de correções com base em um modelo de regressão, onde as priorizações puderam ser percebidas nas revisões seguintes, por meio das classes modificadas. Também pode ser verificado por meio da indicação de membros para a execução de atividades de melhoria no código-fonte.

- Objetivo. O esforço na aplicação do método foi abstraído pela automatização da coleta de dados e o monitoramento dos projetos de forma contínua, sendo possível direcionar ações com base em indicadores, projeções de custos, classificação e melhoria contínua em partes do sistema.

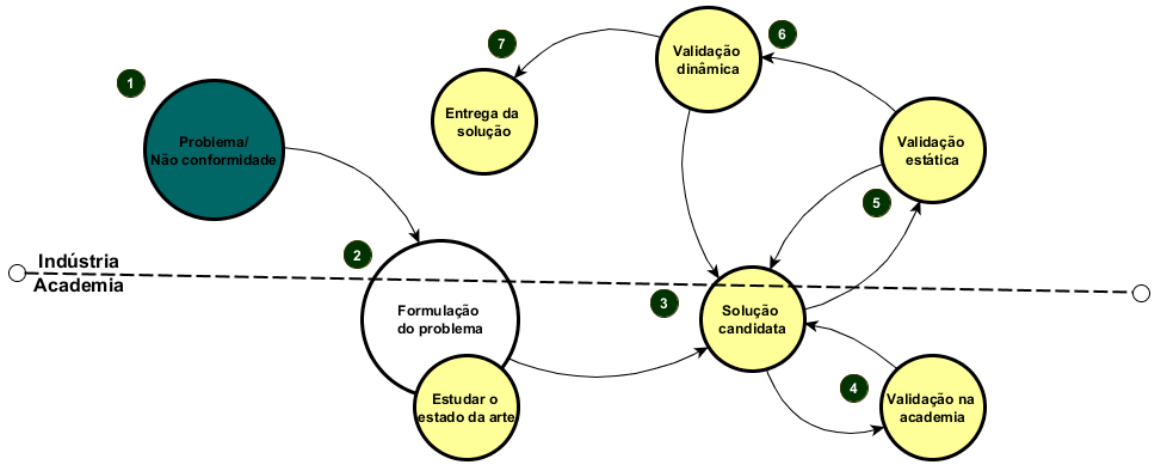
- **Discussão**. Para esta objetivo foi desenvolvida uma ferramenta descrita no Apêndice A, porém, é possível o uso de ferramentas de código aberto para a coleta dos

dados e o uso de planilhas eletrônica para a execução dos cálculos descritos no método. Neste sentido, definimos a fase de Implementação como um apoio à execução do método de forma não exclusiva.

5 AVALIAÇÕES PRÁTICAS DO MÉTODO

Para as avaliações do método proposto por esta tese, seguimos a abordagem proposta por (GORSCHEK et al., 2006), onde o principal foco é a transferência de tecnologia com a cooperação entre a academia e a industria. Deste modo, a Figura 31, se apresenta como um direcionador, sendo aplicado para efeito de comprovação em três casos práticos cobrindo os segmentos de Mídia, Telecomunicações e Financeiro.

Figura 31 – Modelo de transferência de tecnologia (GORSCHEK et al., 2006)



Seguindo o fluxo descrito na Figura 31, detalharemos a aderência da proposta conforme os pontos destacados no modelo:

- **1. Identificar potenciais áreas de melhorias baseado em necessidades da indústria**. Verificamos tal necessidade por meio de entrevistas com o gestores de projetos da consultoria, onde foi identificada a dificuldade em se entender o nivel de qualidade dos projetos de software baseando-se unicamente em um conjunto de métricas, cujo o entendimento é mais comum em times de desenvolvimento de software. Tal necessidade impedia a tomada de decisões e gerava incertezas na precificação de novas propostas de desenvolvimentos, dado que, as propostas adicionavam percentuais de contigência que aumentava o preço e diminuia a competitividade da consultoria.

- **2. Formulação de uma agenda de pesquisa**. Elaboramos uma revisão sistemática conforme o Capítulo 2, um levantamento da fundamentação teórica conforme o Capítulo 3 e uma avaliação e comparação dos trabalhos relacionados conforme o Capítulo 6.

- **3. Formulação da solução candidata**. Neste sentido, esta tese propôs um método sistemático baseando-se em conceitos e métricas existentes na academia, estabele-

cendo o entendimento em níveis operacionais e estratégicos. Como resultado foram propostas visões de analise qualidade baseado no paradigma orientado a objetos, manuteniblidade baseado na ISO-9126, evolução de não-conformidades, caracterização (tamanho e distribuição de classes e objetos), evolução de débito técnico e juros, bem como, um índice para a priorização de correções de não-conformidades para direcionamento dos times.

- **4. Validação na academia**. Neste item não foram executadas avaliações em laboratórios com alunos da Universidade, as avaliações foram executadas em projetos reais, dado que, existia a demanda por um cliente da consultoria exigindo uma forma de visualizar a qualidade dos seus projetos alinhado com os propósitos desta tese. Sendo assim, a avaliação foi apresentada a um cliente real.

- **5. Execução de validação estática**. Se constitui na apresentação da solução candidata na indústria, coletando os feedbacks dos participantes. Neste sentido, tivemos por volta de 31 e-mails contendo discussões sobre a solução, com representantes do segmento da industria de Telecom, 3 reuniões presenciais para a apresentação da proposta e coleta de feedbacks. Do staff envolvidos nas discussões podemos considerar: 4 analistas de sistemas, 2 arquitetos, 1 gerente e 1 gerente senior.

- **6. Executar validações dinâmicas** Conforme a Tabela 29, nas avaliações aplicadas em projetos reais, houveram tomadas de decisão tanto de parada e reconstrução (ID 14), quanto de não-redução em valores de absorção de legado (ID 20), nos demais houveram discussões dado o ROI%/A, que representa o percentual de RoI em anos, se valeria a pena uma ação de refatoramento ou continuidade com melhorias pontuais.

- **7. Entrega da solução** Neste ponto, a ferramenta resultante desta tese constante no Apêndice A está sendo aplicada ao *pipeline* de integração contínua de projetos, além disso, com a extração os resultados são elaboradas visões periódicas para reuniões sobre o tema "Qualidade dos projetos", permitindo um melhor direcionamento sobre as próximas ações a serem endereçadas.

Para um melhor entendimento sobre a Tabela 29, segue o entendimento sobre as legendas:

- IND-Industria, neste caso foram avaliados projetos nas industrias de Financial, Media e Telecom;

- ORG=Origem, foram mapeados projetos D=Consultoria, L=Legado e DF=Consultoria Offshore, neste caso, filiais da mesma consultoria em outros países.

- LoC, tamanho do projetos em linhas de código;

Tabela 29 – Avaliações executadas com a aplicação do método

IND	ID	ORG	LoC	DUP	MFR	VR	VM	ROI%/A
TEL	1	D	15181	9%	4	16	6	5-10
TEL	2	L	7796	1%	4	8	3	1-10
TEL	3	L	30815	7%	3	36	27	11-7
TEL	4	L	15378	6%	3	16	12	8-8
TEL	5	L	4776	2%	4	5	2	1-10
TEL	6	L	35813	9%	3	48	36	2-7
TEL	7	L	11226	4%	3	12	9	2-7
TEL	8	L	2877	2%	4	3	1	1-10
TEL	9	L	9184	3%	3	10	7	2-7
TEL	10	L	69465	5%	3	109	82	2-7
TEL	11	L	1903	29%	3	2	1	2-7
TEL	12	L	40779	31%	3	55	41	2-7
TEL	13	L	2715	0%	5	3	0	0-0
MED	14	DF	5377	21%	2	6	7	0-0
FIN	15	D	47765	6%	3	64	48	8-7
FIN	16	D	22473	11%	4	27	10	3-10
FIN	17	D	11839	11%	3	12	9	2-7
FIN	18	D	67905	17%	3	107	80	2-7
FIN	19	D	56353	16%	3	76	57	17-7
FIN	20	L	81815	45%	3	127	96	6-7

- DUP=Duplicidade, percentual de duplicidade de código;
- MFR= Ranking de Manutenibilidade, baseado na ISO-9126;
- VR= Valor de Reconstrução;
- VM = Valor de Manutenção;
- ROI%/A, apresenta o percentual de retorno do investimento a partir de quantos anos.

5.1 CASO PRÁTICO: LPE - MÍDIA

O LPE foi um dos projetos construídos dentro da consultoria citada neste trabalho, cujo o desenvolvimento se deu utilizando o Oracle Application Framework (OAF) [1], um framework proprietário desenvolvido pela Oracle Corporation para o desenvolvimento de aplicações na Suíte Oracle E-Business.

[1] http://www.oracle.com/technetwork/testcontent/s281402-sara-woodhull-128601.pdf, acessado em 23/07/2018

5.1.1 Problema estudado

O início do projeto se deu em outro site da consultoria localizado em Manila - Filipinas, com o repasse do projeto para a continuidade da construção e manutenção em Recife - Pernambuco - Brasil, notou-se por parte da gestão do projeto um alto esforço na execução das atividades, com um consumo de horas além da expectativa comprometendo o orçamento previsto para o projeto.

5.1.2 Contexto onde a pesquisa foi aplicada

Um projeto real para uma empresa com atuação no segmento de comunicação e mídia.

5.1.3 Objetivos da avaliação

De forma geral, identificar quais os pontos do projeto geravam um maior consumo de tempo e esforço no andamento das atividades. Sendo guiados pelas seguintes questões:

- Qual a qualidade do projeto do ponto de vista técnico, considerando somente o código-fonte?
- Qual a expectativa de custos em termos de manutenção evolutiva do projeto?

5.1.4 Seleção dos participantes

Este caso prático foi selecionado de forma individual, e, dentro deste contexto houve um grupo de pessoas envolvidas na solicitação e avaliação da demanda, sendo estes:

- Diretor Assistente
- Gerente de projetos senior
- Gerente de arquitetura de software
- 2 Gerentes de projeto assistentes
- 1 Analista de sistemas

5.1.5 Procedimento de coleta de dados, análise e validação

O staff demandante solicitou ao site de Manila-Filipinas todo o código-fonte relacionado ao projeto, a documentação não foi enviada pelo fato de sua não-existência. Considerando somente o código-fonte, os critérios analisados foram:

- Complexidade;
- Legibilidade;

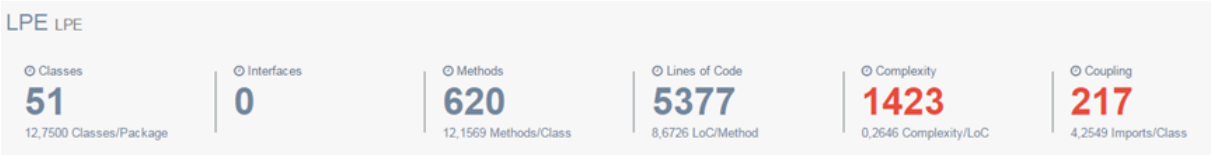
- Duplicidade;
- Manutenibilidade;
- Não-conformidades de código;
- Dívida técnica.

Com relação a análise e validação, seguimos os procedimentos descritos pelo método proposto nesta tese.

5.1.6 Descrição do passo a passo da execução

Para a execução, foi obtido o código-fonte fornecido pelos demandantes e por meio da ferramenta RADAR (resultante desta tese), inserida as informações na base de dados. A seguir serão apresentadas algumas visões avaliadas.

Figura 32 – LPE-Visão Geral



A Figura 32 apresenta a ausência de abstração (Interfaces – 0) e intenso uso de classes concretas, reduzindo a flexiblidade do sistema para adequações futuras.

Figura 33 – LPE-Complexidade / Legibilidade

O gráfico na Figura 33(A), apresenta o percentual de linhas de código (SLoC) do projeto, comprometidos por métodos (operações) e sua complexidade:

- 25% Muito alta;

- 17% Alta;
- 13% Moderada;
- 45% Baixa;

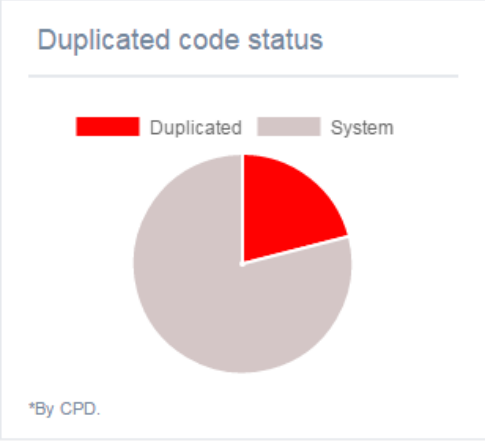
Conclusão:
- 42% de SLoC do projeto estão comprometidos com complexidades > Moderada.

O gráfico na Figura 33(B), apresenta o percentual de operações, com uma quantidade excessiva de SLoC comprometendo a legibilidade do código:

- 58% Muito alta;
- 10% Alta;
- 3% Moderada;
- 29% Baixa;

Conclusão:
- 68% de SLoC do projeto estão comprometidos com operações longas > Moderada.

Figura 34 – LPE-Duplicidade



Na Figura 34, 20.7% do código está comprometido pela duplicidade (copiar-colar). Deste modo, SLoC = 5377 * .207, equivale a aproximadamente 1.113 SLoC.

A Figura 35 apresenta 23 classes do sistema com boa manutenibilidade e somente 1 classe com baixa manutenibilidade.

