Avaliação de Causalidade entre Métricas de Qualidade Interna e Defeitos

César Couto[1,2], Marco Túlio Valente[2], Roberto da Silva Bigonha[2]

[1]Departamento de Computação, CEFET-MG
[2]Departamento de Ciência da Computação, UFMG

cesar@decom.cefetmg.br, {mtov,bigonha}@dcc.ufmg.br

Abstract. *Despite the interest and the common recommendation for the use of metrics to improve the internal quality of a system, there is still no consensus about the real impacts of metrics in external quality in software systems. Therefore, this paper reports a study carried out to evaluate whether there is a causal relationship between source code metrics and central measure of the external quality of an system, number of bugs. The study aimed to evaluate the cause-effect relationship between CK and OO metrics and the number of bugs of five systems belonging to a public benchmark. As a result, we have concluded that the presence of all classes of a system, there is no evidence of causality between metrics is bugs. However, when we have restricted to classes of a system that had at least one bug in its history, there is evidence of causality for some metrics.*

Resumo. *Apesar do interesse e da recomendação frequente para o uso de métricas para melhorar e monitorar a qualidade de um sistema, ainda não existe clareza sobre os reais impactos de métricas na qualidade externa de sistemas de software. Assim, neste artigo relata-se um estudo desenvolvido com o objetivo de avaliar se existe relação de causalidade entre métricas de código fonte e uma medida central de qualidade externa de um sistema: número de defeitos (bugs). No estudo, procurou-se avaliar a existência de relação de causa-efeito entre métricas CK e OO e o número de defeitos de cinco sistemas pertencentes a um benchmark de domínio público. Como resultado, observou-se que quando são consideradas todas as classes de um sistema, não há indícios da existência de causalidade entre valores reportados para as métricas e número de defeitos. No entanto, quando se restringe a análise às classes de um sistema que tiveram pelo menos um defeito reportado ao longo de sua vida há indícios da existência de causalidade para algumas métricas.*

1 Introdução

Nas últimas décadas, dezenas ou talvez até centenas de métricas foram propostas para se avaliar diversos aspectos da estrutura interna de sistemas de software, incluindo propriedades como acoplamento, coesão, tamanho, separação de interesses, complexidade etc. A justificativa sempre foi centrada no argumento de que "não se pode controlar aquilo que não se consegue medir". Assim, por meio de métricas, engenheiros de software poderiam avaliar e controlar a qualidade interna de um software antes de sua entrega para os clientes finais. Potencialmente, tal controle poderia inclusive viabilizar a descoberta e correção de eventuais problemas, antes que esses se transformassem em defeitos (*bugs*).

Por outro lado, pouco se conhece sobre os reais impactos de métricas na qualidade externa de um software, avaliada por exemplo, por meio do número de defeitos reportados por seus usuários. A grande maioria dos estudos visando a validação empírica de métricas de software baseia-se, por exemplo, em testes de correlação, como o Teste de Spearman [15, 5, 13, 11, 16, 12]. No entanto, reconhecidamente correlação não implica em causalidade, ou seja, quando se encontra uma medida de correlação entre duas variáveis não se pode afirmar que uma delas é a "causa" da outra. Em outras palavras, correlações espúrias constituem um fenômeno bastante comum em qualquer área. Por exemplo, quando se detecta uma eventual correlação positiva entre o grau de acoplamento das classes de um sistema e o número de defeitos reportados por seus usuários, não se pode concluir sumariamente que existe uma relação de causalidade entre essas duas variáveis.

Assim, neste artigo propõe-se a investigação de relações de causalidade – e não apenas de correlação – entre métricas de qualidade interna, mais especificamente métricas de código fonte, e uma medida central da qualidade externa de um sistema: o número de defeitos reportados por seus usuários e desenvolvedores. Mais especificamente, duas características distinguem o presente trabalho de outros estudos já realizados visando a validação empírica de métricas de software:

- Em primeiro lugar, descreve-se no artigo a aplicação no contexto de qualidade de software de um teste de causalidade, originalmente proposto pelo prêmio Nobel de Economia Clive William Granger. Em econometria, tal teste – conhecido como Teste de Granger – é extensivamente usado para avaliar causalidade entre séries temporais de variáveis econômicas e financeiras, tais como produto interno bruto, inflação, preço de *commodities*, salários, dentre outras. Por exemplo, existem estudos que mostram, por meio do Teste de Granger, que o aumento do preço do petróleo causa um aumento na taxa de inflação, pelo menos na economia norte-americana.

- Em segundo lugar, o presente artigo utiliza e estende um *benchmark* recentemente proposto por D'Ambros et al. para avaliar modelos de predição de defeitos de software [5]. Esse *benchmark* disponibiliza um conjunto de séries temporais (medidas em intervalos de bi-semanas) para diversas métricas de cinco sistemas implementados em Java. Para cada classe de cada versão analisada de tais sistemas, são diponibilizados os valores de dezessete métricas de código fonte, incluindo por exemplo métricas de acoplamento, coesão, tamanho etc. No presente artigo, estende-se esse *benchmark* com uma nova série temporal: número de defeitos. O objetivo final é avaliar a existência de causalidade – conforme medida pelo Teste de Granger – entre cada uma das dezessete séries de métricas originalmente disponibilizadas e essa nova série incluída no *benchmark*.

O restante deste artigo está organizado conforme descrito a seguir. Na Seção 2, descreve-se o Teste de Causalidade de Granger. A Seção 3 apresenta o estudo realizado para avaliar a existência de causalidade entre métricas de código fonte e defeitos, bem como são analisados os resultados obtidos. Na Seção 4, relacionam-se as principais lições aprendidas durante o desenvolvimento deste trabalho. A Seção 5 documenta os principais riscos inerentes ao tipo de estudo descrito no artigo. Na Seção 6, são apresentados trabalhos relacionados. Por fim, a Seção 7 descreve as conclusões da pesquisa.

