Aferição de Qualidade de Código-Fonte em Gestão de Contrato Ágil com apoio de um ambiente de *Data Warehousing*: um estudo de caso em uma autarquia da administração pública federal

Guilherme Baufaker Rêgo[1]

[1]Faculdade UnB Gama – Universidade de Brasília

gbre.111@gmail.com

Abstract.

Resumo.

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 1999a]. 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, orgãos da Administração Pública Federal (APF) também têm aderido a utilização de metodologias ágeis nos processos de desenvolvimento de software.

Recentemente, Tribunal de Contas da União publicou o Acórdão no 2314/2013 [BRASIL 2013a] o qual contém um relatório de levantamento elaborado pela Secretaria de Fiscalização de Tecnologia da Informação (SEFTI) cujo objetivo foi conhecer as bases teóricas do processo de desenvolvimento de software com metodologia ágil, além de conhecer experiências práticas de contratação realizadas por instituições públicas federais, como por exemplo, Banco Central do Brasil (BACEN), Instituto do Patrimônio Histórico e Artístico Nacional (IPHAN), Instituto Nacional de Estudos e Pesquisas Educacionais Anísio Texeira (INEP), Tribunal Superior do Trabalho (TST) e Supremo Tribunal Federal (STF).

Nestes orgãos, alguns casos de sucesso advém de processos de desenvolvimento interno e também por meio da transferência da execução tarefas executivas não ligadas com a função fim para iniciativa privada na prática conhecida comumente como teceirização de serviços que é apoiada pela legislação brasileira na Lei 8.666/1993 [BRASIL 1993] que estabelece normas gerais sobre licitações e contratos administrativos; no Decreto-Lei 200/1967 que enunciou sobre a execução indireta de funções não finalísticas visando impedir o crescimento desmesurado da máquina administrativa e na 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.

A Instrução Normativa nº 4 [BRASIL 2010], que é a norma mais recente, enuncia um conjunto e boas práticas e um modelo processo de contratações conforme visto na figura 1.

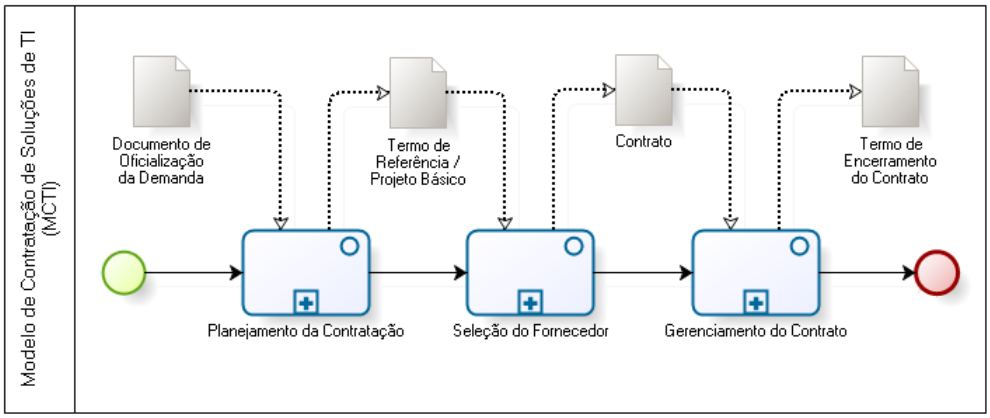


Figura 1. Modelo de Processo de Contratações de Soluções de TI [BRASIL 2013b]

A fase de Gerenciamento do Contrato visa acompanhar e garantir a adequada prestação do serviço e o fornecimento de bens que compõem a solução de TI, sendo que está diposto no art. 25 III, 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 o princípio ágil de valoração de produto sobre documentação abrangente, com objetivo de melhorar a qualidade interna utilizando de práticas do eXtreme Programmming, como por exemplo, desenvolvimento orientado a testes, refatoração e integração contínua do código-fonte [Beck 1999b] e que o processo de gestão do contrato, como já destacado pelo Acórdão no 2314/2013 [BRASIL 2013a], é a fase mais sensível do processo de desenvolvimento de software quando há a utilização de metodologias ágeis, questionou-se como motivação inicial do trabalho:

Como automatizar um processo de medição da qualidade internado produto de forma que o fiscal técnico do contrato possa verificar oportunidades de melhoria no código-fonte do produto?