Na Figura 36, podemos verificar os seguintes pontos:

- Volume de código-fonte baixo;
- Alta duplicidade;
- Alta complexidade por operações;

Figura 35 – LPE-Manutenibilidade

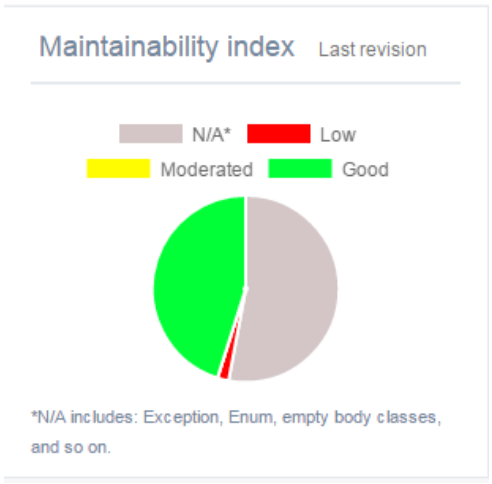
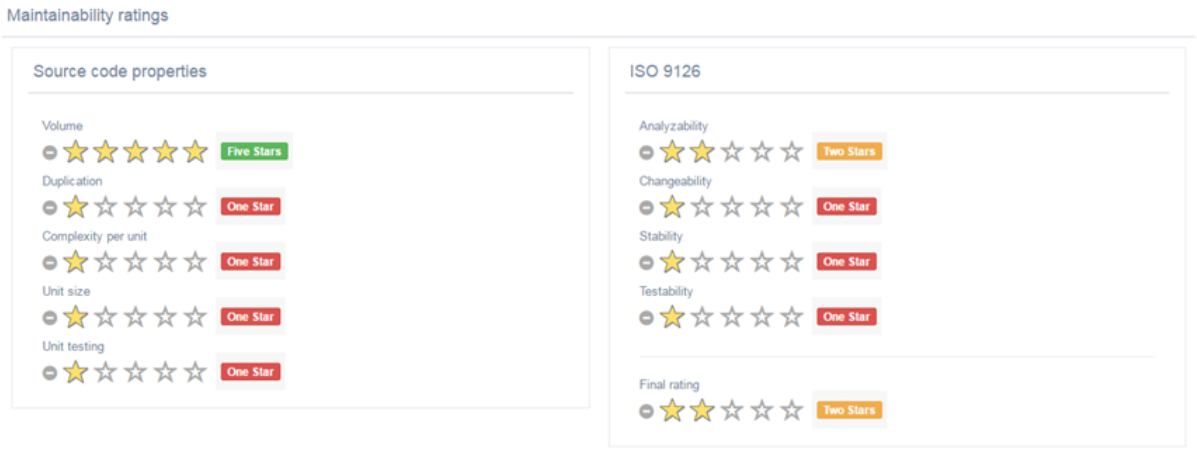


Figura 36 – LPE-Manutenibilidade (Ratings)

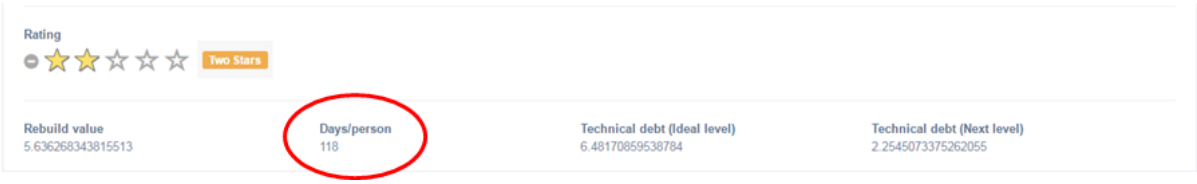


- Operações longas predominantes;
- Ausência de classes de testes.

Figura 37 – LPE-Não conformidades

A Figura 37 apresenta um total de 510 não-conformidades.

Figura 38 – LPE-Divida tecnica



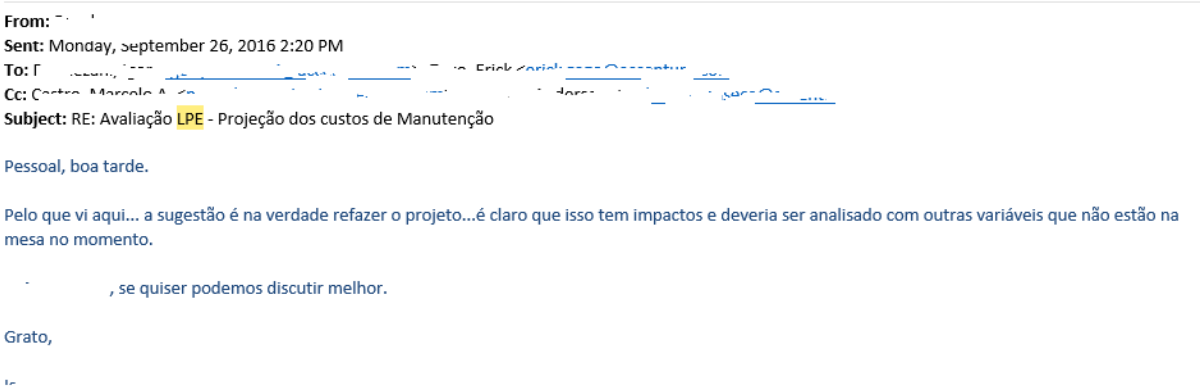
A Figura 38, apresentou os seguintes pontos:

- O custo de reconstrução do projeto equivale a 118 dias/pessoa;
- O custo para melhoria do projeto para o nível 3-Star equivale a aprox. 47 dias/pessoa.
- O custo para melhoria do projeto para o nível 5-Star equivale a aprox. 136 dias/-pessoa.

5.1.7 Avaliação dos resultados

Ao final da avaliação foi divulgado para os demandantes que o custo para a reconstrução do sistema seria mais barato do que mantê-lo. Com o esclarecimento dos resultados do trabalho, foram recebidos dois e-mails confirmando o entendimento, e formalizando o reconhecimento da abordagem pelo Diretor da TI, um profissional com mais de 20 anos de empresa e experiência em uma considerável gama de tecnologias, conforme as Figura 39 e 40.

Figura 39 – LPE-Feedback



From:
Sent: Monday, September 26, 2016 2:20 PM
To:
Cc:
Subject: RE: Avaliação LPE - Projeção dos custos de Manutenção

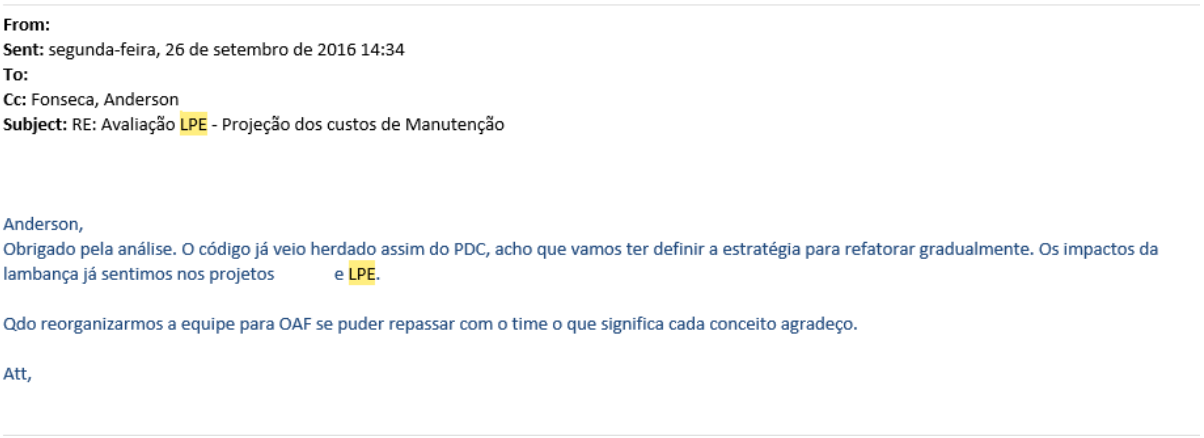
Pessoal, boa tarde.

Pelo que vi aqui... a sugestão é na verdade refazer o projeto...é claro que isso tem impactos e deveria ser analisado com outras variáveis que não estão na mesa no momento.

, se quiser podemos discutir melhor.

Grato,

Figura 40 – LPE-Reconhecimento



From:
Sent: segunda-feira, 26 de setembro de 2016 14:34
To:
Cc: Fonseca, Anderson
Subject: RE: Avaliação LPE - Projeção dos custos de Manutenção

Anderson,
Obrigado pela análise. O código já veio herdado assim do PDC, acho que vamos ter definir a estratégia para refatorar gradualmente. Os impactos da lambança já sentimos nos projetos e LPE.

Qdo reorganizarmos a equipe para OAF se puder repassar com o time o que significa cada conceito agradeço.

Att,

Buscando avaliar se a recomendações direcionadas pelo método foi seguido, sobre reconstruir o projeto, o retorno obtido foi que por se tratar de um projeto que já tem histórico

de insucessos, o direcionamento seria manter o construção atual e avaliar oportunidades de refatoramento de forma oportunista.

5.2 CASO PRÁTICO: BLODES - TELECOMUNICAÇÕES

O Sistema de Bloqueio e Desbloqueio (BLODES) foi um dos projetos construídos sob um chamado guarda-chuva de modernização e manutenção de sistemas chamado SOPHIA em uma das mais renomadas empresas de telefonia do país. Neste sentido, o contexto deste projeto era o de controlar o bloqueio e desbloqueio de contas em canais de telefonia e internet em nível nacional.

Na época, a empresa se encontrava em um processo de fusão com a aquisição de outra companhia.

5.2.1 Problema estudado

No inicio deste projeto foi escolhido um framework para a validação das regras e o time de desenvolvimento possuia pouca experiencia tanto no negócio quanto na tecnologia. De certo modo, a qualidade das entregas ficou comprometida e causou rejeição por parte do cliente.

Como o processo de construção se deu por meio da consultoria, pessoal da empresa adquirida e da empresa adquirente, o Diretor da empresa adquirente começou a questionar a qualidade das entregas, pois de fato, não lhe foi apresentado evidências de como um projeto com tamanha responsabilidade poderia suportar as demandas ao entrar em produção, o que estava lhe causando insegurança em dar o aceite.

5.2.2 Contexto onde a pesquisa foi aplicada

Um projeto real para uma empresa com atuação no segmento de telecomunicações.

5.2.3 Objetivos da avaliação

De forma geral, identificar qual a situação do projeto em termos de qualidade considerando aspectos de design e manutenibilidade.

- Qual a qualidade do projeto em termos de design e suas evoluções?
- Qual a situação de manutenibilidade do projeto?

5.2.4 Seleção dos participantes

Este caso prático foi selecionado de forma individual, e, dentro deste contexto houve um grupo de pessoas envolvidas na solicitação e avaliação da demanda, sendo estes:

- Gerente de projetos senior
- Gerente de arquitetura de software

- 1 Gerente de projeto assistentes
- 2 Analista de sistemas

5.2.5 Procedimento de coleta de dados, análise e validação

O código-fonte de todos os projetos eram versionados no SVN do próprio cliente, sendo possível selecionar 12 revisões com intervalos quinzenais, deste modo, foi possível averiguar dados mais consistentes em termos de evolução do código. Observando a fase de Coleta desta tese, os números do projeto são os seguintes:

- Numero de classes: 250
- Interfaces: 9
- Operações: 1941
- Linhas de código: 15181
- Complexidade 12/classe
- Acoplamento: 7 dependencias/classe

5.2.6 Descrição do passo a passo da execução

Novamente, para a execução, foi obtido o código-fonte fornecido pelos demandantes e por meio da ferramenta RADAR (resultante desta tese), inserida as informações na base de dados. Com os resultados gerados pela ferramenta, destacamos a visão de evolução do design e rating de manutenibilidade, com o intuíto de exercitar outros aspectos do método proposto por esta tese.

Os gráficos da Figura 41, apresentam uma visão sobre a evolução do design considerando 6 aspectos. Observando o quesito Extendibilidade, que se refere a facilidade em se adicionar novas funcionalidades considerando o design atual, foi identificado um decréscimo desta facilidade, bem como, no gráfico de Entendimento, houve um aumento na dificuldade de se compreender o software.

Observando os atributos de qualidade conforme a Figura 42, vimos que o projeto se encontrava em situação de manutenibilidade 4 estrelas, ou seja, próximo do ideal. Estes números foram alcançados depois de um trabalho de refatoramento com a aplicação de boas práticas e de padrões de projeto.

