Aferição da Qualidade do Código-Fonte com apoio de um ambiente de *Data Warehousing* na gestão de contrato ágil: um estudo de caso preliminar em uma autarquia da Administração Pública Federal

Hilmer Rodrigues Neri[1]Guilherme Baufaker Rêgo[1], Aline Gonçalves do Santos[1]

[1]Faculdade UnB Gama – Universidade de Brasília

hilmer@unb.br, gbre.111@gmail.com, alinegsantoss@gmail.com

Abstract. *Some Brazilian Federal organizations have started to use agile methods on software development contracts. In such situation, it has been observed a weakness on measuring internal quality of outsourced software. This work proposes an automated solution, based on data warehousing environment, which may measure the quality of source code in order to be approved or rejected by a contractor. Therefore, this paper conducted a initial case study, as an empirical working evidence of that automated solution, that evaluated the 24 monthly releases of Sistema Integrado de Gestão e Conhecimento (SIGC) from IPHAN, the National Institute of Historic and Artistic Heritage, which had represented more than 39.000 source code lines evaluated.*

Resumo. *Algumas organizações da Administração Pública Federal iniciaram investimentos para adotar contratações de serviços de desenvolvimento de software utilizando métodos ágeis. Nesse contexto, percebeu-se a oportunidade de melhorar a capacidade de organizações públicas em aferir a qualidade do produto de software contrato. O objetivo deste trabalho foi propor uma solução automatizada, apoiada em um ambiente de data warehousing, que pudesse aferir a qualidade do código-fonte de forma a apoiar a aceitação ou rejeição do produto por parte da contratante. Visando a validação empírica da solução automatizada, realizou-se um estudo de caso inicial no qual foram avaliadas as 24 releases mensais do Sistema Integrado de Gestão e Conhecimento (SIGC) do IPHAN no qual foram avaliadas mais de 39.000 linhas de código-fonte.*

Palavras-chave: *Código-fonte, contratações, métodos ágeis, IN04, qualidade do código-fonte, qualidade interna do produto, código-limpo, estudo de caso, administração pública federal.*

1. Introdução

No desenvolvimento de software, há diversas variáveis, quer sejam de natureza ambiental ou técnica, que provavelmente impactarão desde a execução do processo de análise de requisitos até a implantação do software [Beck 1999]. Essa característica torna o processo de desenvolvimento pouco previsível e complexo, requerendo flexibilidade e personalização para ser capaz de responder às mudanças. Em consonância a esse movimento, órgãos da Administração Pública Federal (APF) também têm aderido a utilização de metodologias ágeis nos processos de desenvolvimento de software.

Recentemente, o Tribunal de Contas da União publicou o Acórdão no 2314/2013 [BRASIL 2013a] que abordou a experiência do uso de práticas ágeis utilizadas na contratação realizadas por instituições públicas federais, como por exemplo, Banco Central do Brasil (BACEN), Instituto do Patrimônio Histórico e Artístico Nacional (IPHAN) Tribunal Superior do Trabalho (TST).

A Instrução Normativa nº 4 [BRASIL 2010], que versa sobre o processo de Contratação de Soluções de Tecnologia da Informação pelos órgãos integrantes do Sistema de Administração dos Recursos de Informação e Informática do Poder Executivo Federal, enuncia um conjunto de boas práticas e um modelo de processo de contratações conforme visto na Figura 1.

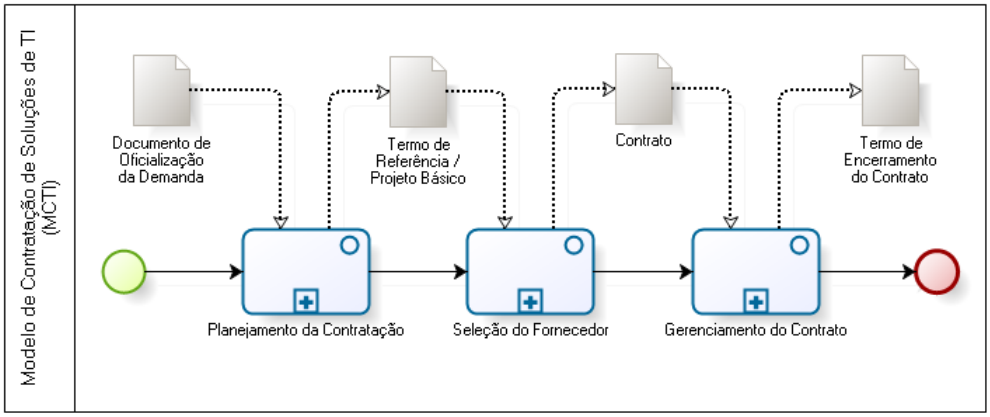


Figura 1. Modelo de Processo de Contratações de Soluções de TI [BRASIL 2013b]

A fase de gerenciamento do contrato visa a acompanhar e garantir a adequada prestação do serviço e o fornecimento de bens que compõem a solução de TI, sendo que está disposto no art. 25, alínea b), que a avaliação da qualidade dos serviços realizados ou dos bens entregues e justificativas deve estar de acordo com os Critérios de Aceitação definidos em contrato, a cargo dos Fiscais Técnico e Requisitante do Contrato [BRASIL 2010].