2 Teste de Causalidade de Granger

A dependência de uma variável y (variável dependente) em relação a uma outra variável x (variável explicativa) pode não ser imediata. Com frequência, y reage a x com um lapso de tempo, chamado de defasagem (*lag*). O Teste de Causalidade de Granger é uma técnica estatística para determinar se uma série temporal afeta outra série com uma certa defasagem [6]. Neste trabalho, usa-se esse teste para avaliar se métricas de código fonte podem "causar", no sentido de Granger, defeitos em sistemas de software.

Testar a causalidade entre duas séries temporais x e y, no sentido de Granger, envolve usar um teste estatístico – normalmente o Teste-F – para verificar se valores defasados da série x fornecem informações úteis para predição da série y, na presença de valores defasados de y. Se fornecerem, diz-se que x causa y. A implementação mais comum do Teste de Causalidade de Granger se vale de auto-regressões bivariadas e univariadas. Um modelo auto-regressivo bivariado inclui valores defasados da variável explicativa x acrescidos de valores defasados da variável dependente y. Já um modelo univariado considera apenas valores defasados da variável dependente y.

Para aplicar o Teste de Granger, deve-se inicialmente fixar o valor da defasagem p (parâmetro de entrada do teste). Em seguida, deve-se estimar o seguinte modelo auto-regressivo bivariado pelo método dos mínimos quadrados (ambos termos serão explicados no próximo parágrafo):

$$\begin{aligned} y_t = c_1 + \alpha_1 y_{t-1} + \alpha_2 y_{t-2} + \cdots + \alpha_p y_{t-p} \\ +\beta_1 x_{t-1} + \beta_2 x_{t-2} + \cdots + \beta_p x_{t-p} + u_t \end{aligned} \tag{1}$$

A Equação 1 é um modelo auto-regressivo bivariado porque usa valores defasados de x e y, conforme definido pelo parâmetro p. Além disso, essa equação é estimada pelo método dos mínimos quadrados, o qual procura minimizar a soma dos quadrados dos resíduos da regressão de forma a maximizar o grau de ajuste da equação aos dados observados.

Em seguida, estabelece-se a seguinte hipótese nula a ser rejeitada:

$$H_0 : \beta_1 = \beta_2 = \cdots = \beta_p = 0$$

Essa hipótese assume que valores defasados de x não agregam valor à regressão. A remoção de valores defasados de x tem como objetivo verificar se os valores defasados da série y já fornecem informações suficientes para predição dessa série, de forma que pode-se prescindir da série x.

Para rejeitar a hipótese nula, deve-se primeiro estimar o seguinte modelo auto-regressivo univariado (isto é, uma equação que exclui valores defasados de x) também pelo método dos mínimos quadrados:

$$y_t = c_1 + \gamma_1 y_{t-1} + \gamma_2 y_{t-2} + \cdots + \gamma_p y_{t-p} + e_t \tag{2}$$

Por último, deve-se calcular o somatório dos quadrados dos resíduos de ambas as regressões:

$$RSS_1 = \sum_{t=1}^{T} \hat{u}_t^2 \qquad RSS_0 = \sum_{t=1}^{T} \hat{e}_t^2$$

Se o teste estatístico:

$$S_1 = \frac{(RSS_0 - RSS_1)/p}{RSS_1/(T-2p-1)} \sim F_p, T-2p-1$$

exceder o valor crítico de F com nível de significância de 5% para a distribuição $F(p, T-2p-1)$, o modelo auto-regressivo bivariado é melhor (em função dos resíduos) do que o univariado e, portanto, rejeita-se a hipótese nula. Logo, pode-se concluir que x causa y, nos termos propostos pelo Teste de Granger.

3 Experimentos

Neste trabalho, foram realizados alguns experimentos com o objetivo de investigar a existência de relações de causalidade entre métricas de código fonte e número de defeitos. Para tanto, foi usado o Teste de Causalidade de Granger. Conforme afirmado na Introdução, os trabalhos existentes que analisam relações entre métricas de código fonte e número de defeitos são baseados em testes de correlação. Embora uma análise de correlação seja capaz de mostrar indícios de dependência de duas ou mais variáveis, tais indícios não são suficientes para indicar relações de causa-efeito. Assim, o objetivo dos experimentos descritos nesta seção é analisar se as propriedades sintáticas avaliadas por métricas de código fonte – tais como acoplamento, coesão, tamanho, profundidade da hierarquia de herança, complexidade etc – são realmente capazes de causar defeitos em sistemas de software.

Os experimentos foram realizados usando um *benchmark* público para avaliação de modelos e algoritmos para predição de defeitos, proposto recentemente por D'Ambros et al. [5]. Basicamente, esse *benchmark* disponibiliza séries temporais para dezessete métricas de código fonte, relativas a cinco sistemas de médio porte implementados em Java. As séries disponibilizadas incluem os valores dessas métricas a cada bi-semana (isto é, em intervalos de quinze dias). Logo, trata-se de um *benchmark* bastante adequado para o estudo de causalidade descrito neste trabalho. A Tabela 1 mostra informações detalhadas dos cinco sistemas pertencente ao *benchmark*. Nessa tabela, a coluna Período informa o intervalo de tempo em que as métricas foram coletadas.