5.2.7 Avaliação dos resultados

Ao final da avaliação foi divulgado para o Diretor de TI os resultados via *conference call*, onde informamos algumas questões sobre a evolução do design como a redução da

Figura 41 – BLODES-Visão Design

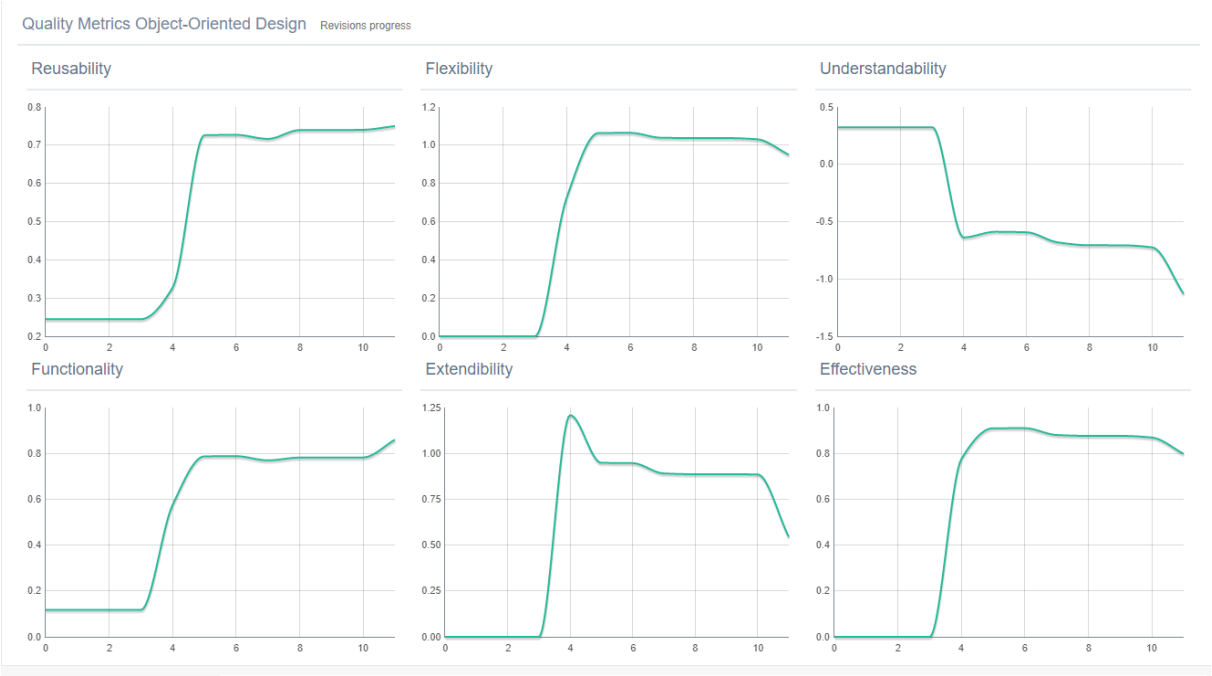
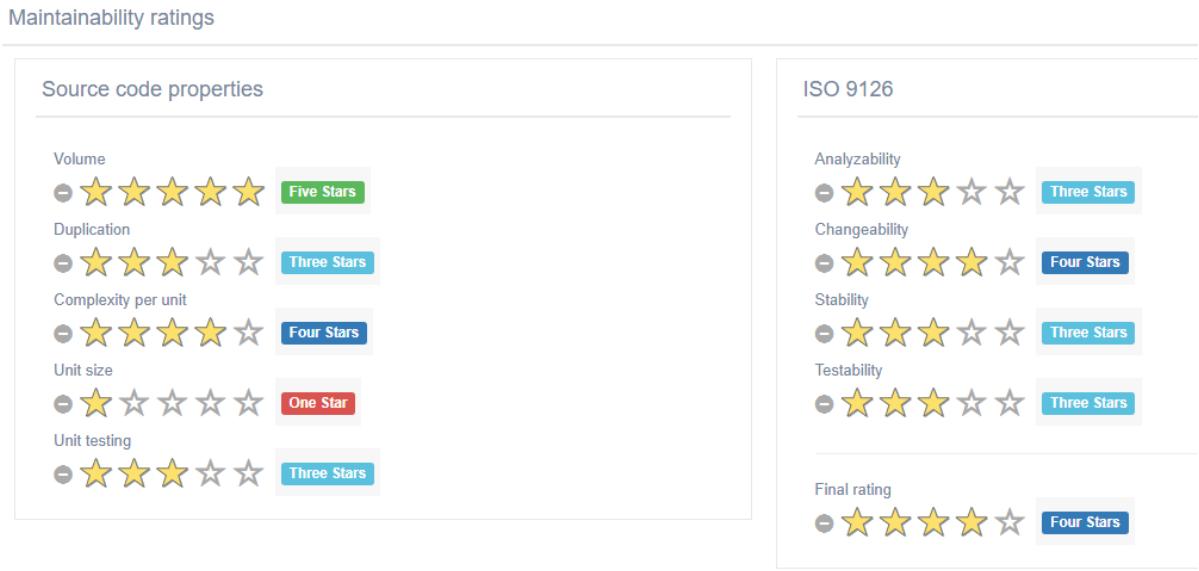


Figura 42 – BLODES-Manutenibilidade



flexibilidade, neste sentido, o Diretor TI confirmou que o aumento do acoplamento e da profundidade na árvore de herança se deu devido à sua interferência no modelo de construção.

Contudo, o cliente entendeu o valor da proposição e solicitou o levantamento dos projetos de forma geral que estavam sob responsabilidade da consultoria, onde ao final seria apresentado os resultados ao time de consultores do Cliente como uma abordagem educativa.

5.3 CASO PRÁTICO: NPC - FINANCEIRO

A Nova Plataforma de Câmbio (NPC) se caracteriza pela unificação de diversos sistemas com diferentes tecnologias e linguagens cujo o negócio tem como escopo a compra e venda de moeda estrangeira regulados pelo Banco Central do Brasil.

Este projeto, desenvolvido na plataforma Java, tem como cliente uma das principais instituições bancárias do país e um staff de 200 profissionais, distribuidos em 3 estados, seguindo a metodologia ágil.

Ainda neste contexto, devemos considerar o projeto com previsão de execução de 7 anos, sendo a primeira parte, chamada de Onda-1 iniciada em 2012, e a segunda parte, Onda-2 iniciada em 2016.

5.3.1 Problema estudado

Com o início da Onda-2, houve também a contratação de uma nova consultoria para a execução do projeto. Juntamente com este novo contrato, houve também a solicitação de continuidade do legado desenvolvido na Onda-1 por outro fornecedor, ou seja, a nova consultoria deveria manter o projeto anterior enquanto desenvolveria o projeto atual.

Neste âmbito, também houve uma mudança no processo de desenvolvimento (Onda-1 -> Onda-2) para uma abordagem ágil, utilizando o SAFE [2], porém, o modelo de cálculo de esforço e precificação sobre a entregas continuaram no formato de pontos de função.

O problema a ser avaliado surgiu quando a contratante sugeriu que ambas as 'Ondas' fossem desenvolvidas e mantidas cada um com seu valor de esforço, ou seja, o preço praticado na Onda - 1 continuaria o mesmo independente da mudança de fornecedor, argumentando que se os frameworks e tecnologias eram iguais, o esforço também seria igual.

Neste sentido, o ponto em questão foi se a qualidade dos projetos eram iguais ou se existiam variações significativas que inviabilizavam a solicitação da contratante.

5.3.2 Contexto onde a pesquisa foi aplicada

Um projeto real para uma empresa com atuação no segmento financeiro.

5.3.3 Objetivos da avaliação

De forma geral, identificar qual a situação do projeto em termos de qualidade considerando aspectos de manutenibilidade.

- Qual a qualidade dos projetos da Onda-1 e Onda-2 em termos de manutenibilidade?

[2] https://www.scaledagileframework.com/ - acessado em 08/2018.

5.3.4 Seleção dos participantes

Este caso prático foi selecionado de forma individual, e, dentro deste contexto houve um grupo de pessoas envolvidas na solicitação e avaliação da demanda, sendo estes:

- 3 Gerente de projetos senior
- 1 Gerente de arquitetura de software
- 1 Gerente de projeto assistente
- 2 Analistas de sistema

5.3.5 Procedimento de coleta de dados, análise e validação

O código-fonte de todos os projetos da Onda-2 eram versionados no repositório Git da consultoria, e o código da Onda-1 enviado pelo Cliente em um arquivo compactado. Deste modo, foi possível averiguar dados mais consistentes em termos de evolução do código conforme a Tabela 30, onde são apresentadas métricas primitivas para efeito de caracterização.

Tabela 30 – Quadro comparativo Onda-1 x Onda-2

Projetos/Itens	Onda-1	Onda-2
Número de classes	1.356	2.228
Interfaces	52	118
Operações	21.141	16.143
Linhas de Código	81.815	67.905
Complexidade	26.361	21.229
Acoplamento	6.469	9.814

5.3.6 Descrição do passo a passo da execução

Como o ponto discutido comtempla a questão da manutenibilidade, a Figura 43, apresenta uma comparação entre os projetos da Onda-1 (A) e Onda-2 (B). Na análise do Índice de Manuteniblidade, o (A) apresenta 12 classes com baixa manutenibilidade equivalendo 28,41% do tamanho do projeto em LoC, enquanto que o (B) apresenta 4 classes equivalendo a 13% do tamanho do projeto em LoC. Deste modo, vemos uma diferença de quase o triplo de classes, apresentando uma variação de 66.66% entre os projetos.

No quesito Duplicidade, verificamos que o (A) possui um percentual de 45% e o (B) 17%, neste sentido é possível constatar que aproximadamente metade do código-fonte de (A) possui duplicação, dificultando manutenções futuras. Comparando ambos os projetos, novamente estão com diferenças de quase o triplo em termos percentuais, apresentando uma variação de 62,22%.

Figura 43 – Onda-1 x Onda-2 - Visão Manutenibilidade

Com relação a complexidade distribuída em operações, o projeto A apresenta 25% de operações com alta complexidade, para o projeto B, foi identificado 11%, uma diferença de pouco mais do dobro neste item.

No quesito Débito técnico e Juros, a Tabela 31, apresenta as informações sobre custos e se faz sentido iniciar um trabalho de melhoria do projeto dado o Retorno do Investimento (embora a avaliação tenha como escopo somente a manutenibilidade).

Tabela 31 – Quadro comparativo Onda-1 x Onda-2

Projetos/Itens	Onda-1	Onda-2
Ranking manutenibilidade	3-Estrelas	3-Estrelas
Valor de reconstrução	129	107
Dias/pessoa	2701	2242
Débito técnico (Proximo nivel)	45	37
Débito técnico (Nivel ideal)	96	80
RoI/anos	6% - 7 anos	2% - 7 anos

5.3.7 Avaliação dos resultados

Em reunião com o contratante, formalizado através do e-mail da Figura 44, a consultoria apresentou os resultados e indicou as diferenças apresentadas no levantamento, informando a inviabilidade em se manter o valor de manutenção na Onda - 1 praticados pelo fornecedor anterior, dado que os percentuais em termos de qualidade da construção se comparada a Onda - 2 se mostravam significativos.

Deste modo, após os esclarecimentos junto ao o Cliente, ficou definido que o custo de evolução da Onda - 1 seria precificado conforme as estórias em desenvolvimento na Onda - 2.

Figura 44 – Onda-1 x Onda-2 - Alinhamento reunião

From: Fonseca, Anderson
Sent: sexta-feira, 27 de abril de 2018 09:31
To.
Cc:

Subject: RE: Documentação - Montagem Ambiente - NPC - Onda 1

Bom dia . tudo bem?

Fiz uma verificação do código que você enviou (Onda 1), contra o código que produzimos na onda 2. Queria alinhar contigo a apresentação dos pontos, se possível ainda hoje pela manhã, você teria disponiblidade?

Aguardo seu retorno.

Abs,

Anderson Fonseca
Tech Architecture Delivery Manager

Recife, Brazil

6 TRABALHOS RELACIONADOS

Os trabalhos apresentados neste Capítulo, se complementam com os estudos apresentados nos Capítulos 2 e 3. Como esta Tese propôe um método com um abrangência em mais de uma área, ou seja, não limita somente a mineração em repositório, ou, verificação de métricas, foi visto que seria importante não verificar o objetivo desta Tese como um tod, mas também, verificar as partes e quais os trabalhos que encaixariam neste contexto. Deste modo, tais comparações foram segmentada conforme as fases do método proposto.

- **Coletar**
  - O trabalho proposto por (KAGDI; MALETIC, 2006a), apresenta uma infraestrutura baseada no conceito de mineração de repositório de sofware para a predição de mudanças no código-fonte. **Neste trabalho também foi utilizada a mineração em repositório de código como abordagem. Contudo, sendo a atividade de mineração em repositórios de código-fonte considerada como uma abordagem usual para trabalho de arqueologia de software, o diferencial desta tese esteve em caracterizar e direcionar ações, facilitando a melhoria no entendimento do código avaliado no âmbito da industria de TI**.
  - No trabalho proposto por (BANSIYA; DAVIS, 2002) a ferramenta avaliou projetos na linguagem C, porém não a encontramos disponível para avaliação. Considerando que a linguagem Java é a mais utilizada mundialmente segundo a TIOBE[1], optamos pela construção em tal linguagem, para inicialmente avaliarmos projetos nesta tecnologia; **Esta tese utilizou como base para a análise de evolução de qualidade do design o modelo proposto por (BANSIYA; DAVIS, 2002), porém, existiu a subsituição de algumas métricas conforme a popularidade descrita no estado da arte. Neste sentido, comparamos esta parte do método como um melhoria na prescrição da atividade de caracterização proposta**.
  - O modelo DQM proposto por (PLÖSCH et al., 2016), direciona para uma abordagem baseada em mapeamentos de qualidade do design, verificando aspectos de design, desconsiderando métricas primitivas, neste trabalho não são considerados direcionamentos e priorizações para a melhoria da qualidade de código no projeto. **Deste modo, esta tese se diferencia por direcionar o uso de métricas primitivas, compostas e derivadas, conforme apresentado durante o desenvolvimento da proposta**.

[1] https://www.tiobe.com/tiobe-index/ - acessado em 08/2018.