Considerando: i) o princípio ágil de valoração de produto sobre documentação abrangente; ii) o foco na qualidade interna e iii) a necessidade das entidades da Administração Pública Federal do Poder Executivo aferirem a qualidade do produto entregue por seus fornecedores de desenvolvimento de software, conforme previsto na alínea b) do art. 25 da [BRASIL 2010], o objetivo desta pesquisa foi:

Analisar	**o código-fonte**
com o propósito de	**aferir a qualidade interna**
com respeito à	**aceitação do produto**
do ponto de vista do	**fiscal técnico do contrato**
no contexto da	**contratação de desenvolvimento de software por parte da Administração Pública Federal**

2. Medição da Qualidade Interna do Produto

A qualidade interna do produto de software pode ser medida, ainda em ambiente de desenvolvimento, por meio da avaliação de estruturas internas que compõem o sistema de software [ISO/IEC 25023 2011], sendo que as métricas extraídas diretamente do código-fonte são bons indicadores de qualidade interna de produto [Beck 2003], pois são métricas objetivas e com características como validade, simplicidade, objetividade, fácil obtenção e robustez [Mills 1999].

Para a escolha das métricas utilizadas neste trabalho, consideramos: i) os estudos encontrados na literatura sobre métricas de código-fonte, entre eles o trabalho de Marinescu [Marinescu 2005] um dos mais reconhecidos nesta área; ii) a existência de estudos que definem valores de referência para análise das métricas dos conceitos mais relevantes à manutenibilidade do software como o tamanho, complexidade interna, modularidade e grau de depedência entre módulos [Meirelles 2013]; iii) métricas que reflitam indicadores sobre boas práticas de programação, a exemplo da aderência a padrões de projeto e boas práticas de programação [Machini et al. 2010]; iv) a existência das métricas na ferramenta de análise estática de código-fonte utilizada neste trabalho. Essas métricas escolhidas foram reunidas e são apresentadas na Tabela 1.

Tabela 1. Conjunto de Métricas de Código-Fonte

Métrica	Descrição
ACCM (*Average Cyclomatic Complexity per Method*)	Essa métrica pode ser representada através de um grafo de fluxo de controle, sendo que o uso de estruturas de controle, tais como, {if, else, while} aumentam a complexidade ciclomática de um de um método [McCabe 1976].
ANPM (*Average Number of Parameters per Method*)	Calcula a média de parâmetros dos métodos da classe [Basili and Rombach 1987].
AMLOC (*Average Method Lines of Code*)	Indica se o código está bem distribuído entre os métodos [Meirelles 2013].
CBO (*Coupling Between Objects*)	Número total de classes dentro de um pacote que dependem de classes externas ao pacote [Chidamber and Kemerer 1994].
NOC (*Number of Children*)	Número de subclasses ou classes filhas que herdam da classe analisada [Rosenberg and Hyatt 1997].
NPA (*Number of Public Attributes*)	Mede o encapsulamento de uma classe, pois é recomendado que os atributos devem ser manipulados por meio dos métodos de acesso [Beck 1997].
LCOM4 (*Lack of Cohesion in Methods*)	Número de componentes (métodos e atributos) fracamente conectados a uma classe [Hitz and Montazeri 1995].
RFC (*Response For a Class*)	Número de métodos dentre todos os métodos que podem ser invocados em resposta a uma mensagem enviada por um objeto de uma classe [Sharble and Cohen 1993].

Em um trabalho recente sobre a análise de métricas de código-fonte, foi observado o comportamento estatístico das métricas de código-fonte de 38 projetos de software livre com mais de 100.000 downloads em um esforço de análise de mais de 300.000 classes. Entre os softwares analisados estão Tomcat, OpenJDK, Eclipse, Google Chrome, VLC e outros que foram desenvolvidos em C, C++ e Java [Meirelles 2013]. Neste trabalho, concluiu-se empiricamente que há valores de referência para métricas de código-fonte, que foram classificados como muito frequentes, frequentes, pouco frequentes e não frequentes para softwares escritos em uma mesma linguagem de programação. Nos trabalhos como os de Marinescu [Marinescu 2005], Moha [Moha et al. 2010] e Rao [Rao and Reddy 2007], foi mostrado que é possível utilizar métricas de código-fonte como forma de detecção de trechos de código-fonte que podem ser melhorados com a refatoração [Fowler 1999]. Seguindo a abordagem de Marinescu, Machini [Machini et al. 2010] mapeou as técnicas e práticas de limpeza de código-fonte propostas por [Martin 2008] e [Beck 2007] de forma a constuir cenários de limpeza de código-fonte que são recomendações ou práticas para eliminar um determinado trecho de código-fonte não coeso. No trabalho de [Machini et al. 2010], não foi utilizada nenhuma automatização para cálculo das métricas e posterior identificação de um cenário de limpeza.