2. Processo de 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].

Visando a identificação de um conjunto pequeno porém representativo de métricas de código-fonte que possam aferir a qualidade interna do produto de software em desenvolvimento, realizou-se uma revisão da literatura buscando identificar métricas para conceitos amplamente conhecidos como o tamanho do código-fonte que foi um dos primeiros conceitos mensuráveis do software; a complexidade que aumenta à medida que o software é evoluído, sendo que métricas de complexidade estão ligadas as métricas de ta-

manho [Lehman 1980]; e métricas associadas a bom design no paradigma da orientação à objetos que permitiu evolução no desenvolvimento quando comparado ao paradigma procedural, fato que implicou a paradigma de orientação à objetos uma longa permanência na academia quanto na indústria de desenvolvimento de software [Li and Henry 1993]. Estas métricas foram reunidas e são apresentadas na Tabela 1.

Tabela 1. Conjunto de Métricas de Código-Fonte

Métrica	Descrição
ACCM (*Average Cyclomatic Complexity per Method*)	Mede a complexidade do métodos ou funções de um programa. 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* - Média do Número de Parâmetros por Método)	Calcula a média de parâmetros dos métodos da classe o valor mínimo desta métrica é zero e não existe um limite máximo para o seu resultado, mas um número alto de parâmetros pode indicar que um método pode ter mais de uma responsabilidade [Basili and Rombach 1987]
AMLOC (*Average Method Lines of Code* - Média do número de linhas de código por método)	Indica se o código está bem distribuído entre os métodos. Quanto maior, mais pesados são os métodos. É preferível ter muitas operações pequenas e de fácil entendimento que poucas operações grandes e complexas [Meirelles 2013]
CBO (*Coupling Between Objects* - Acoplamento entre Objetos)	Número total de classes dentro de um pacote que dependem de classes externas ao pacote. Quando calculada no nível da classe, essa medida também é conhecida como Fan-out da classe [Chidamber and Kemerer 1994]
NOC (*Number of Children* - Número de Filhos)	Número de subclasses ou classes filhas que herdam da classe analisada [Rosenberg and Hyatt 1997]
LCOM4 (*Lack of Cohesion in Methods* - Falta de Coesão entre os Métodos)	Número de componentes (métodos e atributos) fracamente conectados a uma classe [Hitz and Montazeri 1995].
RFC (*Response For a Class* - Repostas para uma Classe)	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 entre outros que foram desenvolvidos em C, C++ e Java [Meirelles 2013]. Neste trabalho, concluiu-se que há valores muitos frequentes, frequentes, pouco frequentes e não

frequentes para softwares escritos em uma mesma linguagem de programação. Ao considerarmos a linguagem Java e o softwares que obtiveram os melhores resultados, como Eclipse e o OpenJDK 8, obtiveram-se as configurações tal como se mostra na Tabela 2.

Tabela 2. Configurações para os Intervalos das Métricas

Métrica	Intervalo de Frequência	Valores Encontrados
ACCM	Muito Frequente	[de 0 a 2,8]
	Frequente	[de 2,9 a 4,4]
	Pouco Frequente	[de 4,5 a 6,0]
	Não Frequente	[acima de 6]
AMLOC	Muito Frequente	[de 0 a 8,3]
	Frequente	[de 8,4 a 18]
	Pouco Frequente	[de 19 a 34]
	Não Frequente	[acima de 34]
ANPM	Muito Frequente	[de 0 a 1,5]
	Frequente	[de 1,6 a 2,3]
	Pouco Frequente	[de 2,4 a 3,0]
	Não Frequente	[acima de 3]
CBO	Muito Frequente	[de 0 a 3]
	Frequente	[de 4 a 6]
	Pouco Frequente	[de 7 a 9]
	Não Frequente	[acima de 9]
LCOM4	Muito Frequente	[de 0 a 3]
	Frequente	[de 4 a 7]
	Pouco Frequente	[de 8 a 12]
	Não Frequente	[acima de 12]
NPA	Muito Frequente	[0]
	Frequente	[1]
	Pouco Frequente	[de 2 a 3]
	Não Frequente	[acima de 3]
NOC	Muito Frequente	[1]
	Frequente	[2]
	Pouco Frequente	[3]
	Não Frequente	[acima de 3]
RFC	Muito Frequente	[de 0 a 9]
	Frequente	[de 10 a 26]
	Pouco Frequente	[de 27 a 59]
	Não Frequente	[acima de 59]