- **Avaliar**

  - A abordagem proposta por (NUGROHO; VISSER; KUIPERS, 2011) descreve uma ferramenta utilizada em atividades de consultoria não disponivel ao público, sem esclarecer como se deu o cálculo do Fator de Produtividade. **Nesta tese foi apresentada uma ferramenta e a definição do Fator de Produtividade baseada em uma tabela de conversão proposta por (LANZA, 2008) permitindo uma replicação do estudo com maior clareza ao leitor**.
  - A ferramenta SonarQube utiliza o SCALE proposto por (LETOUZEY, 2012), o que direciona a definição do débito técnico computando o tempo de correção para cada defeito, ou seja, o custo de manutenção para a avaliação do débito técnico se torna razoável dada o surgimento de novos defeitos. **O método proposto por esta tese utilizou uma técnica de backfiring e priorização, simplificando o esforço de avaliação de débito técnico por defeito**.
  - No trabalho proposto por (IZURIETA et al., 2012), para a análise de débito técnico, existe a necessidade de intervenção de analistas e especialistas em determinados pontos da avaliação, tornando-o um processo semi-automatizado. **Em comparação a esta abordagem, o método proposto sugere um modo completamente automatizado por meio das técnicas de coleta, avaliação e direcionamento apresentados**.

- **Direcionar**

  - A proposta apresentada por (VIDAL; MARCOS; DÍAZ-PACE, 2016) utiliza na composição da fórmula de priorização na correção de code smells o uso de *Beta analysis*, que consiste na avaliação da classe com relação ao projeto como um todo. **Neste sentido, decidimos optar pela utilização da Volatilidade Histórica por se tratar de uma análise aplicada em uma granularidade em nível de classes, independente de variações do projeto**;
  - Em (MUSCO et al., 2016), é proposto um algoritmo de aprendizagem que avalia o impacto em mudanças verificando as falhas dos testes automatizados, para que a avaliação se tornasse possível foram testadas somente aplicações que foram fortemente testadas. **Neste sentido, esta tese verifica projetos na industria de TI que possuem somente o código-fonte como artefato, deste modo, foi proposta um modelo baseado em uma análise de regressão observando o histórico de alterações em conjunto de projetos na indústria, não existindo a necessidade de classes de testes para que houvesse a indicação de possíveis mudanças**.

- **Implementar**

  - A ferramenta proposta por (CHAIKALIS et al., 2014), apresenta uma abordagem similar a RADAR, porém, **não considera questões como Débito Técnico e Projeção de Manuteniblidade**;
  - (THIRUVATHUKAL et al., 2018) propôe uma ferramenta online para a checagem de métricas estruturais e de qualidade de software, porém, não existe uma abordagem além da visualização do conteúdo por meio de um dashboard. **Na RADAR são apresentadas outras funcionalidades em termos de melhoria no entendimento do projeto**

- **Geral**

  - Analisando a tese elaborada por (GHEZZI; LANZA; D'AMBROS, 2004), verificamos que esta apresentou como contribuição um banco de dados no qual combinou o histórico de versões e relatório de defeitos; uma discussão sobre visões polimétricas com foco na evolução de sistemas de software; e o desenvolvimento de uma metodologia *top-down* que conduzia a uma análise de um sistema de software, e por fim, houve uma validação com base em um estudo de caso utilizando o Mozilla. **Nesta tese, apresentou-se como direcionadores os objetivos definidos na Seção 1.3.1, tendo como meio a definição de um método para a busca e análise histórica e de evolução em repositório de código-fonte; a caracterização e a análise de limites com base em definições do estado da arte e da análise do próprio código-fonte; a priorização de correções e o direcionamento de times em atividade melhorias, apresentou uma visão financeira com simulação de custos e RoI em termos de manutenção do código-fonte, e por fim, apresentou a aplicação de proposta em um projeto de código aberto e em três projetos reais desenvolvidos em uma consultoria de TI onde foram analisados como se dava a prática de desenvolvimento de software em um período de 2 anos. Deste modo esta tese teve como diferencial**
  - A Tese proposta por (PETE, 2017) trabalha o gerenciamento de consistência entre artefatos do projeto, neste contexto, foi observada a questão da rastreabilidade, detecção e propagação de mudanças e checagem de consistência entre os artefatos. Algo interessante nesta Tese foi a questão da análise de impacto onde foi verificado por meio da rastreabilidade quais os objetos seriam afetados por tais modificações.

    Nesta Tese, seguimos a mesma proposta em não direcionar ou declarar obrigatóriedade no uso de ferramentas, permitindo que o método possa ser implementado conforme a realidade do projeto, porém, a nossa visão de impacto se relaciona ao custo de manutenção e como

por meio de métricas de software é possível direcionar tal entendimento.

- O trabalho publicado por (LI et al., 2013), apresenta um estudo sobre 30 artigos relacionados a analise de impacto em modificações de código-fonte,sendo verificados que 12 dentre os trabalhos se utilizam de análise histórica, algo que corrobora com a dinâmica prescrita em analise do historico de revisões para a análise de priorização na correção de não-conformidades.
- O TAC++ proposto por (FIORAVANTI; NESI, 2000), apresenta um método e uma ferramenta para a avaliação de projetos orientados a objetos e gerenciamento de métricas, ou seja, uma proposta similiar ao propósito desta Tese. **Neste sentido, nosso trabalho tem como adicional visões de avaliação de termos de projeção financeira e direcionamento de ações para correções e pessoas dentro dos times**

Por fim, o que pode ser concluído na a análise dos trabalhos neste e nos Capítulos 2 e 3, foi o grande valor como contribuição para a elaboração desta tese. Em alguns momentos foi necessária a subsituição ou a complementação de entendimentos pelo fato de haver falta de clareza em algumas abordagens propostas pelos autores dos trabalhos. Porém, o objetivo maior em direcionar algo de forma prática e que trouxesse um considerável valor agregado, foi possível como o uso da teoria associada à automatização do processo de avaliação de qualidade em si.

7 CONSIDERAÇÕES FINAIS

Este trabalho identificou dentro do dia a dia de uma consultoria de TI com atuação global, a necessidade de se avaliar a qualidade do projetos, considerando 3 contextos: novos projetos desenvolvidos a partir do zero; projetos advindos de outras consultorias, e, projetos legados já em produção com o propósito de atuar em melhorias e correções.

Neste sentido, foi verificado junto ao corpo executivo da empresa, a ausência de um método adequado para a predição de impactos em atividades de codificação considerando os 3 contextos descritos. Mesmo havendo estimadores e algumas abordagens tidas como *ferramentas inteligentes*, sendo estas uma forma de embasar a justificativa de prazo e esforço, normalmente, quando estes valores eram repassados aos clientes, os mesmos eram rejeitados. A justificativa para tal rejeição era de que os prazos eram altos e que estes não cabiam na abordagem proposta para o projeto em termos de construção ou manutenção.

Deste modo, primeiramente esta tese se iniciou cobrindo o segundo e terceiro contextos, onde os clientes dessa consultoria não desejavam somente uma estimativa de atuação, mas, um panorama geral dos projetos, sendo apresentados os pontos onde deveriam haver melhorias, quais os custos e os porquês do projeto está na situação atual, podendo este ser um estado bom ou indesejável.

Para atender às necessidades demandadas pelos clientes, o que havia sido solicitado como atuação neste ponto seria a prospecção de ferramentas que atendessem ao tema. Consequentemente, as ferramentas encontradas não cobriam totalmente ou parcialmente os pontos solicitados, porém, mais adiante foi visto que abordagem somente baseada na prospecção de ferramentas sem um entendimento científico, não suportaria a contento à necessidade das avaliações pedidas.

Sendo assim, o método proposto foi elaborado tendo como base toda uma análise histórica sobre qualidade de software fundamentada em propostas e teorias desenvolvidas nas décadas de 70, 80 e 90, com relação a busca em repositórios de código e versionamento, métricas de software, modelos para avaliação de qualidade de software, monitoramento e análise de débito técnico. Dentro desta tese, não houveram proposições de novas métricas, nem a criação de novos modelos de avaliação de qualidade, mas, a definição de um método que permite coletar, avaliar e direcionar um modo de avaliação contínua oferecendo visões que suportem o entendimento em níveis estratégicos.

Diante deste contexto, trazemos os objetivos específicos descritos na Seção 1.3.1, bem como, a aderência desta tese a tais pontos:

- Dado que sistemas de software carecem de artefatos consistentes, e tomando por base somente o código-fonte dos projetos, investigar e entender o porquê das métricas de software coletadas não apresentarem um modelo onde, efetivamente, os resultados

obtidos fornecem a visibilidade necessária para suportar decisões estratégicas;

Neste sentido, onde os entendimentos foram obtidos e descritos nos Capítulos 2 e 3, no qual os modelos identificados tratavam este objetivo de forma parcial ou seus formatos eram proprietários não permitindo uma abordagem direta com uso de métricas, mapeamentos entre níveis de qualidade e visão financeira.

- Considerando as métricas de software obtidas e apresentadas em um nível executivo, entender e definir como os resultados obtidos poderiam ser apresentados de forma a subsidiar seguramente uma visão sobre a real situação do projeto, em forma de indicadores;

  Seguramente o uso de indicadores e *dashboards* por si, se tornam facilitadores do entendimento, porém, durante as atuações no dia a dia dos projetos, o grande desafio esteve em responder perguntas sobre: Como está a qualidade...?, O que significam tais resultados...? Por onde devemos começar as melhorias...? Enfim, o que se buscou neste objetivo foi transmitir de forma clara, como os resultados obtidos a partir do código-fonte poderiam ser tornar de fácil entendimento para uma audiência em nível estratégico sem a necessidade de um tradutor, um suporte técnico.

- Entendendo os indicadores recebidos quanto à saúde de um projeto, direcionar um modelo de avaliação e predição de impactos nos custos de forma a subsidiar a tomada de decisão.

  Este objetivo se tornou importante devido à questão financeira, pois, é bastante comum que em tomadas de decisão se considere o fator custo e quais as ações pertinentes para o controle do orçamento. Deste modo, esta tese apresentou uma visão de projeção de custos por um período de 10 anos, baseando-se em fatores descritos na Seção 4.3.2.

- Estabelecer uma abordagem para o monitoramento dos projetos de forma contínua, sendo possível direcionar ações com base em indicadores, projeções de custos, classificação e melhoria contínua em partes do sistema.

  Para este item, foi elaborado um índice de priorização conforme a Seção 4.4.1 onde o que se procurou atender foi como se iniciar um trabalho corretivo com resultados vísiveis a curto prazo, dado que, as atividades de construção continuam existindo em paralelo. Neste sentido, é comum que trabalhos de refatoramento não se reflitam de forma imediata caso as classes envolvidas não estejam relacionadas às demandas correntes, o que de certo modo, gera dúvidas se a tomada de decisão de correção e melhorias de fato foi efetiva.

Conforme já discutido, a formatação deste método permitiu o ganho de confiança e a demonstração de expertise da consultoria aos clientes (conforme os Casos práticos apresentados no Capítulo 5) no tema referente a suporte a decisão, dado que estes perceberam a evolução e o estado dos projetos avaliados sob diferentes perspectivas.

Enfim, como conclusão foi possível assumir que técnicas, métricas, métodos e teorias existentes, eram suficientes para subsidiar uma avaliação sobre a qualidade do software, porém, como encaixar estas peças de forma que apresentasse algum valor agregado, apresentando os resultados obtidos em formato mais leve e inteligível, considerando uma audiência não técnica, tornou-se o grande desafio deste trabalho.

7.1 CONTRIBUIÇÃO

Este trabalho fez as seguintes contribuições até o presente momento:

- Um método sistêmico de coleta de métricas em repositórios de código e versionamento, fornecendo indícios sobre o estado atual do projeto;
- Investigar, definir e especificar um método que torna o tema avaliação de qualidade de código uma peça importante na tomada de decisão, dado que o que está sendo proposto não se resume somente a exposição de quantidade de defeitos;
- Investigar, definir e especificar uma forma de avaliação da qualidade do projeto, observando mais de um fator como direcionador e a possibilidade de prever custos e retorno do investimento;
- Investigar, definir e especificar um modelo de direcionamento onde são considerados o grau de contribuição e expertise de membros do time, bem como, a probabilidade de manutenções em determinados pontos do projeto.

7.2 TRABALHOS FUTUROS

Como a abordagem proposta necessita de uma ferramenta robusta a ponto de subsidar outros projetos, seguem alguns pontos como trabalhos futuros:

- Aplicar a proposta para outras linguagens de desenvolvimento;
- Evoluir a parte de gestão de projetos para a integração com ferramentas de mercado;
- Criação de novos conectores para a busca em repositórios como Git, CVS e StarTeam;
- Replicar os estudos em projetos de código aberto.

- Avaliar a viabilidade do uso da proposta em outras verticais, dado que esta avaliação se resumiu a projetos de telecomunicações, midia e financeiro;
- Publicação de artigos em conferências e periodicos.

REFERÊNCIAS

AL-BADAREEN, A. B.; SELAMAT, M. H.; JABAR, M. A.; DIN, J.; TURAEV, S. Software quality models: A comparative study. In: SPRINGER. *International Conference on Software Engineering and Computer Systems*. [S.l.], 2011. p. 46–55.

AL-QUTAISH, R. E. Quality models in software engineering literature: an analytical and comparative study. *Journal of American Science*, v. 6, n. 3, p. 166–175, 2010.

AL-QUTAISH, R. E. Quality Models in Software Engineering Literature An Analytical and Comparative Study. v. 6, n. 3, p. 166–175, 2010.

ALVARO, A.; ALMEIDA, E.; MEIRA, S. Towards a software component quality model. 2005.