Sistema	Período	# Classes	# Versões	# Defeitos	KLOC
Eclipse JDT Core www.eclipse.org/jdt/core/	1/1/2005 - 17/6/2008	1041	91	2510	224
Eclipse PDE UI www.eclipse.org/pde/pde-ui/	1/1/2005 - 11/9/2008	1924	97	2520	146
Equinox framework www.eclipse.org/equinox/	1/1/2005 - 25/6/2008	444	91	612	39
Mylyn www.eclipse.org/mylyn/	17/1/2005 - 17/3/2009	2564	98	4190	156
Apache Lucene lucene.apache.org	1/1/2005 - 8/10/2008	889	99	358	73

Tabela 1. Sistemas do *benchmark* de D'Ambros

Para cada classe em cada versão considerada dos sistemas, ou seja, em intervalos de bi-semanas, o *benchmark* disponibiliza os valores de dezessete métricas de código fonte, sendo seis métricas CK (propostas por Chidamber e Kemerer [3]) e onze outras OO, tais como número de linhas de código, número de métodos públicos, número de métodos herdados, *fan-in*, *fan-out* etc. A Tabela 2 lista as métricas medidas no *benchmark*.

Tipo	Métricas	Descrição
CK	WMC	Métodos ponderados por classe
CK	DIT	Profundidade da árvore de herança
CK	RFC	Resposta de classe
CK	NOC	Número de filhos
CK	CBO	Acoplamento entre classes
CK	LCOM	Ausência de Coesão em métodos
OO	FANIN	Número de outras classes que referenciam a classe
OO	FANOUT	Número de outras classes referenciadas pela classe
OO	NOA	Número de atributos
OO	NOPA	Número de atributos públicos
OO	NOPRA	Número de atributos privados
OO	NOAI	Número de atributos herdados
OO	LOC	Número de linhas de código
OO	NOM	Número de métodos
OO	NOPM	Número de métodos públicos
OO	NOPRM	Número de métodos privados
OO	NOMI	Número de métodos herdados

Tabela 2. Métricas CK e OO medidas no *benchmark* de D'Ambros

Para viabilizar a investigação de causalidade entre métricas e número de defeitos, o *benchmark* de D'Ambros foi estendido com uma nova série: número de defeitos localizados em cada versão das classes dos sistemas considerados no estudo . Para obter essa nova série foi necessário realizar um mapeamento entre defeitos e classes. Para tanto, foram levantados dados sobre os defeitos e sobre o histórico de versões dos cinco sistemas. Para obter informações sobre defeitos, foram usados dados armazenados nos sistemas de gerenciamento de *issues* dos cinco sistemas do *benchmark*, que são o Jira[1] e o Bugzilla[2]. Esses sistemas permitem que mantenedores e usuários reportem defeitos, requisitem melhorias e novas características etc. A coluna # Defeitos da Tabela 1 apresenta o número de defeitos reportados via Bugzilla ou Jira para cada um dos sistemas, no intervalo de tempo considerado no estudo.

Por outro lado, para obter informações sobre o histórico de versões foram consultados dois sistemas de controle de versões: CVS e SVN. Os dados extraídos desses repositórios incluíram os registros (*logs*) de alterações efetuadas ao longo do tempo de vida dos sistemas considerados. Para cada versão armazenada nos repositórios, esses *logs* permitem extrair informações tais como: arquivos alterados, data de criação da versão e motivo de criação da versão, dentre outros. Uma vez obtidas informações sobre defeitos e sobre o histórico de versões dos sistemas foi realizado um mapeamento entre cada defeito reportado e as classes alteradas para sua correção. Esse mapeamento é descrito em detalhes na Seção 3.1.

Em seguida, a avaliação de eventuais relações de causalidade entre métricas de código fonte e defeitos foi realizada no nível de classe, para cada um dos sistemas avaliados (uma vez que as métricas analisadas são também calculadas no nível de classe). Assim, foram consideradas no total 6862 classes e mais de 600 mil linhas de código. Mais especificamente, para cada classe foi avaliada a existência de causalidade entre dezessete séries temporais de métricas e a série temporal número de defeitos, usando o Teste de Granger.

[1]http://www.atlassian.com/software/jira
[2]http://www.bugzilla.org

3.1 Coleta de Dados

Para obtenção da série temporal número de defeitos para cada classe dos cinco sistemas do *benchmark*, as seguintes atividades foram realizadas:

Filtragem de Defeitos: Usando-se informações disponibilizadas no sistema Jira – usado pelo sistema Lucene – e no sistema Bugzilla – usado pelos sistemas JDT, PDE, Equinox e Mylyn – foram coletados registros de defeitos que atendem às seguintes condições:

- Foram reportados no período de tempo estabelecido pelo *benchmark* (por exemplo o período de coleta para o sistema Lucene foi entre 01/01/2005 a 08/10/2008, conforme descrito na Tabela 1).

- Denotam efetivamente defeitos (ou seja, foram desconsiderados registros que denotam solicitações de melhorias e novas características). O Jira possui um campo que indica a classificação da solicitação do usuário, logo foram considerados apenas solicitações do tipo defeito. Já o sistema Bugzilla funciona exclusivamente para solicitações que visam a correção de defeitos.

- Possuem situação de corrigido (ou seja, foram desconsiderados registros de defeitos ainda em aberto, defeitos duplicados, inválidos e incompletos). Por meio do campo *fixed* dos sistemas Jira e Bugzilla, conseguiu-se selecionar apenas os registros defeitos que foram realmente corrigidos.

Em seguida, foi implementado um *parser* XML para ler os registros de defeitos e coletar a data na qual cada defeito foi reportado e sua identificação (BUG-ID). A data de criação do defeito é essencial para extração da série temporal número de defeitos. Já o BUG-ID é útil para realizar o mapeamento entre um defeito e e as classes alteradas para sua correção, conforme descrito a seguir.