De posse da análise estatística efetuada por Meirelles [Meirelles 2013], agregamos as configurações dos intervalos de métricas como a forma de detecção dos cenários de limpeza propostos por [Machini et al. 2010], tal como se é possível observar na Tabela 2.

Tabela 2. Detecção dos Cenários de Limpeza de Código-Fonte

Cenário de Limpeza	Características	Recomendações	Forma de Detecção
Classe Pouco Coesa	Classe Subdivida em grupos de métodos que não se relacionam.	Distribuir as responsabilidades da classes em métodos.	Intervalos Pouco Frequente e Não Frequente de LCOM4, RFC.
Interface dos Métodos	Elevada Média de parâmetros repassados pela Classe.	Minimizar o número de Parâmetros.	Intervalos Pouco Frequente e Não Frequente de ANPM.
Classes com muitos filhos	Muitas Sobreescritas de Métodos	Trocar a Herança por uma Agregação.	Intervalos Pouco Frequente e Não Frequente de NOC.
Classe com métodos grandes e/ou muitos condicionais	Grande Número Efetivo de Linhas de Código	Reduzir o número de linhas dos seus métodos, Quebra de métodos.	Intervalos Pouco Frequente e Não Frequente de AMLOC, ACCM.
Classe com muita Exposição	Grande Número de Parâmetros Públicos	Reduzir o Número de Parâmetros Públicos.	Intervalos Pouco Frequente e Não Frequente de NPA.
Complexidade Estrutural	Grande Acoplamento entre Objetos	Reduzir a aquantidade de responsabilidades dos Métodos.	Intervalos Pouco Frequente e Não Frequente de CBO e LCOM4.

A partir da identificação dos cenários de limpeza de código-fonte e da contagem do número de classes em uma determinada *release* do software, criamos a Taxa de Aproveitamento de Oportunidade de Melhoria de Código-Fonte (T_r) como um indicador objetivo de monitoramento da oportunidade de melhoria da qualidade do código-fonte, sendo este descrito conforme a Equação 1, em que Ce_r é o total de cenários de limpeza na release e Cl_r é a quantidade classes na mesma release.

$$T_r = \frac{Ce_r}{Cl_r} \tag{1}$$

3. Solução para Automação do Processo de Medição da Qualidade Interna do Produto

Visando atingir a automação do processo de medição de qualidade interna do produto por meio de técnicas não intrusivas no ambiente de desenvolvimento de software, realizou-se uma revisão bibliográfica da literatura em busca de ambientes de informação que permitissem o uso de ferramentas (não intrusivas) para automatizar a coleta e o uso de sofisticadas técnicas de análise de dados [Gopal et al. 2005] . Trabalhos como o de Palza [Palza et al. 2003], Ruiz [Ruiz et al. 2005], Castellanos [Castellanos et al. 2005], Becker [Becker et al. 2006], Folleco [Folleco et al. 2007] e Silveira [Silveira et al. 2010], evidenciaram que ambientes de *Data Warehousing* são boas soluções para se automatizar um programa de métricas em processos de desenvolvimento de software.

Data Warehousing é uma coleção de tecnologias de suporte à decisão disposta a capacitar os responsáveis por tomar decisões a fazê-las de forma mais rápida [Chaudhuri and Dayal 1997]. Em outras palavras, trata-se de um processo para gerenciar dados vindos de várias fontes, com o objetivo de prover uma visão analítica de parte ou de todo o negócio em repositório central chamado de *data warehouse* [Gardner 1998] [Kimball and Ross 2002]. A arquitetura de um ambiente de *data warehousing* pode ser vista na Figura 2.