ALVES, T. L.; YPMA, C.; VISSER, J. Deriving metric thresholds from benchmark data. In: IEEE. *Software Maintenance (ICSM), 2010 IEEE International Conference on*. [S.l.], 2010. p. 1–10.

AVELINO, G.; VALENTE, M. T.; HORA, A. What is the Truck Factor of popular GitHub applications? A first assessment. *PeerJ PrePrints*, v. 3, p. e1683, 2015. ISSN 2167-9843. Disponível em: <https://dx.doi.org/10.7287/peerj.preprints.1233v2>.

BANSIYA, J.; DAVIS, C. G. A hierarchical model for object-oriented design quality assessment. *IEEE Transactions on Software Engineering*, v. 28, n. 1, p. 4–17, 2002. ISSN 00985589.

BARRY, E. J.; KEMERER, C. F.; SLAUGHTER, S. A. Toward a detailed classification scheme for software maintenance activities. *AMCIS 1999 PROCEEDINGS*, v. 251, 1999.

BARTON, B.; STERLING, C. Manage project portfolios more effectively by including software debt in the decision process. *Cutter IT Journal*, v. 23, n. 10, p. 19, 2010.

BASILI, V. R.; ROMBACH, H. D. The TAME project towards improvement oriented software environments. *Software Engineering, IEEE Transactions on*, v. 14, n. 6, p. 758–773, 1988.

BASILI, V. R.; WEISS, D. M. A methodology for collecting valid software engineering data. *IEEE Trans. Software Eng.*, v. 10, n. 6, p. 728–738, 1984.

BENLARBI, S.; EMAM, K. E.; GOEL, N.; RAI, S. Thresholds for object-oriented measures. In: IEEE. *Software Reliability Engineering, 2000. ISSRE 2000. Proceedings. 11th International Symposium on*. [S.l.], 2000. p. 24–38.

BERTOA, M. F.; VALLECILLO, A. Quality attributes for cots components. 2002.

BOEHM, B. W. *Characteristics of Software Quality*. 1st. ed. [S.l.]: North-Holland Publishing Company, 1978. v. 1. ISBN 0444851054.

BUSE; PL., W. R.; R., W. Learning a metric for code readability. *IEEE Transactions on Software Engineering*, v. 36, n. 4, p. 546–558, 2010. ISSN 00985589.

BUZLUCA, F.; ESKI, S. An empirical study on object-oriented metrics and software evolution in order to reduce testing costs by predicting change-prone classes. In: *Software Testing Verification and Validation Workshop, IEEE International Conference on(ICSTW)*. [s.n.], 2011. v. 00, p. 566–571. Disponível em: <doi.ieeecomputersociety.org/10.1109/ICSTW.2011.43>.

CARRELL, M. R.; JENNINGS, D. F.; HEAVRIN, C. *Fundamentals of organizational behavior*. [S.l.]: Prentice Hall, 1997.

CHAIKALIS, T.; LIGU, E.; MELAS, G.; CHATZIGEORGIOU, A. SEAgle Effortless Software Evolution Analysis. p. 582–585, 2014.

CHATZIGEORGIOU, A.; MANAKOS, A. Investigating the evolution of code smells in object-oriented systems. *Innovations in Systems and Software Engineering*, v. 10, n. 1, p. 3–18, 2014. ISSN 16145046.

CHIDAMBER, S. R.; KEMERER, C. F. A metrics suite for object oriented design. *IEEE Trans. Softw. Eng.*, IEEE Press, Piscataway, NJ, USA, v. 20, n. 6, p. 476–493, jun. 1994. ISSN 0098-5589. Disponível em: <http://dx.doi.org/10.1109/32.295895>.

CUNNINGHAM, W. The wycash portfolio management system. *OOPS Messenger*, v. 4, n. 2, p. 29–30, 1993. Disponível em: <http://dblp.uni-trier.de/db/journals/oopsm/oopsm4.html#Cunningham93>.

DASKALANTONAKIS, M. A practical view of software measurement and implementation experiences within motorola. *IEEE Transactions on Software Engineering*, IEEE Computer Society, Los Alamitos, CA, USA, v. 18, p. 998–1010, 1992. ISSN 0098-5589.

DINARI, F. Halstead Complexity Metrics in Software Engineering. *Journal of Renewable Natural Resources Bhutan*, 2015.

DROMEY, R. G. A model for software product quality. *IEEE Trans. Software Eng.*, v. 21, n. 2, p. 146–162, 1995. Disponível em: <http://dblp.uni-trier.de/db/journals/tse/tse21.html#Dromey95>.

FENTON, N. E.; NEIL, M. Software metrics: successes, failures and new directions. *Journal of Systems and Software*, v. 47, n. 2-3, p. 149–157, 1999.

FENTON, N. E.; NEIL, M. Software metrics: roadmap. In: ACM. *Proceedings of the Conference on the Future of Software Engineering*. [S.l.], 2000. p. 357–370.

FERREIRA, K. A.; BIGONHA, M. A.; BIGONHA, R. S.; MENDES, L. F.; ALMEIDA, H. C. Identifying thresholds for object-oriented software metrics. *Journal of Systems and Software*, Elsevier, v. 85, n. 2, p. 244–257, 2012.

FILÓ, T. G. S.; BIGONHA, M. A. da S. A catalogue of thresholds for object-oriented software metrics. In: . [S.l.: s.n.], 2015.

FIORAVANTI, F.; NESI, P. A method and tool for assessing object-oriented projects and metrics management. *Journal of Systems and Software*, Elsevier, v. 53, n. 2, p. 111–136, 2000.

FOWLER, M.; K., B.; W., B. J. O.; D., R. Refactoring: Improving the design of existing code. In: XTEMP01. *Refactoring: Improving the Design of Existing Code*. [S.l.], 1999. p. 1–337.

FRITZ, T.; MURPHY, G. C.; MURPHY-HILL, E.; OU, J.; HILL, E. Degree-of-knowledge: Modeling a developer's knowledge of code. *ACM Trans. Softw. Eng. Methodol.*, ACM, New York, NY, USA, v. 23, n. 2, p. 14:1–14:42, abr. 2014. ISSN 1049-331X. Disponível em: <http://doi.acm.org/10.1145/2512207>.

FRITZ, T.; OU, J.; MURPHY, G. C.; MURPHY-HILL, E. A degree-of-knowledge model to capture source code familiarity. In: *Proceedings of the 32Nd ACM/IEEE International Conference on Software Engineering - Volume 1*. New York, NY, USA: ACM, 2010. (ICSE '10), p. 385–394. ISBN 978-1-60558-719-6. Disponível em: <http://doi.acm.org/10.1145/1806799.1806856>.

GAMMA, E.; HELM, R.; JOHNSON, R.; VLISSIDES, J. Design patterns: Elements of reusable object-oriented software. In: ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES. *Design Patterns: Elements of reusable object-oriented software*. [S.l.], 1996. p. 395.

GANEA, G.; VEREBI, I.; MARINESCU, R. Continuous quality assessment with incode. *Science of Computer Programming*, Elsevier, v. 134, p. 19–36, 2017.

GEORGIADOU, E. Gequamo—a generic, multilayered, customisable, software quality model. *Software Quality Journal*, v. 11, n. 4, p. 313–323, Nov 2003. ISSN 1573-1367. Disponível em: <https://doi.org/10.1023/A:1025817312035>.

GHEZZI, C.; LANZA, M.; D'AMBROS, M. *Software Archaeology-Reconstructing the Evolution of Software Systems*. Tese (Doutorado), 2004.

GÎRBA, T.; DUCASSE, S.; LANZA, M. Yesterday's weather - guiding early reverse engineering efforts by summarizing the evolution of changes. *20th IEEE International Conference on Software Maintenance, 2004. Proceedings.*, p. 40–49, 2004. ISSN 1063-6773. Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1357788>.

GLASS, R. L. Frequently forgotten fundamental facts about software engineering. In: SE. *IEEE Software*. [S.l.], 2001.

GLOTT, R.; GROVEN, A.; HAALAND, K.; TANNENBERG, A. Quality models for free/libre open source software towards the "silver bullet"? In: *2010 36th EUROMICRO Conference on Software Engineering and Advanced Applications*. [S.l.: s.n.], 2010. p. 439–446. ISSN 2376-9505.

GÖG, C.; WEISSGERBER, P. Detecting and visualizing refactorings from software archives. *Proceedings - IEEE Workshop on Program Comprehension*, p. 205–214, 2005. ISSN 10928138.

GORSCHEK, T.; GARRE, P.; LARSSON, S.; WOHLIN, C. A model for technology transfer in practice. *IEEE Softw.*, IEEE Computer Society Press, Los Alamitos, CA, USA, v. 23, n. 6, p. 88–95, nov. 2006. ISSN 0740-7459. Disponível em: <http://dx.doi.org/10.1109/MS.2006.147>.

GOUSIOS, G.; SPINELLIS, D. Alitheia Core: An extensible software quality monitoring platform. In: *ICSE '09: Proceedings of the 31st International Conference on Software Engineering — Formal Research Demonstrations Track*. IEEE, 2009. p. 579–582. ISBN 978-1-4244-3743-6. Disponível em: <http://www.dmst.aueb.gr/dds/pubs/conf/2009-ICSE-SQO-OSS-resdemo/html/GS09.html>.

GRADY, R. B. *Practical Software Metrics for Project Management and Process Improvement*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1992. ISBN 0-13-720384-5.

HALSTEAD, M. H. et al. *Elements of Software Science (Operating and programming systems series)*. [S.l.]: Elsevier Science Inc., New York, NY, 1977.

HASSAN, A. E. The road ahead for mining software repositories. In: *2008 Frontiers of Software Maintenance*. [S.l.: s.n.], 2008. p. 48–57.

HEITLAGER, I.; KUIPERS, T.; VISSER, J. A Practical Model for Measuring Maintainability. *6th International Conference on the Quality of Information and Communications Technology (QUATIC 2007)*, p. 30–39, 2007. ISSN 9780769529486. Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4335232>.

HERBOLD, S.; GRABOWSKI, J.; WAACK, S. Calculation and optimization of thresholds for sets of software metrics. *Empirical Software Engineering*, Springer, v. 16, n. 6, p. 812–841, 2011.

HYATT, L.; ROSENBERG, L. A software quality model and metrics for risk assessment. 1996.

ISO. *International Standard ISO/IEC 12207 Software Life Cycle Processes*. [S.l.], 1995.

ISO/IEC 25010. Iso/Iec 25010-2011. *Software Process: Improvement and Practice*, v. 2, n. Resolution 937, p. 1–25, 2011. ISSN 10774866. Disponível em: <http://www.iso.org/iso/iso{_}catalogue/catalogue{_}tc/catalogue{_}detail.htm?csnum>.

ISO/IEC 9126. *ISO/IEC 9126-1:2001, Software engineering – Product quality – Part 1: Quality model*. [S.l.], 2001. Disponível em: <http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=22749>.

IZURIETA, C.; VETRÒ, A.; ZAZWORKA, N.; CAI, Y.; SEAMAN, C.; SHULL, F. Organizing the technical debt landscape. *2012 3rd International Workshop on Managing Technical Debt, MTD 2012 - Proceedings*, p. 23–26, 2012.

JONES, C. Backfiring: Converting lines of code to function points. *Computer*, IEEE, v. 28, n. 11, p. 87–88, 1995.

JONES, C. Programming languages table. *Software Productivity Research Inc*, 1996.

JONES, G. R. *Organizational Theory, Design, And Change /*. 6th/ global ed. ed. Upper Saddle River, N.J.: Prentice Hall, 2010.

KAGDI, H.; MALETIC, J. I. Software-change prediction: Estimated+ actual. In: IEEE. *Software Evolvability, 2006. SE'06. Second International IEEE Workshop on*. [S.l.], 2006. p. 38–43.

KAGDI, H.; MALETIC, J. I. Software-change prediction: Estimated+actual. 09 2006.

KAN, S. H. *Metrics and Models in Software Quality Engineering - Paperback*. 2nd. ed. [S.l.]: Addison-Wesley Professional, 2014. ISBN 0133988082, 9780133988086.

KEENEY R. L., H. R. Decisions with multiple objectives. *John Wiley and Sons, New York.*, 1976.

KHOMH, F.; GUÉHÉNEUC, Y.-G. Dequalite: building design-based software quality models. In: ACM. *Proceedings of the 15th Conference on Pattern Languages of Programs*. [S.l.], 2008. p. 2.

KHURUM, M.; GORSCHEK, T.; WILSON, M. The software value map - an exhaustive collection of value aspects for the development of software intensive products. *Journal of Software: Evolution and Process*, v. 25, p. 711–741, 2013.

KHURUM, M.; GORSCHEK, T.; WILSON, M. The software value map—an exhaustive collection of value aspects for the development of software intensive products. *Journal of Software: Evolution and Process*, Wiley Online Library, v. 25, n. 7, p. 711–741, 2013.

KOC, A.; TANSEL, A. U.; BICER, M. Towards social version control. In: *Proceedings of the 2012 International Conference on Advances in Social Networks Analysis and Mining (ASONAM 2012)*. Washington, DC, USA: IEEE Computer Society, 2012. (ASONAM '12), p. 722–723. ISBN 978-0-7695-4799-2. Disponível em: <http://dx.doi.org/10.1109/ASONAM.2012.129>.

KOMI-SIRVIÖ, S.; PARVIAINEN, P.; RONKAINEN, J. Measurement automation: Methodological background and practical solutions-a multiple case study. In: *7th IEEE International Software Metrics Symposium (METRICS 2001), 4-6 April 2001, London, England*. [S.l.: s.n.], 2001. p. 306–316.