Filtragem de Versões: Os sistemas de controle de versões CVS – usados pelos sistemas JDT, PDE, Equinox e Mylyn – e SVN – usado pelo sistema Lucene – foram acessados. Para cada um dos cinco sistemas foi recuperado os *logs* de cada versão, começando da data inicial definada pelo *benchmark* até a data de elaboração do experimento (por exemplo, para o sistema Lucene o período de coleta dos *logs* foi entre 01/01/2005 a 11/02/2011). A data final definida pelo *benchmark* não foi escolhida, pois as modificações para correção dos defeitos coniderados no experimento podem ter sido feitas após a mesma. Em seguida, foi implementado um segundo *parser* XML para ler os *logs* e extrair as descrições textuais dos *commits* e os arquivos alterados pelos mesmos. Essas informações são úteis no mapeamento de defeitos para classes.

Mapeamento de Defeitos para Classes: Inicialmente, nesta etapa os BUG-IDs levantados no primeiro passo (Filtragem de Defeitos) foram mapeados para seus respectivos *commits*. Em seguida, os arquivos alterados em tais *commits* foram usados para identificar as classes alteradas para correção dos defeitos associados aos BUG-IDs considerados. Para isso, foram realizados os seguintes procedimentos:

1. Por meio de uma pesquisa textual, procurou-se cada BUG-ID levantado no primeiro passo (Filtragem de Defeitos) no texto livre de descrição dos *commits*.

A grande maioria das descrições dos *commits* para correção de defeitos inclui uma referência para o BUG-ID vindo do sistema de gerenciamento de *issues* (por exemplo, "*bugfix for LUCENE-455*"). Assim, tais referências foram usadas para mapear BUG-IDs para seus respectivos *commits*. Este mesmo procedimento é usado em outros trabalhos, como por exemplo no trabalho de D'Ambros et al. [5].

2. Todas as classes dos sistemas pertencentes ao *benchmark* são classes públicas, como afirmado pelos criadores do *benchmark*. Por outro lado, em Java, deve haver exatamente uma classe pública em um dado arquivo fonte e seu nome deve ser precisamente o nome do arquivo. Sendo assim, os nomes dos arquivos envolvidos nos *commits* em cuja descrição textual encontrou-se o BUG-ID de um determinado defeito indicam as classes corrigidas para remoção desse defeito.

Em resumo, por meio dos procedimentos mencionados, conseguiu-se identificar as classes modificadas para correção de cada defeito considerado no experimento. A Figura 1 ilustra o processo descrito anteriormente.

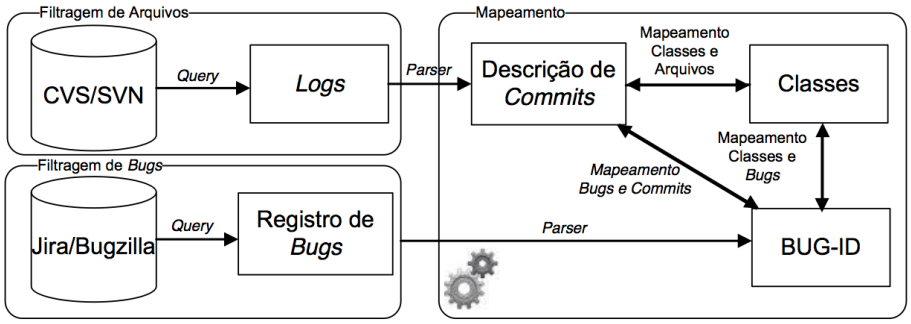

Figura 1. Procedimentos de coletada de dados

Criação da Série Temporal Número de Defeitos: Uma vez coletados os defeitos e as classes modificadas para correção dos mesmos, o passo seguinte foi criar uma série temporal de número de defeitos em intervalos de bi-semanas. Para tanto, os seguintes procedimentos foram realizados:

1. A data de criação de um defeito foi usada para definir a bi-semana do *benchmark* a qual ele se refere. Assim, cada defeito foi distribuído ao longo das bi-semanas.

2. Uma vez realizada a distribuição dos defeitos em bi-semanas, o próximo passo foi calcular o número de defeitos novos que cada classe apresentou em uma determinada bi-semana t. Esse número foi acrescentado ao número defeitos dessa classe até a bi-semana $t-1$. Em outras palavras, na série temporal de defeitos, o número de defeitos de uma classe na bi-semana t é igual ao número de defeito novos que essa classe apresentou na semana t mais o número de defeitos que a classe já havia apresentado até a semana $t-1$.

A série temporal número de defeitos gerada foi usada no cálculo do Teste de Causalidade de Granger, descrito na na próxima seção.

3.2 Resultados do Teste de Causalidade de Granger

Para avaliar se existe relação de causa-efeito entre métricas de código fonte e defeitos foi usado o Teste de Causalidade de Granger. Os testes foram calculados por meio da ferramenta estatística R com nível de significância de pelos menos 95% ($\alpha = 0.05$) e tamanho de defasagem (*lag*) variando de 1 a 5. Considerou-se que o Teste de Granger indica causalidade quando a variável *p-value* resultante da aplicação do teste de hipótese F é menor ou igual a α, ou seja, quando *p-value* ≤ 0.05. Além disso, no experimento foram consideradas apenas classes com tempo de vida maior ou igual a vinte bi-semanas. Classes com séries temporais com poucos valores foram descartadas, pois podem dar origem a modelos auto-regressivos imprecisos.

O teste de causalidade foi processado analisando para cada classe a série temporal de cada uma das dezessete métricas listadas na Tabela 2 e a série temporal de número de defeitos. Para ilustrar a aplicação do teste – e seu princípio básico de funcionamento – a Figura 2 apresenta um gráfico onde encontra-se plotada a série temporal da métrica WMC (Métodos Ponderados por Classe) e a série número de defeitos para a classe `org.apache.lucene.index.IndexWriter` do sistema Lucene. No gráfico mostrado, o eixo x representa as 99 bi-semanas consideradas no experimento e o eixo y representa os valores da métrica WMC e número de defeitos. Como pode ser observado, as linhas do gráfico mostram que quando o WMC aumenta o número de defeitos também aumenta. Nesta classe especificamente, o Teste de Granger demonstrou a existência de uma relação de causa-efeito entre a métrica WMC e a série número de defeitos, pois o valor *p-value* encontrado foi igual a 0.04 e, portanto, menor que o nível de significância adotado no trabalho (α = 0.05).