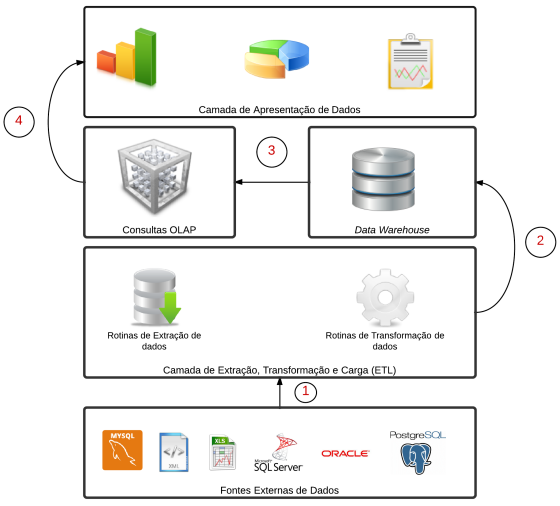


Figura 2. Arquitetura do Ambiente de *Data Warehousing*

Na Figura 2, as setas 1 e 2 representam o processo de *Extraction-Transformation-Load (ETL)* que é o processo de extração, transformação e carga dos dados de forma automática e não intrusiva ao ambiente de desenvolvimento. Este processo pode consumir até 85% de todo o esforço em um ambiente de *Data Warehousing* [Kimball and Ross 2002]; A seta 3 representa as consultas *On-Line Analytical Processing (OLAP)* que são operações de consulta e análise realizadas sobre *data warehouse* projetado sobre um modelo dimensional [Kimball and Ross 2002] [Codd et al. 1993]; A seta 4 representa a visualização dos dados em forma de gráficos, tabelas ou painéis customizáveis conhecidos como *dashboards*.

A implementação do ambiente de *data warehousing* mostrado na figura 2 se deu pela ferramenta Analizo[1], responsável por coletar as métricas de código-fonte, a suite Pentaho BI Community[2], que dispõe de ferramentas de ETL e OLAP, um banco de dados MariaDB[3], o plugin Saiku Analitics[4] para visualização de dados.

4. Estudo de Caso

Visando à validação da solução apresentada na Seção 3, foram consultadas as instituições citadas pelo Acórdão nº 2314/2013 [BRASIL 2013a]. Entre elas, o Instituto do Patrimônio Histórico e Artístico Nacional (IPHAN), autarquia da administração pública federal responsável pela gestão de diversos processos de preservação do patrimônio cultural, permitiu o acesso ao processo, documentos e o código-fonte do SICG, um dos primeiros softwares desenvolvidos sob contrato de terceirização de serviços com a utilização de metodologias ágeis. Essa aplicação foi desenvolvida na linguagem Java com a utilização de *frameworks* como VRaptor e Hibernate, durante 24 *releases* mensais. Possui um total 39.790 linhas de código distribuídas em 914 classes.

4.1. Protocolo do Estudo de Caso

Seguindo a metodologia proposta por Wohlin [Wohlin et al. 2012], foi construído um protocolo de estudo de caso, baseado em Brereton [Brereton et al. 2008], quanto aos aspectos gerais e Yin [Yin 2011] com relação as ameaças à validade do estudo.

O objeto do estudo de caso é o ambiente de *Data Warehousing* (DWing) projetado e implementado conforme apresentado na seção 3. A unidade de análise foi o código fonte do SICG. A partir do objetivo da pesquisa, foram elaborados objetivos específicos do estudo de caso utilizando o *Goal Question Metric* (GQM) [Basili et al. 1996]. Nesta abordagem cada objetivo específico originou uma série de questões específicas respondidas com métricas específicas, tal como se mostra na Tabela 3.

[1]Disponível em `http:/http://analizo.org/`
[2]Disponível em `http://community.pentaho.com/`
[3]Disponível em `http://mariadb.org/`
[4]Disponível em `http://meteorite.bi/saiku`

Tabela 3. Objetivos Específicos do Estudo de Caso

Objetivo Específico	Questão Específica	Métricas Específicas
OE1 - Automatizar o processo de medição da qualidade do código-fonte do produto de software	QE1 - Quais e Quantos são os passos necessários para medir a qualidade do código-fonte do produto de software de forma automatizada?	M1 - Quantidade de passos do processo de medição da qualidade do código-fonte automatizados.
OE2 - Avaliar a qualidade global do código-fonte do software	QE2 - Quais são os indicadores de código-limpo em um código-fonte de um determinado projeto?	M2 - Quantidade de Cenários de Limpeza identificados em uma determinado projeto
OE3 - Avaliar a qualidade local do código-fonte do software	QE3 - Quais são os indicadores de código-limpo em um código-fonte de uma determinada classe/módulo?	M3 - Quantidade de Cenários de Limpeza identificados em uma determinada classe/módulo
OE4 - Medir qual é o aproveitamento de oportunidades de melhoria de código-Fonte	QE4 - Qual é o nível de qualide interna interna do produto, para efeito de sua aceitação por parte do fiscal técnico do contrato?	M4 - Taxa de Aproveitamento de Oportunidades de Melhoria de Código-Fonte.