KOTHAPALLI, C.; GANESH, S. G.; SINGH, H. K.; RADHIKA, D. V.; RAJARAM, T.; RAVIKANTH, K.; GUPTA, S.; RAO, K. Continual monitoring of code quality. In: *Proceedings of the 4th India Software Engineering Conference*. New York, NY, USA: ACM, 2011. (ISEC '11), p. 175–184. ISBN 978-1-4503-0559-4. Disponível em: <http://doi.acm.org/10.1145/1953355.1953379>.

LANZA, G. Function point: how to transform them in effort? this is the problem! *Proceedings 5th Software Measurement European Forum*, 2008.

LANZA, M.; MARINESCU, R. *Object-oriented metrics in practice: using software metrics to characterize, evaluate, and improve the design of object- oriented systems*. [S.l.]: Springer Science & Business Media, 2007.

LETOUZEY, J.-L. *The SQALE Method Definition Document*. [s.n.], 2012. ISBN 0321166078. Disponível em: <http://www.sqale.org/wp-content/uploads/2010/08/SQALE-Method-EN-V1-0.pdf>.

LI, B.; SUN, X.; LEUNG, H.; ZHANG, S. A survey of code-based change impact analysis techniques. *Software Testing, Verification and Reliability*, Wiley Online Library, v. 23, n. 8, p. 613–646, 2013.

LI, W.; HENRY, S. Object-oriented metrics that predict maintainability. *J. Syst. Softw.*, Elsevier Science Inc., New York, NY, USA, v. 23, n. 2, p. 111–122, nov. 1993. ISSN 0164-1212.

LI, Z.; AVGERIOU, P.; LIANG, P. A systematic mapping study on technical debt and its management. *Journal of Systems and Software*, Elsevier Ltd., v. 101, n. October 2017, p. 193–220, 2015. ISSN 01641212. Disponível em: <http://dx.doi.org/10.1016/j.jss.2014.12.027>.

LI, Z.; GUELFI, N.; LIANG, P.; AVGERIOU, P.; AMPATZOGLOU, a. An empirical investigation of modularity metrics for indicating architectural technical debt. *QoSA 2014 - Proceedings of the 10th International ACM SIGSOFT Conference on Quality of Software Architectures (Part of CompArch 2014)*, p. 119–128, 2014. Disponível em: <http://www.scopus.com/inward/record.url?eid=2-s2.0-84904469347{&}partnerID=40{&}md5=406afb73e9bffa5b3a0acdeee0>.

LORENZ, M.; KIDD, J. *Object-oriented Software Metrics: A Practical Guide*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1994. ISBN 0-13-179292-X.

LUCIA, A. D.; FASOLINO, A. R.; POMPELLE, E. A decisional framework for legacy system management. In: *Proceedings IEEE International Conference on Software Maintenance. ICSM 2001.* [S.l.: s.n.], 2001. p. 642–651. ISSN 1063-6773.

MCCABE. A complexity measure. *IEEE Transactions on Software Engineering*, v. 2, p. 308–320, 1976.

MCCALL, J. *Factors in Software Quality: Preliminary Handbook on Software Quality for an Acquisiton Manager*. General Electric, 1977. v. 1-3. Disponível em: <http://oai.dtic.mil/oai/oai?verb=getRecord&metadataPrefix=html&identifier=ADA049055>.

MENEELY, A.; SMITH, B.; WILLIAMS, L. Validating software metrics: A spectrum of philosophies. *ACM Trans. Softw. Eng. Methodol.*, ACM, New York, NY, USA, v. 21, n. 4, p. 24:1–24:28, fev. 2013. ISSN 1049-331X. Disponível em: <http://doi.acm.org/10.1145/2377656.2377661>.

MIGUEL, J. P.; MAURICIO, D.; RODRÍGUEZ, G. A review of software quality models for the evaluation of software products. *arXiv preprint arXiv:1412.2977*, 2014.

MORDAL-MANET, K.; BALMAS, F.; DENIER, S.; DUCASSE, S.; WERTZ, H.; VAILLERGUES, P. The Squale Model – A Practice -Based Industrial Quality Model. p. 531–534, 2009.

MURPHY-HILL, E.; PARNIN, C.; BLACK, A. P. How we refactor, and how we know it. *IEEE Transactions on Software Engineering*, v. 38, n. 1, p. 5–18, 2012. ISSN 00985589.

MUSCO, V.; CARETTE, A.; MONPERRUS, M.; PREUX, P. A learning algorithm for change impact prediction: Experimentation on 7 java applications. *arXiv preprint arXiv:1512.07435*, 2015.

MUSCO, V.; CARETTE, A.; MONPERRUS, M.; PREUX, P. A learning algorithm for change impact prediction. In: IEEE. *Realizing Artificial Intelligence Synergies in Software Engineering (RAISE), 2016 IEEE/ACM 5th International Workshop on.* [S.l.], 2016. p. 8–14.

NUGROHO, A.; VISSER, J.; KUIPERS, T. An empirical model of technical debt and interest. *MTD '11: Proceedings of the 2nd Workshop on Managing Technical Debt*, p. 1–8, 2011. ISSN 02705257. Disponível em: <http://portal.acm.org/citation.cfm?id=1985362.1985364{\%}7B{\%}5C{&}{\%}7Dcoll=DL{\%}7B{\%}5C{&}{\%}7Ddl=ACM{\%}7B{\%}5C{&}{\%}7DCFID=358287041{\%}7B{\%}5C{&}{\%}7DCFTOKEN=64144089{\%}5CnC:{\%}5CUsers{\%}5Ctsoutom{\%}5CDesktop{\%}5CSystematicMappingTD>.

NUÑEZ-VARELA, A. S.; PÉREZ-GONZALEZ, H. G.; MARTÍNEZ-PEREZ, F. E.; SOUBERVIELLE-MONTALVO, C. Source code metrics: A systematic mapping study. *Journal of Systems and Software*, Elsevier, v. 128, p. 164–197, 2017.

OLIVEIRA, P.; LIMA, F.; VALENTE, M. T.; SEREBRENIK, A. RTTOOL: A tool for extracting relative thresholds for source code metrics. In: *30th International Conference on Software Maintenance and Evolution (ICSME), Tool Demo Track*. [S.l.: s.n.], 2014. p. 1–4.

OLIVEIRA, P.; VALENTE, M. T.; BERGEL, A.; SEREBRENIK, A. Validating metric thresholds with developers: An early result. In: *2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. [S.l.: s.n.], 2015. p. 546–550.

OLIVEIRA, P.; VALENTE, M. T.; LIMA, F. Extracting relative thresholds for source code metrics. In: *IEEE Conference on Software Maintenance, Reengineering and Reverse Engineering (CSMR-WCRE)*. [S.l.: s.n.], 2014. p. 254–263.

OMAN, P.; HAGEMEISTER, J. Metrics for assessing a software system's maintainability. In: *Proc. Conf. on Software Maintenance*. [S.l.: s.n.], 1992. p. 337–344.

OUHBI, S.; IDRI, A.; ALEMÁN, J. L. F.; TOVAL, A. Evaluating software product quality: A systematic mapping study. In: *2014 Joint Conference of the International Workshop on Software Measurement and the International Conference on Software Process and Product Measurement, Rotterdam, The Netherlands, October 6-8, 2014*. [S.l.: s.n.], 2014. p. 141–151.

PAPAMICHAIL, M.; DIAMANTOPOULOS, T.; SYMEONIDIS, A. User-perceived source code quality estimation based on static analysis metrics. In: IEEE. *2016 IEEE International Conference on Software Quality, Reliability and Security (QRS)*. [S.l.], 2016. p. 100–107.

PARIZI, R. M.; GHANI, A. A. A. Towards automated monitoring and forecasting of probabilistic quality properties in open source software (oss) a striking hybrid approach. *Software Engineering Research, Management and Applications, ACIS International Conference on*, IEEE Computer Society, Los Alamitos, CA, USA, v. 00, p. 329–334, 2010.

PETE, I. *Towards a holistic framework for software artefact consistency management*. Tese (Doutorado) — University of St Andrews, 2017.

PETERSEN, K.; FELDT, R.; MUJTABA, S.; MATTSSON, M. Systematic mapping studies in software engineering. In: *EASE*. [S.l.: s.n.], 2008. v. 8, p. 68–77.

PIGOSKI, T. M. Chapter 6. software maintenance. *IEEE – Trial Version 1.00 - Technical Software Services (TECHSOFT), Inc.*, 2001.

PLÖSCH, R.; BRÄUER, J.; KÖRNER, C.; SAFT, M. Measuring, Assessing and Improving Software Quality based on Object-Oriented Design Principles. *Open Computer Science*, v. 6, n. 1, p. 187–207, 2016. ISSN 2299-1093. Disponível em: <http://www.degruyter.com/view/j/comp.2016.6.issue-1/comp-2016-0016/comp-2016-0016.xml>.

PLöSCH, R.; MAYR, A.; POMBERGER, G.; SAFT, M. An approach for a method and a tool supporting the evaluation of the quality of static code analysis tools. 07 2009.

QUAH, E.; HALDANE, J. *Cost-benefit analysis*. [S.l.]: Routledge, 2007.

RAWASHDEH, A.; MATALKAH, B. A new software quality model for evaluating cots components. *Journal of Computer Science*, v. 2, n. 4, p. 373–381, 2006.

ROSENBERG, L. Applying and interpreting object oriented metrics. In: *Software Technology Conference, Utah, April 1998*. [S.l.: s.n.], 1998.

SAATY, T. L. Decision making with the analytic hierarchy process. *International journal of services sciences*, Inderscience Publishers, v. 1, n. 1, p. 83–98, 2008.

SAE-LIM, N.; HAYASHI, S.; SAEKI, M. Context-based code smells prioritization for prefactoring. In: IEEE. *Program Comprehension (ICPC), 2016 IEEE 24th International Conference on*. [S.l.], 2016. p. 1–10.

SAMARTHYAM, G.; SURYANARAYANA, G.; SHARMA, T.; GUPTA, S. Midas: a design quality assessment method for industrial software. In: IEEE PRESS. *Proceedings of the 2013 International Conference on Software Engineering*. [S.l.], 2013. p. 911–920.

SANTOS, M. B.; JÚNIOR, P. A. P.; BERMEJO, P. H.; COSTA, H. A. X. Metrics and statistical techniques used to evaluate internal quality of object-oriented software: A systematic mapping. *2016 35th International Conference of the Chilean Computer Science Society (SCCC)*, p. 1–11, 2016.

SARAIVA, J.; FRANçA, M. S. de; SOARES, S.; FILHO, F. C.; SOUZA, R. M. C. R. de. Classifying metrics for assessing object-oriented software maintainability: A family of metrics' catalogs. *Journal of Systems and Software*, v. 103, p. 85–101, 2015.

SEAMAN, C.; GUO, Y.; ZAZWORKA, N.; SHULL, F.; IZURIETA, C.; CAI, Y.; VETRò, A. Using technical debt data in decision making: Potential decision approaches. In: *2012 Third International Workshop on Managing Technical Debt (MTD)*. [S.l.: s.n.], 2012. p. 45–48.

SHATNAWI, R. An investigation of ck metrics thresholds. In: *ISSRE Supplementary Conference Proceedings*. [S.l.: s.n.], 2006. p. 12–13.

SHATNAWI, R. A quantitative investigation of the acceptable risk levels of object-oriented metrics in open-source systems. *IEEE Transactions on software engineering*, IEEE, v. 36, n. 2, p. 216–225, 2010.

SHATNAWI, R.; LI, W. An Empirical Assessment of Refactoring Impact on Software Quality Using a Hierarchical Quality Model. *Journal of Information and Software Technology*, v. 5, n. 4, p. 127–150, 2011.

SHATNAWI, R.; LI, W.; SWAIN, J.; NEWMAN, T. Finding software metrics threshold values using roc curves. *Journal of software maintenance and evolution: Research and practice*, Wiley Online Library, v. 22, n. 1, p. 1–16, 2010.

SOARES, G.; CATÃO, B.; VARJÃO, C.; AGUIAR, S.; GHEYI, R.; MASSONI, T. Analyzing refactorings on software repositories. *Proceedings - 25th Brazilian Symposium on Software Engineering, SBES 2011*, p. 164–173, 2011.

SUMAN, M. W.; ROHTAK, M. A comparative study of software quality models. *International Journal of Computer Science and Information Technologies*, v. 5, n. 4, p. 5634–5638, 2014.

SWANSON, E. B. The Dimensions of Maintenance. *Proceedings of the 2nd international conference on Software engineering*, p. 492–497, 1976. ISSN 1098-6596.

THIRUVATHUKAL, G. K.; HAYWARD, N. J.; LÄUFER, K. et al. Metrics dashboard: A hosted platform for software quality metrics. *arXiv preprint arXiv:1804.02053*, 2018.

TIMÓTEO, A. L.; ÁLVARO, A.; ALMEIDA, E. S. de; MEIRA, S. R. de L. Software metrics: A survey. In: . [S.l.: s.n.], 2008.

TRIANTAPHYLLOU, E.; KOVALERCHUK, B.; MANN, L.; KNAPP, G. M. Determining the most important criteria in maintenance decision making. v. 3, p. 16–28, 03 1997.

TUFANO, M.; PALOMBA, F.; BAVOTA, G.; OLIVETOX, R.; Di Penta, M.; De Lucia, A.; POSHYVANYK, D. When and why your code starts to smell bad. *Proceedings - International Conference on Software Engineering*, v. 1, p. 403–414, 2015. ISSN 02705257.