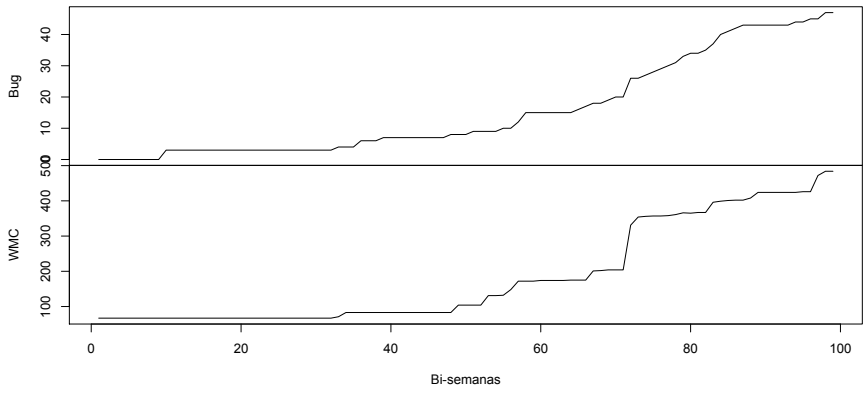

Figura 2. Séries WMC e número de defeitos da classe `org.apache.lucene.index.IndexWriter`

Por outro lado, a Figura 3 ilustra que a mesma métrica WMC não causa defeitos – nos termos do Teste de Granger – na classe `org.apache.lucene.index.FieldInfos`. Conforme pode ser observado, o número de defeitos permanece estável mesmo quando ocorre uma variação no valor de WMC.

Como o número de classes consideradas no experimento é bastante alto, optou-se por apresentar os resultados do Teste de Granger sumarizados em nível de sistema. Para

isso, a seguinte estratégia foi utilizada: calculou-se a percentagem de classes em que o Teste de Causalidade de Granger apresentou resultados positivos para cada um dos cinco sistemas. Essas percentagens são apresentados na Tabela 3.

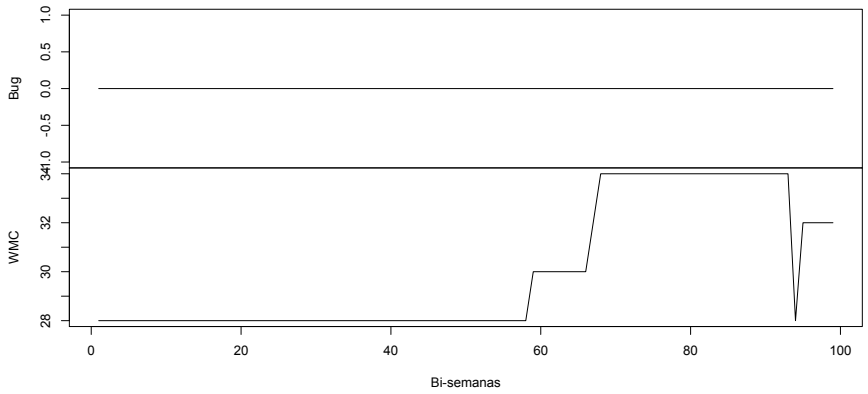

Figura 3. Séries WMC e número de defeitos da classe `org.apache.lucene.index.FieldInfos`

Métrica	JDT	PDE	Equinox	Mylyn	Lucene
CBO	0.18	0.20	0.12	0.17	0.09
DIT	0.04	0.03	0.01	0.03	0.02
LCOM	0.13	0.14	0.07	0.09	0.08
NOC	0.01	0.02	0.01	0.03	0.02
RFC	0.20	0.20	0.10	0.16	0.10
WMC	0.18	0.18	0.10	0.15	0.10
FANIN	0.09	0.11	0.07	0.12	0.07
FANOUT	0.15	0.17	0.09	0.14	0.07
NOA	0.09	0.11	0.06	0.08	0.06
NOPA	0.06	0.01	0.02	0.01	0.02
NOPRA	0.04	0.09	0.04	0.07	0.04
NOAI	0.11	0.07	0.02	0.03	0.03
LOC	0.19	0.20	0.11	0.18	0.10
NOM	0.13	0.14	0.07	0.09	0.09
NOPM	0.10	0.11	0.06	0.09	0.07
NOPRM	0.05	0.08	0.07	0.05	0.03
NOMI	0.22	0.20	0.11	0.18	0.11
Média CK	0.123	0.128	0.068	0.105	0.068
Média OO	0.111	0.117	0.065	0.094	0.062
Média Total	0.115	0.121	0.066	0.098	0.064

Tabela 3. Porcentagem de classes com relação de causalidade entre as métricas listadas e número de defeitos

Análise dos Resultados: Conforme pode ser observado na Tabela 3, os resultados do Teste de Causalidade de Granger mostram que não há indícios da existência de relação de causalidade entre métricas de código fonte e defeitos. As percentagens reportadas nesta tabela são sempre menores que 22%, isto é , no melhor caso, detectou-se que em 22% das classes do sistema JDT existe uma relação de causalidade entre as seguintes séries temporais: valores da métrica NOMI (Número de Métodos Herdados) e número de defeitos. As seguintes observações podem ser também realizadas sobre os resultados apresentados na Tabela 3:

- Para os sistemas Equinox e Lucene, as percentagens de causalidade foram ainda menores, chegando ao máximo de 12%. Considerando o fato de que os dois sistemas juntos totalizam 1333 classes, ou seja, cerca de 20% do total de classes dos sistemas considerados no estudo, pode-se fortalecer os indícios de que as relações de causa-efeito não se estabelecem.