Com relação às ameaças citadas por Yin, a ameça à validade de construção foi mitigada com utilização da abordagem GQM, uma vez que esta estratégia estabelece uma lógica que une as proposições às métricas utilizadas no estudo. A validade interna é obtida quando se consegue observar as relações causais e todos os elementos que as compõem. No presente estudo de caso, procuramos mitigar essa ameaça realizando a análise das variáveis C_e e C_l ao longo de 24 meses, identificando sua relação causal. Já em relação T_r as conclusões ainda são incipientes. Quanto à validade externa, destaca-se que a utilização de um estudo de caso não é suficiente para generalizar os resultados deles obtidos, sendo necessária a utilização de estudo em múltiplos casos, a fim de aumentar o poder de generalização [Yin 2011]. Com relação à confiabilidade, a partir da documentação da implementação do ambiente de *Data Warehousing* conjuntamente com o protocolo de estudo de caso e as bases de dados das métricas de código-fonte, garante-se a repetição do estudo de caso e, por conseguinte, a confiabilidade.

4.2. Execução e Resultados do Estudo de Caso

Para analisar os cenários de limpeza de código-fonte, foram extraídas as métricas de cada classe e analisadas conforme a Tabela 2. Para cada *release*, foram identificados, conforme a Figura 3, os cenários de limpeza de código-fonte. Para se mostrar maior nível de detalhes dos cenários de limpeza de código-fonte, realizou-se uma consulta OLAP de *Drill-Down*, que tem objetivo de expor mais detalhes na visualização dos dados [Kimball and Ross 2002], sobre o modelo dimensional do *data warehouse*.

	Sistema Integrado de Controle e Gestão																		
Nome do Cenario de Limpeza	OS02	OS04	OS05	OS07	OS08	OS09	OS10	OS11	OS12	OS13	OS14	OS16	OS17	OS19	OS20	OS21	OS22	OS23	OS24
Classe Pouco Coesa	4	9	11	20	26	29	36	47	51	54	55	62	65	66	70	71	73	74	75
Classe com muita Exposição										1	1	1	1	1	1	1	1	2	2
Classe com métodos grandes e/ou muitos condic	1	1		1	3	3	4	4	6	13	15	16	19	22	20	22	22	19	19
Classes com muitos filhos	1	1	1	1	1	1	1	2	2	3	3	5	5	5	5	5	5	5	5
Complexidade Estrutural	15	33	40	64	85	90	108	146	151	158	169	193	201	211	220	226	229	229	230
Interface dos Métodos	6	8	8	8	16	15	24	37	44	48	47	52	56	58	61	64	64	66	66

Figura 3. Total de Cenários de Limpeza de Código-Fonte identificados por cenário e *Release*

Conforme é possível observar na figura 3, foram detectados mais cenários de limpeza de código-fonte dos tipos **Complexidade Estrutural**, **Classe Pouco Coesa** e **Interface dos Métodos** respectivamente. Os três Cenários de Limpeza com menor número de incidências foram **Classe com Muita Exposição**, **Classe com Muitos Filhos** e **Classe com Métodos Muito Grande e/ou com muitos condicionais**.

Ao analisarmos os dados conforme apreentado na figura 4-(a) e (b), encontramos uma correlação positiva perfeita entre a quantidade de classes e quantidade de cenários, ou seja, ambas cresceram proporcionalmente na mesma direção, onde r=0,9972. Isso indica que o esforço de refatoração realizado não foi proporcional às necessidades de melhoria do código, nem tampouco direcionado ao tipo de oportunidade de melhoria de maior incidência. Outro sim, é possível inferir que houve pouca atenção da equipe em tomar em boas decisões de projeto(*desing*), em relação a aspectos de modularidade, uma vez que o cenário de complexidade estrutural apresentou uma incidência de 58% dentro os total de cenários identificados.

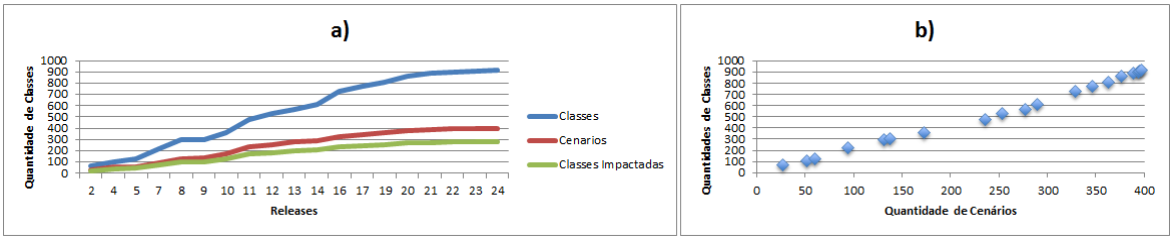
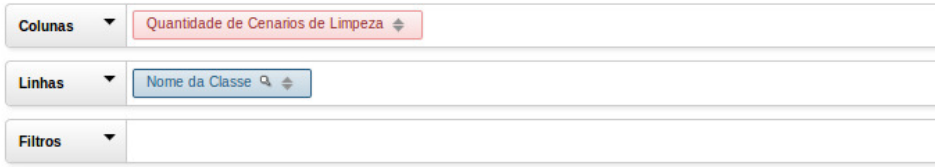