UMARJI, M.; SEAMAN, C. Why do programmers avoid metrics? In: *Proceedings of the Second ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*. ACM, 2008. (ESEM '08), p. 129–138. ISBN 978-1-59593-971-5. Disponível em: <http://doi.acm.org/10.1145/1414004.1414027>.

VIDAL, S. A.; MARCOS, C.; DÍAZ-PACE, J. A. An approach to prioritize code smells for refactoring. *Automated Software Engineering*, Springer, v. 23, n. 3, p. 501–532, 2016.

WAGNER, S.; LOCHMANN, K.; HEINEMANN, L.; KLÄS, M.; TRENDOWICZ, A.; PLÖSCH, R.; SEIDL, A.; GOEB, A.; STREIT, J. The quamoco product quality modelling and assessment approach. In: IEEE PRESS. *Proceedings of the 34th international conference on software engineering*. [S.l.], 2012. p. 1133–1142.

WEDYAN, F.; ALRMUNY, D.; BIEMAN, J. M. The effectiveness of automated static analysis tools for fault detection and refactoring prediction. *Proceedings - 2nd International Conference on Software Testing, Verification, and Validation, ICST 2009*, n. Icst, p. 141–150, 2009. ISSN 2159-4848.

WETTEL, R. *Software Systems as Cities*. Tese (Doutorado) — Università della Svizzera Italiana, 2010.

WILLIAMS, L.; KESSLER, R. *Pair Programming Illuminated*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2002. ISBN 0201745763.

WOHLIN, C. Guidelines for snowballing in systematic literature studies and a replication in software engineering. In: *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering*. New York, NY, USA: ACM, 2014. (EASE '14), p. 38:1–38:10. ISBN 978-1-4503-2476-2. Disponível em: <http://doi.acm.org/10.1145/2601248.2601268>.

XIAO, L.; CAI, Y.; KAZMAN, R.; MO, R.; FENG, Q. Identifying and quantifying architectural debt. In: ACM. *Proceedings of the 38th International Conference on Software Engineering*. [S.l.], 2016. p. 488–498.

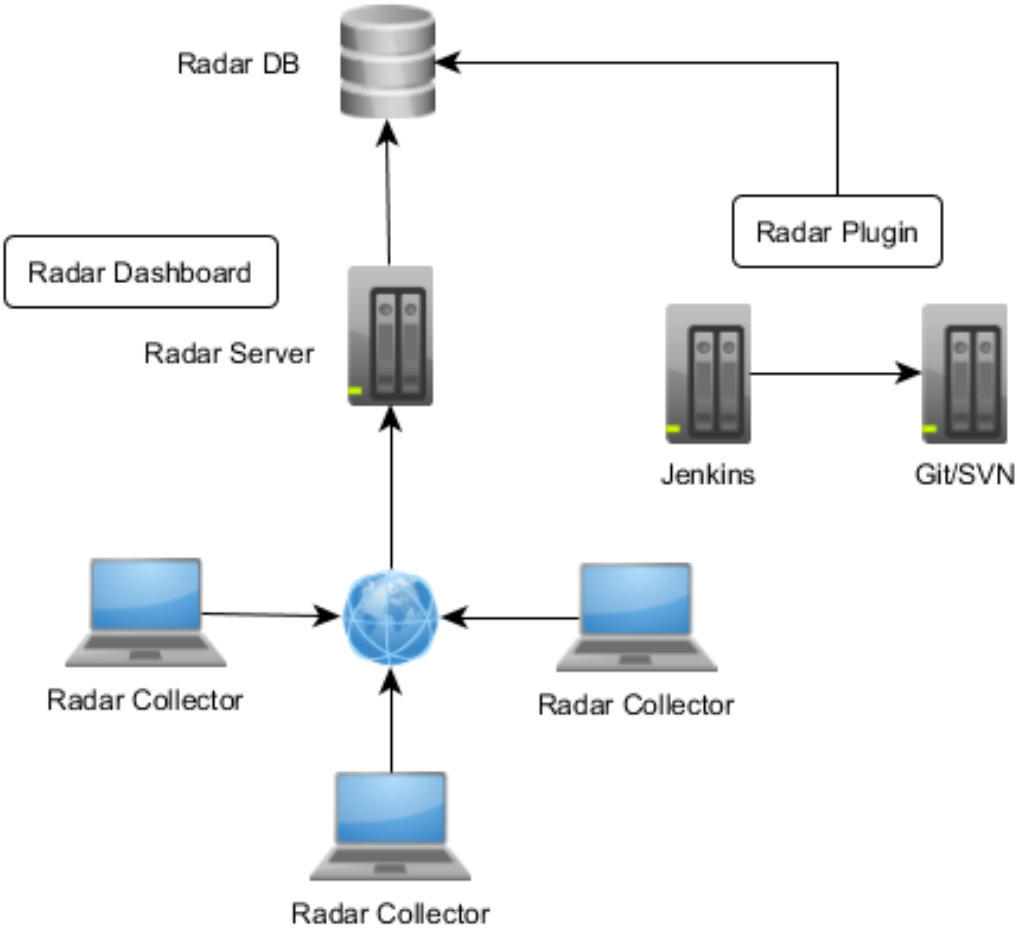
YAMASHITA, K.; HUANG, C.; NAGAPPAN, M.; KAMEI, Y.; MOCKUS, A.; HASSAN, A. E.; UBAYASHI, N. Thresholds for size and complexity metrics: A case study from the perspective of defect density. In: IEEE. *Software Quality, Reliability and Security (QRS), 2016 IEEE International Conference on*. [S.l.], 2016. p. 191–201.

YOUNESS, B.; ABDELAZIZ, M.; HABIB, B.; HICHAM, M. Comparative study of software quality models. In: . [S.l.: s.n.], 2013.

APÊNDICE A – RADAR: FERRAMENTA DE APOIO AO MÉTODO PROPOSTO

A ferramenta RADAR foi criada com o objetivo de automatizar o método proposto neste trabalho, bem como, tentar preencher a ausência de ferramentas observando a afirmação onde segundo (TIMÓTEO et al., 2008), "Métricas automatizadas careciam de mapeamento entre os resultados coletados em código e o seu mapeamento em atributos de qualidade".

Figura 45 – RADAR Visão da solução



A Figura 45, apresenta uma visão da coleta de dados de forma manual via o componente Radar Collector, bem como, de forma automatizada com base na utilização do Radar Plugin, acionado por meio de uma *taskjob* do Jenkins.

Para a coleta de métricas e análise do código-fonte foram desenvolvidos mecanismos utilizando o componente JavaParser[1].

[1] https://github.com/javaparser/javaparser

Para a construção do protótipo, o suporte inicial foi para a busca em repositórios de projetos SVN por meio do componente RADAR-SVN, porém, é possível desenvolver e plugar outro conector para a busca de projetos em outros tipos de repositorios.

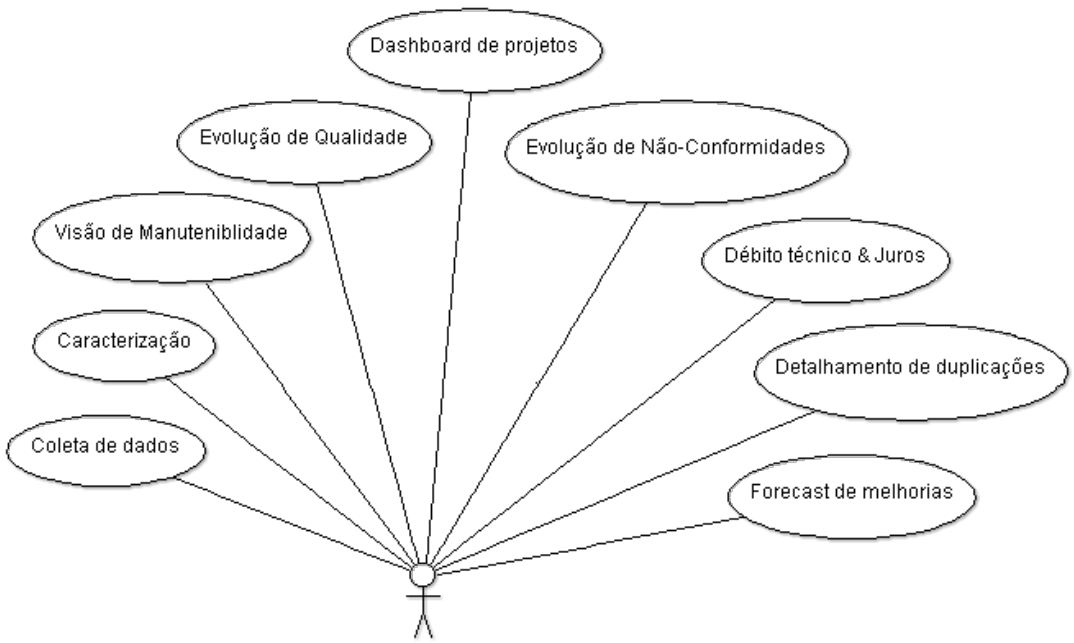
Atualmente a RADAR utiliza o PMD[2] para análise de defeitos do projeto, bem como, o uso do CPD para a detecção de *copy-and-paste* no código-fonte por meio do componente RADAR-PMD.

[2] https://pmd.github.io/

A.1 VISÃO FUNCIONAL

Esta visão subsidia o entendimento do que é possível realizar com a ferramenta do ponto de vista funcional (do usuário).

Figura 46 – RADAR Funcionalidades



Conforme a Figura 46 existem 9 funcionalidades macros, porém, dentro do fluxo de navegação torna-se possível visualizar itens adicionais como: evolução de classes, operações e caracterização dos componentes.

- **Coleta de dados** permite a obtenção do código-fonte, seja por meio de plugins ou entrada manual;
- **Caracterização** analisar o código-fonte checando métricas primitivas e derivadas;
- **Visão de manuteniblidade** deve exibir a qualidade do projeto em termos de manutenção;
- **Evolução de qualidade** deve exibir o status qualitativo do design com base em revisões do projeto;
- **Dashboard de projetos** apresenta uma lista com todos os projetos e a quantidade de revisões ja avaliadas;
- **Evolução de não-conformidades** deve oferecer uma visão histórica e categorizada sobre as não-conformidades do código;

- **Débito tecnico e juros** exibir uma visão estratégica sobre o custo finsnceiro em termos de homem/hora e evoluções do débito técnico;
- **Detalhamento das duplicações** exibir de forma analítica pontos de duplicidade em classes do projeto;
- **Forecast de melhorias** apresentar opções de correção de classes baseando-se em critérios de eligibilidade;

A.2 VISÃO LÓGICA

Tecnicamente, a componentização do projeto visa a extensibilidade e a flexibilidade, seguindo direcionamentos conforme padrões de projeto e boas práticas, utilizando as seguintes tecnologias:

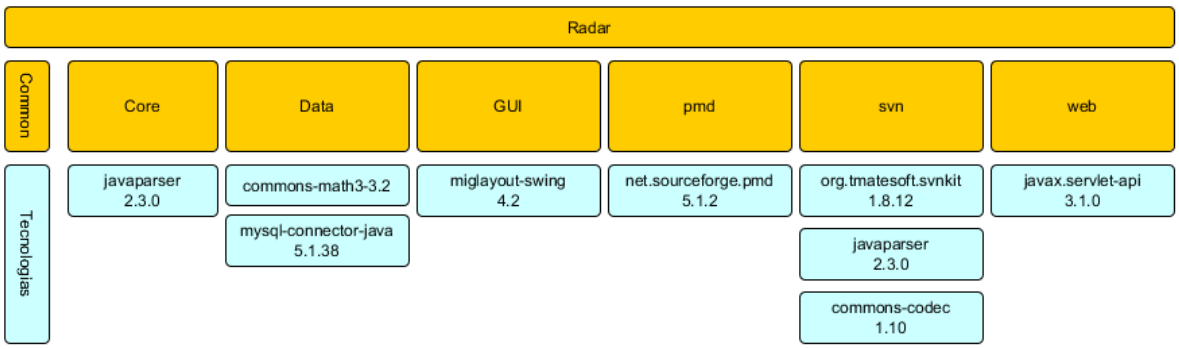
- Camada integração e negócio: Java
- Camada de apresentação: Swing / Jsp / Servlet 2.5;
- Estrutura de implementação e build: Maven 3.3.9
- Versionamento: Bitbucket: https://andersonfonseka@bitbucket.org/andersonfonseka/refviz.git

Conforme a Figura 47, o empacotamento segue um modelo baseado em módulos sendo dividido em 8 partes, o que permite uma maior extensibilidade e facilidade de manutenção, contendo as seguintes responsabilidades:

- *Radar-commom* incui as peças de domínio e objetos comuns compartilhados entre outras partes da solução;
- *Radar-core* contém todo o motor de cálculo e análise das métricas;
- *Radar-data* suporta a conexão, guarda e busca das informações;
- *Radar-pmd* encapsula e adapta o componente PMD e CPD para a análise dos defeitos e duplicidades de código;
- *Radar-SVN* habilita o mecanismo de acesso aos códigos-fontes contidos no SVN;
- *Radar-GUI* entrega uma aplicação swing para o envio das métricas ao servidor;
- *Radar-Maven-Plugin* utilizando em conjunção com o Jenkins para o envio dos dados no momento da execução das *taskjobs*;
- *Radar-Web* apresenta o dashboard com os projetos coletados pela solução.

A implementação da análise apresentada na Seção 4.4.2 ainda se encontra em fase inicial de análise e projeto.