Em outros trabalhos como os de Marinescu [Marinescu 2005], Moha [Moha et al. 2010] e Rao [Rao and Reddy 2007], foi mostrado é possível detectar pedaços de código-fonte que podem ser melhorados, conhecidos como *code-smell* [Fowler 1999], com a utilização de métricas de código-fonte como forma de detecção. Embasado em alguns destes trabalhos, [Machini et al. 2010] construiu um mapeamento entre as métricas de código-fonte e as técnicas e práticas propostas por [Martin 2008] e [Beck 2007] de forma a construir cenários que indicam a possibilidade de melhoria no código-fonte.

Considerando alguns cenários do trabalho de Machini [Machini et al. 2010] como base, foram construídos mapeamentos chamados de cenários de limpeza de código-fonte, que são mostrados Tabela 3. Cada cenário foi identificado por um nome, características da disposição do código-fonte, recomendações a fim de eliminar o pedaço de código não coesa e a forma de detecção pelos intervalos de frequência do trabalho de Meirelles [Meirelles 2013] que é compatível também com a forma de detecção apresentada nos trabalhos de Marinescu, Moha e Rao.

Tabela 3. 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.	Reduzir a subdvisão da Classe.	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, criou-se Taxa de Aproveitamento de Oportunidade de Melhoria de Código-Fonte como uma indicador objetivo de melhoria da qualidade do código-fonte, sendo que este é descrito conforme a Equação 1.

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

onde Ce_r é o total de cenários de limpeza em uma release e Cl_r é a quantidade classes na mesma release.

Com a identificação em um processo automatizado dos cenários e o cálculo da taxa de aproveitamento de oportnunidades, é possivel fornecer meios de verificação da

qualidade código-fonte aos fiscais técnicos do contratos de desenvolvimento de software, de forma a introduzir a cultura de melhoria contínua da qualidade interna do produto em ciclos futuros de desenvolvimento do software.

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 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] [Rocha 2000]. Em outras palavras, trata-se de um processo para montar e gerenciar dados vindos de várias fontes, com o objetivo de prover uma visão analítica de parte ou do todo do 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.