- As percentagens das métricas DIT (Profundidade da Arvore de Herança) e NOC (Número de Filhos) mostram com precisão que não existe relação de causa-efeito entre herança e defeitos. As porcentagens reportadas são sempre menores que 4%. Além disso, as métricas relacionadas com atributos (NOA, NOPA, NOPRA e NOAI) possuem baixos valores, mostrando que classes que são grandes armazenadoras de dados não são mais sujeita a defeitos. Por fim, os baixos valores das percentagens de causalidade para a métrica relacionada com métodos privados (NOPRM) também sugerem que tais classes não são mais susceptíveis a defeitos.

- Por último, não existe uma diferença significativa entre as médias das percentagens das métricas CK e OO. Conforme pode ser observado, as diferenças entre tais categorias de métricas são sempre inferiores a 1,5%, na média. Em outras palavras, não se pode concluir que métricas CK impactam mais no número de defeitos do que métricas OO ou vice-versa.

Para melhor entender as razões do pequeno percentual de classes com relação de causalidade foi investigada a seguir a distribuição dos defeitos levantados no estudo pelas classes dos cinco sistemas analisados. Descobriu-se que essa distribuição segue a Lei de Pareto, ou seja, cerca de 80% dos defeitos estão localizados em 20% das classes. Por exemplo, para o sistema Lucene, em uma amostra de 726 classes, somente 170 possuíam pelo menos um defeito (ou 23% das classes analisadas). Além disso, quando a série temporal com o número de defeitos de uma classe possui sempre valores nulos, ou seja, quando uma classe nunca teve um defeito, o Teste de Granger não aponta causalidade. Assim, devido ao fato de 80% dos defeitos serem encontrados em 20% das classes analisadas e devido ao fato de o Teste de Granger não indicar causalidade em classes sem nenhum defeito no período de tempo analisado, as percentagens apresentadas na Tabela 3 foram tão reduzidas.

Logo, pode-se formular a seguinte conclusão: em cerca de 80% das classes de um sistema o monitoramento de métricas não é importante, pelo menos para fins de detecção de defeitos. Independentemente dos valores das dezessete métricas analisadas nesse estudo, essas classes não apresentaram defeitos no período de tempo analisado. Ou seja, em tais classes certas medidas, como tamanho, coesão e acoplamento, têm pouco impacto no número de defeitos reportados pelos usuários e desenvolvedores dos sistemas analisados. Para ser claro, mesmo que tais classes apresentem, por exemplo, elevados níveis de acoplamento e baixos níveis de coesão, elas conseguem ficar imunes a defeitos por um longo período de tempo.

Por outro lado, surge então uma nova pergunta: qual o impacto nos percentuais de causalidade previamente reportados caso sejam consideradas apenas classes com algum defeito ao longo período de tempo analisado no estudo? Para responder a essa segunda pergunta, aplicou-se novamente o Teste de Granger, porém considerando somente as classes com ocorrência de defeitos, isto é, classes que tiveram pelo menos um defeito

no período considerado no estudo. A Tabela 4 apresenta os resultados obtidos para tais classes.

Métrica	JDT	PDE	Equinox	Mylyn	Lucene
CBO	0.30	0.27	0.15	0.27	0.30
DIT	0.13	0.05	0.01	0.05	0.09
LCOM	0.27	0.21	0.10	0.14	0.32
NOC	0.08	0.03	0.02	0.04	0.09
RFC	0.32	0.28	0.12	0.24	0.35
WMC	0.30	0.25	0.12	0.23	0.38
FANIN	0.22	0.16	0.10	0.18	0.26
FANOUT	0.29	0.24	0.11	0.21	0.26
NOA	0.25	0.17	0.07	0.12	0.27
NOPA	0.20	0.01	0.04	0.02	0.12
NOPRA	0.16	0.14	0.06	0.10	0.20
NOAI	0.20	0.11	0.04	0.04	0.16
LOC	0.30	0.27	0.13	0.28	0.36
NOM	0.28	0.21	0.10	0.14	0.35
NOPM	0.27	0.17	0.08	0.13	0.29
NOPRM	0.18	0.13	0.10	0.08	0.15
NOMI	0.29	0.28	0.13	0.28	0.36
Média CK	0.233	0.181	0.086	0.161	0.255
Média OO	0.240	0.171	0.087	0.143	0.252
Média Total	0.237	0.175	0.087	0.15	0.253

Tabela 4. Porcentagem de classes com relação de causalidade entre as métricas listadas e número de defeitos (considerando apenas classes com pelo menos um defeito)

Análise dos Resultados: Quando se restringe a análise a somente classes que tiveram pelo menos um defeito, os resultados do Teste de Causalidade de Granger apresentam percentagens maiores, conforme pode ser observado na Tabela 4. As percentagens reportadas nessa tabela atingiram valores próximos a 40% para alguns sistemas. Além de tal aumento, deve-se ainda destacar as seguintes observações:

- No sistema Equinox, quando se restringe o teste de causalidade de Granger a classes que tiveram pelo menos um defeito, não ocorre um aumento significativo nas porcentagens, já que a média total dos percentuais passa de 0.066 para 0.087.

- Não existe uma diferença significativa nas porcentagens das métricas DIT e NOC quando se restringe o teste a classes com defeitos. Conforme pode ser observado, as percentagens dessas duas variáveis continuam baixas. Assim, reforça-se o argumento de que o uso de herança não é um fator que impacta no número de defeitos.