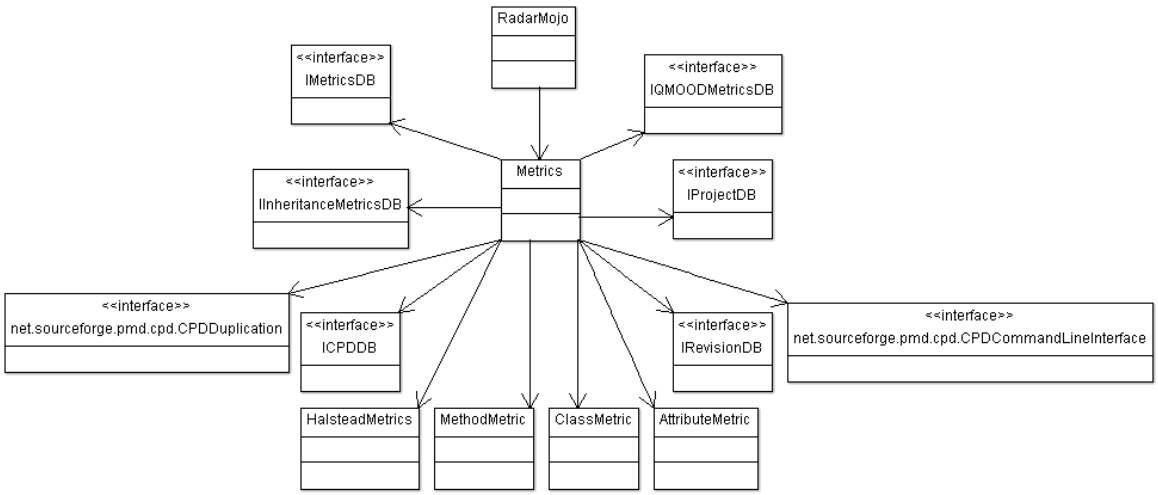
Figura 47 – RADAR Componentes



A.2.1 Visão de classes participantes (VoPC) - Coletar dados

Esta visão nos permite entender quais os componentes estão relacionados a funcionalidade de **Coleta de dados** por meio do plugin desenvolvido para uso com o Jenkins, conforme a Figura 48.

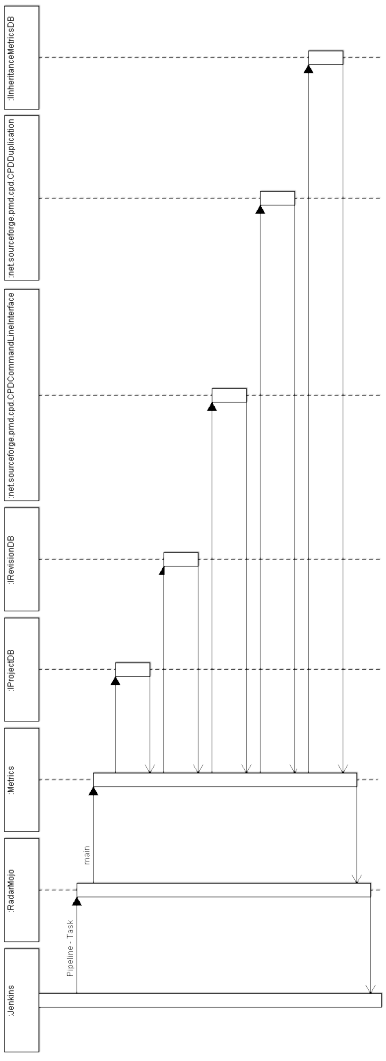
Figura 48 – RADAR VoPC - Coletar dados



A.2.2 Fluxo de execução - Coletar dados

O objetivo desse fluxo está apresentar a colaboração entre as classesde forma dinâmica na execução da funcionalidade **Coletar dados**, conforme a Figura 49.

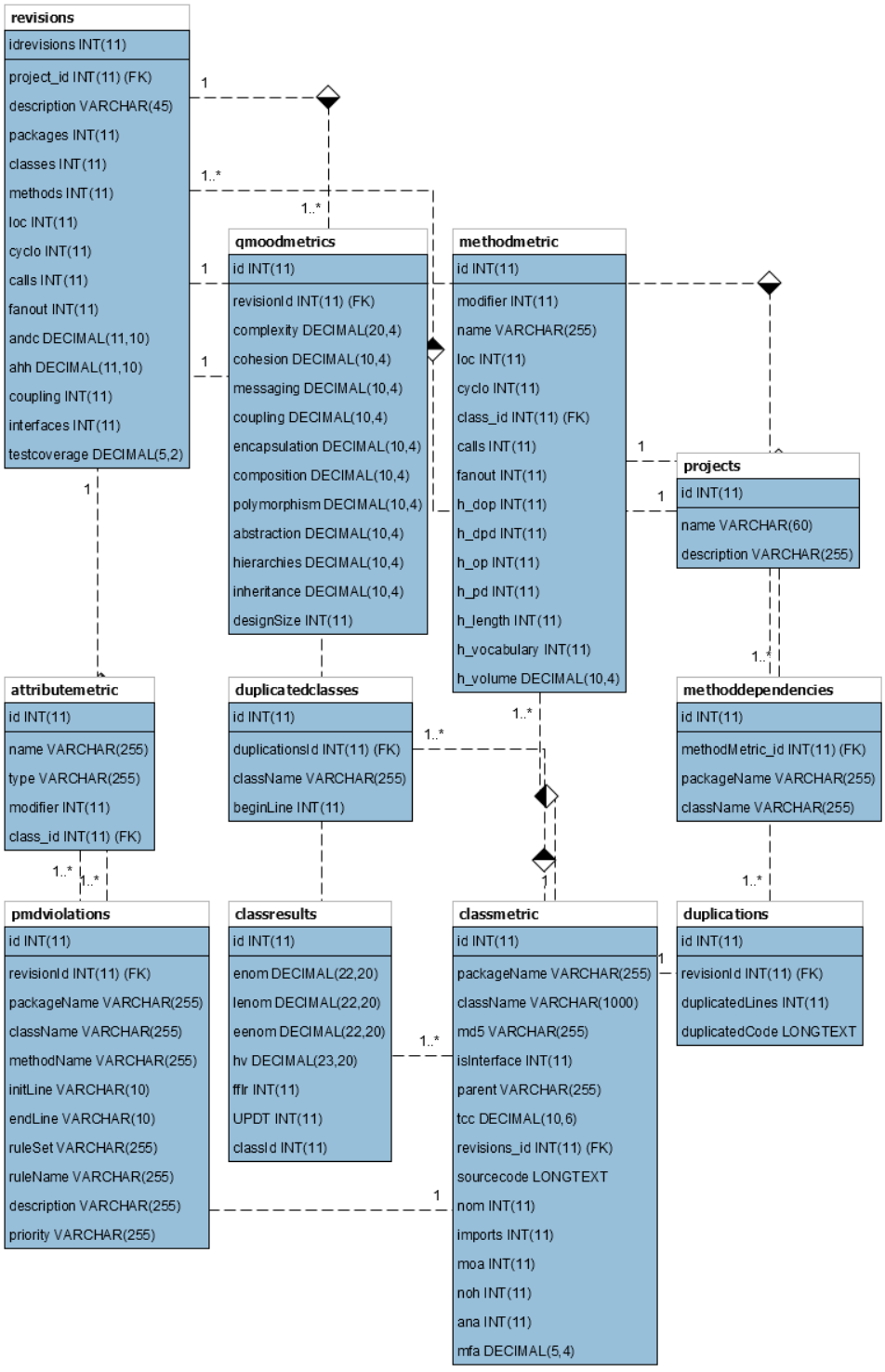
Figura 49 – RADAR Sequencia - Coletar dados



A.3 VISÃO DE DADOS

Conforme a Figura 50, o modelo de dados da solução contém as principais entidades que suportam a coleta e analise dos dados. O banco de dados MySQL 5.7 foi selecionado por se tratar de uma solução gratuíta em sua distribuição community.

Figura 50 – RADAR Modelo de dados



A.4 DISCUSSÃO

Esta ferramenta foi desenvolvida e utilizada dentro da consultoria de TI na qual foram estudados, direcionados e aplicados os métodos descritos nesta tese. Por meio dela, foram avaliados por volta de 12 projetos de um cliente na área de telecomunicações sendo posteriormente apresentando ao comitê executivo responsável pela vertical de telecom dentro da consultoria.

Neste contexto, a RADAR foi apresentada ao cliente responsável pelos projetos avaliados, tendo como resultado a aceitação e o entendimento de qualidade na evolução dos projetos.

Recentemente a RADAR foi hospedada em um servidor na nuvem como um piloto para que o cliente possa direcionar sua equipe com relação à qualidade dos projetos. Internamente a consultoria estuda um plano de utilização de forma global, porém, este processo ainda está incipiente.

APÊNDICE B – QUESTIONÁRIO PARA AVALIAÇÃO DE TIMES DE DESENVOLVIMENTO

Este questionário foi elaborado para a avaliação dos times em projetos de desenvolvimento e manutenção de software dentro da consultoria de TI, contendo 43 questões categorizadas conforme a Tabela 32.

Tabela 32 – Questionário para avaliação de times de desenvolvimento

Experiência Técnica em Projetos
Em quantos projetos (não acadêmicos) de desenvolvimento você já participou ?
Quantos meses de experiência você tem com o JavaServer Faces (JSF)?
Quanta experiência de modelagem de dados você tem?
Qual a sua experiência com testes unitários e testes de integração?
Qual a sua experiência no design de aplicativos?
Qual a sua experiência com design de OO/classe e UML?
Até que nível você se sente confortável trabalhando com JavaServer Pages (JSP)?
Quantos meses de experiência você tem com a API Java para XML Web Services (JAX-WS)?
Quantos meses de experiência você tem com a arquitetura Java para XML Binding (JAXB)?
Quantos meses de experiência você tem com JDBC?
Quanta experiência você tem com mecanismos de regras (Drools, Ilog, JRules, Pega PrPC, Blaze, etc.)?
Quantos meses de experiência você tem com Enterprise JavaBeans (EJBs)?
Qual a sua experiência com atividade de refinamento de código (Tunning de código, Ajuste de Desempenho)?
Quantos meses de experiência você tem com o Java Persistence API (JPA)?
Quantos meses de experiência você tem com o Java Persistence API (Hibernate)?
Participação dos Treinamentos
Como você avalia a frequência dos treinamentos oferecidos?
Como você avalia a qualidade dos treinamentos oferecidos?
Como você avalia o seu grau de absorção dos conteúdos repassados durante o treinamento?
Os prazos de entrega atrapalham a absorção de conhecimento?
Domínio de Ferramentas
Como você avalia o domínio das ferramentas de cobertura de código (SONAR, PMD, FindBugs e CheckStyle)?
Como você avalia o domínio das ferramentas de controle de versão (VSS, CVS, SVN, Git, Mercurial, Bazaar, RTC, etc.)?
Como você avalia o domínio das ferramentas de desenvolvimento que você tem (Maven, Ant, Gradle, etc.)?
Como você avalia o domínio do SQL Developer?
Como você avalia o domínio do Eclipse?
Como você avalia o domínio do Weblogic?
Como você avalia o domínio do SoapUI?
Como você avalia o domínio do Putty?
Como você avalia o domínio do WinSCP?
Como você avalia o domínio do KeePass?
Entendimento do Negócio / Funcional
Como você avalia o entendimento de negócio (funcional) do que precisa ser feito como um todo?
Como você avalia as conversas funcionais com os Arquitetos?
Como você avalia a qualidade dos documentos ITS?
Nível de Ocupação
Como você avalia o seu nível de ocupação?

APÊNDICE C – ISO912 CARACTERÍSTICAS DE MANUTENIBLIDADE E MÉTRICAS ASSOCIADAS

Tabela 33 – Manuteniblidade ISO9126 x Métricas

No. Sub-características	Siglas	Métrica	
1	Analysability	V	Volume
2	Analysability	MLOC	Method Lines of Code
3	Analysability	UT	Unit Test
4	Analysability	DUP	Duplications
5	Changeability	WMC	Weighted Methods per Class
6	Changeability	DUP	Duplications
7	Stability	UT	Unit Test
8	Testability	A, RMA	Abstractness
9	Testability	Ca, AC	Afferent Couplings
10	Testability	CAM, CAMC	Cohesion Among Methods
11	Testability	CBO	Coupling Between Objects
12	Testability	Ce	Efferent Couplings
13	Testability	DIT	Depth of Inheritance Tree
14	Testability	Dn, DMS, RMD, D	Normalized Distance from Main Sequence
15	Testability	I, RMI	Instability
16	Testability	LCOM	Lack of Cohesion in Methods
17	Testability	LOC, NLOC	Number of Lines of Code
18	Testability	MLOC	Method Lines of Code
19	Testability	NEST, BD, MAX-NEST, NBD	Nesting level
20	Testability	NMO, NOO, NORM, NOVM, NOOC	Number of Overriden Methods
21	Testability	NOA, NOF, NA, NF	Number of Attributes
22	Testability	NOC, NSC	Number of Children
23	Testability	NOM, NM, MPC, NCM, NOO, TNM	Number of Methods
24	Testability	NPAR, PAR	Number of Parameters
25	Testability	NSF	Number of Static Attributes
26	Testability	NSM	Number of Static Methods
27	Testability	RFC	Response for a Class
28	Testability	SIX, SI	Specialization index
29	Testability	V(G), CC, MVG, CYCLO	McCabe Cyclomatic Complexity
30	Testability	WMC	Weighted Methods per Class
31	Testability	UT	Unit Test