Figura 4. (a) Crescimento de classes e cenários ao longo das Releases - (b) Correlação entre quantidade de classes e quantidade de cenários

Considerando que é importante conhecer as classes com maior incidência de problemas com relação limpeza de código-fonte, foram identificadas, como se mostra na Figura 5, as 10 classes que apresentaram a maior quantidade de cenários de limpeza de código-fonte. Para se obter a informação desta consulta foram necessárias uma consulta OLAP de *Drill Down* sobre algumas dimensões e outra de *Slice and Dice*, que tem o objetivo de selecionar os dados, sobre o resultado a consulta de *Drill-Down*.



Nome da Classe	Quantidade de Cenarios de Limpeza
br::gov::iphan::sicg::apresentacao::controllers::BemProtecaoController	54
br::gov::iphan::sicg::apresentacao::controllers::BemImovelCaracterizacaoInternaController	48
br::gov::iphan::sicg::apresentacao::controllers::AcaoController	46
br::gov::iphan::sicg::apresentacao::controllers::BemImovelCaracterizacaoExternaController	46
br::gov::iphan::sicg::apresentacao::controllers::BemImaterialController	45
br::gov::iphan::sicg::apresentacao::controllers::BemMultimidiaController	43
br::gov::iphan::sicg::persistencia::dao::impl::BemDAOImpl	40
br::gov::iphan::sicg::apresentacao::controllers::BemController	39
br::gov::iphan::sicg::apresentacao::controllers::ContextoImediatoController	39
br::gov::iphan::sicg::apresentacao::controllers::BemImovelLoteController	38

Figura 5. As 10 classes com maior número identificado de Cénarios de Limpeza

Com a quantidade de classes e o total de cenários de limpeza de código-fonte, foi possível calcular a Taxa de Aproveitamento de Oportunidade de Melhoria de Código-Fonte por cada release do software conforme se mostra na Figura 6.

Nome do Projeto	OS	Numero de Classes	Quantidade de Cenarios de Limpeza de Codigo	Taxa de Aproveitamento de Oportunidades de Melhoria de Codigo
Sistema Integrado de Controle e Gestão	OS02	69	27	0,39
	OS04	104	52	0,50
	OS05	125	60	0,48
	OS07	222	94	0,42
	OS08	294	131	0,45
	OS09	300	138	0,46
	OS10	360	173	0,48
	OS11	474	236	0,50
	OS12	532	254	0,48
	OS13	568	277	0,49
	OS14	613	290	0,47
	OS16	730	329	0,45
	OS17	769	347	0,45
	OS19	805	363	0,45
	OS20	861	377	0,44
	OS21	889	389	0,44
	OS22	899	394	0,44
	OS23	908	395	0,44
	OS24	914	397	0,43

Figura 6. Taxa de Aproveitamento de Oportunidades de Melhoria de Código-Fonte

Ao analisarmos a taxa de oportunidade de melhoria, podemos inferir que: i) apesar do crescimento dos cenários, há uma tendência de estabilidade da taxa a partir da release 16. Isso pode indicar que a equipe adquiriu maturidade sobre o projeto e passou a controlar a qualidade interna, apresentando inclusive, uma suave melhoria. ii) a melhoria não foi significativa em relação às necessidades. Contudo, não houve uma piora desproporcional entre o crescimento do tamanho do código-fonte e dos cenários de oportunidade de limpeza.

A idéia é que a taxa possa ser utilizada como um indicador de qualidade interna do código-fonte e que o fiscal técnico do contrato possa aferí-la e monitorá-la ao longo do tempo de forma a utilizar essa informação para apoiá-lo na atividade do ateste técnico da solução contratada. A forma de interpretar a taxa é: ao se aproximar de zero, o valor

de T_r deve indicar maior qualidade interna do produto. Já ao se aproximar de 1, deve indicar menor qualidade interna. Contudo, no presente estudo não obtivemos conclusões relevantes a cerca da taxa, que necessita de um estudo estatístico mais profundo, para por exemplo, definir valores de referência para sua interpretação.