4 Lições Aprendidas

Esta seção discute as principais lições aprendidas com o estudo realizado. Primeiro, o estudo de causalidade entre métricas de código fonte e número de defeitos revelou que não há indícios da existência de relação de causalidade entre essas duas variáveis. Ficou claro para os cinco sistemas analisados que métricas de código fonte não afetam diretamente o número de defeitos, pois o percentual de classes nas quais o Teste de Granger apresentou causalidade foi bastante reduzido (sempre inferior a 22%). Assim, quando se considera um sistema completo, ou seja, quando se considera todas as classes de um sistema, pode-se concluir que métricas não são tão importantes como instrumentos para predição de defeitos.

Por outro lado, quando se restringe a análise a somente aquelas classes que tiveram pelos menos um defeito em seu tempo de vida, o estudo de causalidade revelou que há indícios da existência de relação de causa-efeito para a maioria das métricas de código fonte em quatro dos cincos sistemas envolvidos (com exceção do sistema Equinox). Particularmente, os resultados obtidos por meio do Teste de de Granger permitem afirmar o seguinte: uma variação negativa nos valores das métricas de código fonte de um sistema implica em um aumento no número de defeitos. Assim, conclui-se que métricas de código fonte causam defeitos em classes que já tiveram pelo menos um defeito.

Com base nesses resultados, gerentes de qualidade de software devem então monitorar o número de defeitos nas classes de seus sistemas. Em classes com mais defeitos depois de um certo tempo de vida, esses gerentes devem então começar a monitorar também métricas de código fontes, pois variações negativas nos valores dessas métricas são indicadores de uma maior probabilidade de defeitos futuros, conforme revelado pelo Teste de Granger. Em outras palavras, o controle da qualidade interna de tais classes – medido por meio de métricas CK e OO – pode ajudar a evitar o surgimento de novos defeitos, colaborando assim para um aumento da qualidade externa de tais sistemas.

5 Riscos à Validade do Estudo

Nesta seção, os resultados do estudo de causalidade descrito no artigo são avaliados segundo sua validade interna, externa e de construção [14]:

Validade Externa: Essa forma de validade se refere ao grau de aplicabilidade das conclusões de um estudo a uma população mais ampla. O estudo da relação de causa-efeito envolveu cinco sistemas do *benchmark* de D'Ambros, sendo quatro deles pertencentes à Fundação Eclipse e um pertencente à Fundação Apache, totalizando 6862 classes e mais de 600 mil linhas de código. Essa amostra constitui um ponto forte do estudo realizado, pois considera um bom número de classes, um bom número de sistemas, todos eles relevantes, de relativa complexidade e com uma base consolidada de usuários finais. Mesmo com todos esses ponto fortes, as conclusões reportadas no artigo não podem ser generalizadas para qualquer sistema. O motivo é que os resultados apresentados no artigo dependem de uma série de variáveis que fazem parte do processo de desenvolvimento de software, como por exemplo, maturidade da organização, experiência dos desenvolvedores, complexidade do domínio da aplicação etc. No entanto, a metodologia empregada no estudo (incluindo o mapeamento de defeitos para classes e o Teste de de Granger) pode ser replicada em qualquer organização interessada em monitorar a qualidade externa de seus sistemas por meio de métricas de qualidade interna.

Outro ponto forte que vale a pena destacar é que os defeitos analisados foram coletados de diferentes sistemas de gerenciamento de *issues* (Jira e Bugzilla) e que os *logs* foram coletados de diferentes sistemas de controle de versão (CVS e SVN), o que minimiza eventuais riscos derivados da adoção de uma tecnologia ou ferramenta específica. Além disso, os sistemas do *benchmark* são desenvolvidos por equipes de desenvolvimento independentes e por duas comunidades não relacionados (Eclipse e Apache).

Validade Interna: Essa forma de validade avalia se as conclusões obtidas não são resultantes de fatores que não foram controlados ou medidos. Um primeiro risco diz respeito à maneira como os defeitos foram mapeados para *commits* e subsequentemente para

suas respectivas classes. Na verdade, todos os defeitos que não tem uma referência na descrição textual dos *commits* foram desconsiderados. No entanto, a porcentagem desses defeitos não foi grande, pois dos 10190 registros de defeitos coletados, 7657 continham uma referência na descrição do *commit*, ou seja, mais de 75% dos registro de defeitos foram considerados válidos.

Uma outra ameaça diz respeito à validade interna do estudo diz respeito à qualidade dos dados disponibilizados no repositório Bugzilla. Por exemplo, Antoniol et al. mostraram que uma fração considerável das *issues* classificados como defeitos nesse repositório não são de fato defeitos, ou seja, essas *issues* não estão relacionados com atividades de manutenção corretiva [1]. Para verificar se esse comportamento estava presente nos sistemas analisados no presente artigo, D'ambros et al. [5] fizeram uma avaliação manual dos registros de defeitos cadastrados para o JDT e chegaram à conclusão de que 97% desses registros são efetivamente manutenções corretivas. Consequentemente, o impacto dessa ameaça para o experimento reportado é limitado.

Validade de Construção: Essa forma de validade procura avaliar se as conclusões obtidas não são resultantes de uma condução incorreta do experimento (por exemplo, devido a dados incorretos que foram gerados). Os registros de defeitos foram corretamente coletados, pois o sistema Jira já possui um campo que indica a classificação da solicitação do usuário (esse campo pode ter os seguintes valores: defeito, *improvement*, *new feature*, *task*, *test* ou *wish*; no estudo, foram considerados apenas solicitações do tipo defeito). O sistema Bugzilla funciona exclusivamente para solicitações do tipo defeito. Além disso, por meio do campo *fixed* dos sistemas Jira e Bugzilla, conseguiu-se selecionar apenas os defeitos que foram realmente corrigidos.