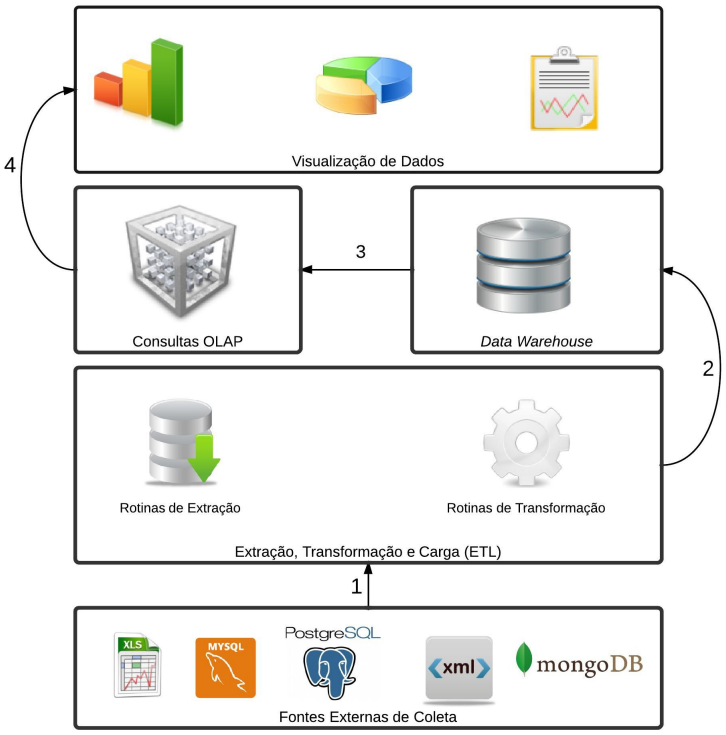


Figura 2. Arquitetura do Ambiente de *Data Warehousing*

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 instrusiva 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 flexíveis 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] modelado sobre um modelo dimensional e o plugin Saiku Analitics[4] para visualização de dados.

4. Estudo de Caso

Visando a validação da solução apresentada na Seção 3, procurou-se aplicar em estudo de caso em umas das instuições citadas pelo Acórdão no 2314/2013 [BRASIL 2013a]. Dentre estas, O Instituto do Patrimônio Histórico e Artístico Nacional que é autarquia federal responsável pela gestão de diversos processos de preservação do patrimônio cultural, permitiu o acesso ao código-fonte Sistema Integrado de Gestão do Conhecimento (SICG), que foi desenvolvido por contrato de terceirização com princípios ágeis, Lean e Kanban, a fim de avaliar a qualidade interna do produto.

O Sistema Integrado de Gestão do Conhecimento (SICG) teve como objetivo automatizar o processo de trabalho decorrente da metodologia de inventário, cadastro, normatização, fiscalização, planejamento e análise e gestão do patrimônio material. Esta solução de software foi construído na linguagem Java com a utilização de*frameworks* conhecidos no mercado, como por exemplom VRaptor e Hibernate, durante 24 *releases* mensais.

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 que foi 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 a avaliação dos Cenários de Limpeza de código-fonte em ambiente de *data warehousing* a fim de verificar a qualidade interna do produto. no qual a fonte de coleta dos dados é o código-Fonte do Sistema Integrado de Gestão do Conhecimento.

Com relação as ameaças citadas por Yin, a validade de construção ao se definir objetivos com evidências diferentes (Taxa de Aproveitamento de Oportunidades e Cenários

[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`

de Limpeza de Código-Fonte). Com relação a validade interna do estudo está garantida quando se consegue avaliar em níveis diferentes ou fontes de informação diferentes (projeto e classe) resultados semelhantes.Quanto a validade externa, destaca-se que a utilização de um estudo de caso não é suficiente para generalizar os resultados dele obtidos, sendo necessário a utilização de estudo em múltiplos casos, a fim de comprovar resultados genéricos [Yin 2011]. Com relação a 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 código-fonte de cada classe e analisadas conforme a Tabela 3. Para cada *release*, foram identificados cenários de limpeza de código-fonte conforme a Figura 3.

	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*

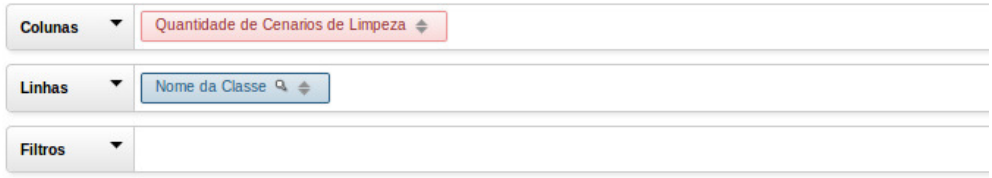
Realizando uma consulta OLAP de consolidação, obteve-se o número total de cenários de limpeza por cada uma das releases de software analisadas tal como se observa na Figura 4.

Nome do Projeto	OS02	OS04	OS05	OS07	OS08	OS09	OS10	OS11	OS12	OS13	OS14	OS16	OS17	OS19	OS20	OS21	OS22	OS23	OS24
Sistema Integrado de Controle e Gestão	27	52	60	94	131	138	173	236	254	277	290	329	347	363	377	389	394	395	397

Figura 4. Total de Cenários de Limpeza de Código-Fonte por Release

Conforme é possível observar nas Figuras 3 e 4, 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**.