A principal dificuldade encontrada com o uso do ambiente de DWing foi o esforço necessário para a construção camada de ETL, amplamente relatado na literatura de referência desta área. Cabe registrar que a definição do esquema do DW pode impactar diretamente as transformações já construídas na camada de ETL. Ou seja, toda vez que realizamos alguma alteração no esquema, tivemos que alterar as transformações previamente construídas na camada de ETL. Por outro lado, nos beneficiamos da facilidade de automação, flexibilidade para consultas e visualização, disponíveis no DWing.

5. Conclusões e Trabalhos Futuros

Com relação a execução do estudo de caso, foi possível mostrar que a automação do processo de medição da qualidade interna do produto de software (objetivo específico OE1) é possível de ser realizada de maneira contínua e automatizada por meio do ambiente de *data warehousing*, pois além da automação da coleta de um volume de dados de díficil verificação manual (39.000 linhas de código-fonte e 914 classes na 24ª release), foram automatizados desde a interpretação dos cenários de limpeza de código-fonte, o cálculo da taxa de aproveitamento das oportunidades de melhoria do código-fonte, até a exibição dos dados em tabelas e gráficos.

Com os resultados apresentados na Seção 4.2, verifica-se que uma outra contribuição ambiente de *data warehousing* foi alcançada por meio das consultas OLAP que permitiram flexibilidade necessária para se avaliar tanto em nível de projeto (objetivo específico OE2), quanto em nível local (objetivo específico OE3) a saúde do projeto com relação a qualidade do código-fonte. Por meio de um ambiente de *data warehousing* semelhante ao apresentado, os fiscais técnicos dos contratos podem contar com mecanismos que venham automatizar a atividade de aferição da qualidade do código-fonte, de forma seja possível definir níveis de aceitação de produto. Como esse estudo ocorreu na fase final do contrato, não foi possível analisar a eficiência e eficácia da medição do aproveitamento de oportunidades de melhoria de código-Fonte. Entretanto, com a solução apresentada, foi possível medir o comportamento da taxa de aproveitamento de oportunidades de melhoria do código-fonte. Porém, a partir dela, não foi possível fazer conslusões mais objetivas de forma a auxiliar o fiscal técnico quanto a aceitação/rejeição do produto contratado, atendendo assim, parcialmente ao objetivo específico OE4.

Os resultados deste trabalho sugerem que o uso do ambiente proposto pode auxiliar a atividade de aferição da qualidade do produto por parte do fiscal técnico de contratos, além de auxiliá-lo a ter uma atitude propositiva ao apresentar os argumentos questionando a qualidade interna do software desenvolvido. Entretanto, trata-se de um estudo preliminar, que carece de maior aprofundamento e análises.

A partir da análise de uma amostragem de projetos, estatisticamente significativa, estudos futuros poderiam, por exemplo: identificar valores aceitáveis de nível de qualidade interna e identificar valores de referência para a taxa de oportunidade de melhoria de código-fonte.

Referências

Basili, V. R., Caldiera, G., and Rombach, H. D. (1996). *The Goal Question Metric Approach*. Encyclopedia of Software Engineering.

Basili, V. R. and Rombach, H. D. (1987). TAME: Integrating Measurement into Software Environments.

Beck, K. (1997). *Smalltalk Best Practice Patterns. Volume 1: Coding*. Prentice Hall, Englewood Cliffs, NJ.

Beck, K. (1999). Embracing change with extreme programming. *Computer*, 32(10):70–77.

Beck, K. (2003). *Test-driven development: by example*. Addison-Wesley Professional.

Beck, K. (2007). *Implementation patterns*. Pearson Education.

Becker, K., Ruiz, D. D., Cunha, V. S., Novello, T. C., and Vieira e Souza, F. (2006). Spdw: A software development process performance data warehousing environment. In *Proceedings of the 30th Annual IEEE/NASA Software Engineering Workshop*, SEW '06, pages 107–118, Washington, DC, USA. IEEE Computer Society.

BRASIL (1993). *Lei n^a 8.666/93, de 21 de Junho de 1993*. Brasília, Brasil.

BRASIL (2010). *Instrução Normativa n^o 04*. Ministério do Planejamento, Orçamento e Gestão, Brasília, Brasil.

BRASIL (2013a). *Acórdão n^o 2314*. Tribunal de Contas da União, Brasília, Brasil.

BRASIL (2013b). *Modelo de Contratação de Soluções de TI*. Ministério do Planejamento, Orçamento e Gestão, Brasília, Brasil.

Brereton, P., Kitchenham, B., Budgen, D., and Li, Z. (2008). Using a protocol template for case study planning. In *Proceedings of the 12th International Conference on Evaluation and Assessment in Software Engineering. University of Bari, Italy*.

Castellanos, M., Casati, F., Shan, M.-C., and Dayal, U. (2005). ibom: A platform for intelligent business operation management. In *Proceedings of the 21st International Conference on Data Engineering*, ICDE '05, pages 1084–1095, Washington, DC, USA. IEEE Computer Society.