Em uma escala de priorização, de qual cenário de limpeza de código-fonte deve ser tratado primeiro, recomenda-se, para o projeto analisado, tratar o cenário de **Complexidade Estrutural**, pois este sozinho chega a responder entre 55% a 68% da quantidade total de cenários identificados. Além da identificação dos cenários de limpeza de código-fonte no projeto como todo, identificou-se, como se mostra na Figura 5, as 10 classes que apresentaram a maior quantidade de cenários de limpeza.



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 de Cénarios de Limpeza

Com o número de classes e os 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

Observa-se, por meio da Figura 6, que o valor da Taxa de Aproveitamento de Oportunidade de Melhoria de Código-Fonte possui uma tendência entre 0,4 a 0,5. Este fato pode indicar duas hipóteses: a primeira de que o projeto cresceu em uma taxa muito maior que a quantidade de cenários de limpeza, indicando assim uma estabilidade na complexidade do projeto; ou que não foram promovidas atividades de melhoria de código-fonte ao longo das 24 releases.

5. Conclusões e Trabalhos Futuros

No ínicio deste trabalho, questionou-se como automatizar um processo de medição da qualidade internado produto de forma que o fiscal técnico do contrato possa verificar

oportunidades de melhoria no código-fonte do produto.

A avaliação de indicadores de código-limpo e acompanhamento da taxa de oportunidades de melhoria de código-fonte são contribuições a monitoramento do código-fonte, pois o acompanhamento destes pode permitir à equipe de desenvolvimento ou até a mesmo a gestores/fiscais de projeto tomarem decisões técnicas mais eficientes no que diz respeito ao código-fonte de um projeto.

Com os resultados apresentados na Seção 4.2, acredita-se que o ambiente de *data warehousing* pode configurar uma boa solução para automatizar a medição de qualidade do código-fonte, pois foi possível ver tanto em nível macro (projeto), quanto em nível micro (classes e módulos) a saúde do projeto com relação a qualidade do código-fonte de forma que os fiscais técnicos podem exigir, desde que esteja protocolo no edital da licitação, níveis de aceitação da qualidade do código-fonte com relação a estes dois indicadores. Outro aspecto que reforça a posição de uma boa solução para o ambiente de *data warehousing*, são os tempos apresentados na Tabela 4, os quais verificamos ser relativamente baixos para necessidade do negócio.

Tabela 4. Tempo de Execução de cada passo automatizado

Tempo	Release 2 (69 classes - 2784 linhas de código)	Release 24 (914 classes - 39.790 linhas de código)
Tempo execução da avaliação dos Cenários de Limpeza de Código-Fonte	20 segundos	1 minuto e 12 segundos
Tempo de Cálculo da Taxa de Aproveitamento de Oportunidades de Melhoria de Código-Fonte	23 segundos	1 minuto e 29 segundos
Tempo Total	43 segundos	2 minutos e 41 segundos

Cabe ressaltar que o IPHAN não realizou análise das métricas de código-fonte e nem pediu a contratada que documentasse as atividades de refatoração de código-fonte ao longo da execução do contrato. Este fato foi o fator limitante a investigação das hipóteses levantadas com relação a taxa de aproveitamento de oportunidades de melhoria do código-fonte.

Como trabalho futuro, ressalta-se a necessidade de melhor investigação das hipóteses com relação a taxa de aproveitamento de oportunidades de melhoria de código-fonte em outros projetos, como por exemplo, os que são ou foram desenvolvidos nos outros orgãos citados pelo Acórdão no 2314/2013 [BRASIL 2013a], desde que seja possível documentar as atividades de refatoração do código-fonte ao longo do desenvolvimento de software.

Referências

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

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

Beck, K. (1999b). *Extreme Programming Explained*. Addison Wesley.

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*. Tribunal de Contas da União, 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.

Li, W. and Henry, S. (1993). Object-oriented metrics that predict maintainability. *Journal of Systems and Software*, 23(2):111–122.

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.

Rocha, A. B. (2000). Guardando históricos de dimensões em data warehouses.

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.