Chaudhuri, S. and Dayal, U. (1997). An overview of data warehousing and olap technology. *ACM Sigmod record*, 26(1):65–74.

Chidamber, S. R. and Kemerer, C. F. (1994). A Metrics Suite for Object-Oriented Design. *IEEE Transactions on Software Engineering*, 20(6):476–493.

Codd, E. F., Codd, S. B., and Salley, C. T. (1993). Providing OLAP (On-Line Analytical Processing) to User-Analysis: An IT Mandate.

Folleco, A., Khoshgoftaar, T. M., Hulse, J. V., and Seiffert, C. (2007). Learning from software quality data with class imbalance and noise. In *Proceedings of the Nineteenth International Conference on Software Engineering & Knowledge Engineering (SEKE 2007), Boston, Massachusetts, USA, July 9-11, 2007*, page 487. Knowledge Systems Institute Graduate School.

Fowler, M. (1999). *Refactoring: improving the design of existing code*. Addison-Wesley Professional.

Gardner, S. R. (1998). Building the. *Communications of the ACM*, 41(9):53.

Gopal, A., Mukhopadhyay, T., and Krishnan, M. S. (2005). The impact of institutional forces on software metrics programs. *IEEE Trans. Softw. Eng.*, 31(8):679–694.

Hitz, M. and Montazeri, B. (1995). Measuring Coupling and Cohesion in Object-Oriented Systems. In *Proceedings of International Symposium on Applied Corporate Computing*.

ISO/IEC 25023 (2011). ISO/IEC 25023: Systems and software engineering - systems and software quality requirements and evaluation (square) - measurement of system and software product quality. International Standard 25023, International Organization for Standardization and International Electrotechnical Commission.

Kimball, R. and Ross, M. (2002). *The Data Warehouse Toolkit: The Complete Guide to Dimensional Modeling*. John Wiley & Sons, Inc., New York, NY, USA, 2nd edition.

Lehman, M. M. (1980). Programs, life cycles, and laws of software evolution. *Proc. IEEE*, 68(9):1060–1076.

Machini, J. a., Almeida, L., Kon, F., and Meirelles, P. R. M. (2010). Código Limpo e seu Mapeamento para Métricas de Código-Fonte.

Marinescu, R. (2005). Measurement and quality in object-oriented design. In *Software Maintenance, 2005. ICSM'05. Proceedings of the 21st IEEE International Conference on*, pages 701–704. IEEE.

Martin, R. C. (2008). *Clean Code: A Handbook of Agile Software Craftsmanship*, volume 1.

McCabe, T. J. (1976). A Complexity Measure. *IEEE Transactions Software Engineering*, 2(4):308–320.

Meirelles, P. R. M. (2013). *Monitoramento de métricas de código-fonte em projetos de software livre*. PhD thesis, Universidade de São Paulo (IME/USP).

Mills, E. E. (1999). Metrics in the software engineering curriculum. *Ann. Softw. Eng.*, 6(1-4):181–200.

Moha, N., Guéhéneuc, Y.-G., Duchien, L., and Le Meur, A.-F. (2010). Decor: A method for the specification and detection of code and design smells. *Software Engineering, IEEE Transactions on*, 36(1):20–36.

Palza, E., Fuhrman, C., and Abran, A. (2003). Establishing a generic and multidimensional measurement repository in cmmi context. In *Software Engineering Workshop, 2003. Proceedings. 28th Annual NASA Goddard*, pages 12–20. IEEE.

Rao, A. A. and Reddy, K. N. (2007). Detecting bad smells in object oriented design using design change propagation probability matrix 1.

Rosenberg, L. H. and Hyatt, L. E. (1997). Software Quality Metrics for Object-Oriented Environments. *Crosstalk - the Journal of Defense Software Engineering*, 10.

Ruiz, D. D. A., Becker, K., Novello, T. C., and Cunha, V. S. (2005). A data warehousing environment to monitor metrics in software development processes. In *16th International Workshop on Database and Expert Systems Applications (DEXA 2005), 22-26 August 2005, Copenhagen, Denmark*, pages 936–940. IEEE Computer Society.

Sharble, R. and Cohen, S. (1993). The Object-Oriented Brewery: A Comparison of Two Object-Oriented Development Methods. *Software Engineering Notes*, 18(2):60–73.

Silveira, P. S., Becker, K., and Ruiz, D. D. (2010). Spdw+: a seamless approach for capturing quality metrics in software development environments. *Software Quality Control*, 18(2):227–268.

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., and Wesslén, A. (2012). *Experimentation in software engineering*. Springer.

Yin, R. K. (2011). *Applications of case study research*. Sage.