6 Trabalhos Relacionados

D'Ambros et al. forneceram um *benchmark* público para predição de defeitos [5]. Usando este *benchmark*, eles avaliaram um conjunto representativo de abordagens para predição de defeitos reportados na literatura, incluindo abordagens baseadas nas seguintes métricas: (i) métricas de alterações definidas por Moser et al. [10]; (ii) *bugfixes* propostas por Kim et al. [9]; (iii) métricas CK e OO cujo valores são usados neste artigo; e (iv) entropia de alterações proposta por Hassan [7]. Além disso, os autores propuseram duas novas métricas chamadas *churn* de código fonte e entropia de código fonte. Um estudo sobre a correlação entre estas métricas e defeitos foi realizado usando o teste de Spearman. Os resultados do teste de Spearman mostraram que o conjunto CK e OO como um todo não possuem uma correlação significativa com defeitos. Neste artigo, quando se considerou todas as classes de um sistema (abordagem semelhante a que os autores usaram), os testes de Granger mostraram que CK e OO não impactam no número de defeitos.

Subramanyam e Krishnan investigaram a relação entre defeitos e métricas CK [15]. Para tanto, eles analisaram um sistema de comércio eletrônico e concluiram que WMC, DIT e CBO*DIT estão correlacionadas com número de defeitos. Entretanto, como a correlação não implica em causalidade não se pode concluir que estas métricas impactam diretamente no número de defeitos. Isto pode ser comprovado pelos resultados deste artigo em que a métrica DIT não apresentou uma relação de causa-efeito com defeitos.

Nagappan et al. conduziram um estudo em cinco componentes do sistema opera-

cional Windows com o objetivo de investigar a relação entre métricas de complexidade (tais como FANIN, FANOUT, *Coupling* etc) e defeitos [13]. Posteriormente, o estudo foi replicado em um grande sistema ERP (SAP R3) [8]. No estudo, eles observaram uma significante correlação entre métricas de complexidade e defeitos. Este estudo possui grande validade externa, pois envolveu cinco componentes, totalizando mais de um milhão de linhas de código fonte. O estudo apresentado neste artigo poderia ser aproveitado pelos autores para avaliar se a correlação encontrada implica em causalidade.

Canfora et al. apresentam uma comparação empírica entre duas técnicas usadas para a identificação de *change coupling* em quatro sistemas de código aberto escritos em Java e C (Mylyn, FreeBSD (i386), Rhino e Squid) [2]. As técnicas usadas na comparação são o teste de causalidade de Granger e regras de associação (*association rules*). Eles mostram que, embora as regras de associação fornecem resultados mais precisos, o teste de causalidade de Granger é capaz de alcançar uma melhor valor para o Teste-F e de recomendar um número maior de *change couplings* verdadeiros. Este trabalho mostra que o teste de causalidade de Granger pode ser aplicado em outras áreas além da área econômica (por exemplo, engenharia de software).

D'Ambros et al. conduziram uma análise sobre a relação entre falhas de projeto de software (por exemplo, *brain method*, *feature envy*, *shotgun surgery* etc) e defeitos [4]. Com este propósito, eles mostraram que falhas de projeto correlacionam com defeitos. No entanto, uma pergunta pode ser formulada: *falhas de projeto causam defeitos ou apenas estão correlacionados?* Não se tem conheci mento de um trabalho que avalia a relação de causa-efeito entre falhas de projeto e número de defeitos.

7 Conclusões

Neste artigo, procurou-se esclarecer se existem relações de causalidade – e não apenas de correlação – entre métricas de qualidade interna e uma medida inequívoca da qualidade externa de um sistema, número de defeitos. Mais especificamente, aplicou-se o teste de causalidade de Granger em classes de cinco sistemas disponíveis em um *benchmark* de domínio público com objetivo de avaliar se um conjunto de métricas CK e OO impactam diretamente no número de defeitos. O teste de Granger é extensivamente usado para avaliar causalidade entre séries temporais.

O estudo de causalidade realizado apresenta duas contribuições principais. Primeiro, quando se realizou o teste de causalidade para todas as classes dos cinco sistemas do *benchmark*, os resultados mostraram que há indícios da não existência de relação de causa-efeito entre métricas de código fonte e defeitos. Por outro lado, quando se restringe a análise a somente aquelas classes que tiveram pelos menos um defeito em seu histórico, há indícios da existência de relação de causa-efeito para a maioria das métricas de código fonte em quatro dos cincos sistemas envolvidos.

Como trabalho futuro, pretende-se ampliar o estudo realizado, possivelmente acrescentando novas métricas e/ou sistemas. Pretende-se também investigar outras técnicas estatísticas de inferência de causalidade, além do Teste de Causalidade de Granger.

Agradecimentos: Este trabalho foi apoiado pela FAPEMIG e CNPq. Gostaríamos de fazer um agradecimento todo especial a Marco D'Ambros pela louvável disponibilização pública das séries temporais das métricas usadas no artigo.

Referências

[1] Giuliano Antoniol, Kamel Ayari, Massimiliano Di Penta, Foutse Khomh, and Yann-Gaël Guéhéneuc. Is it a bug or an enhancement? In *18th Conference of the Center for Advanced Studies on Collaborative Research: Meeting of Minds (CASCON)*, pages 304–318, 2008.

[2] Gerardo Canfora, Michele Ceccarelli, Massimiliano Di Penta, and Luigi Cerulo. Using multivariate time series and association rules to detect logical change coupling: an empirical study. In *26th International Conference on Software Maintenance (ICSM)*, pages 1–10, 2010.

[3] S. R. Chidamber and C. F. Kemerer. A metrics suite for object oriented design. *IEEE Transactions on Software Engineering*, 20(6):476–493, 1994.

[4] Marco D'Ambros, Alberto Bacchelli, and Michele Lanza. On the impact of design flaws on software defects. In *10th International Conference on Quality Software (QSIC)*, pages 23–31, 2010.

[5] Marco D'Ambros, Michele Lanza, and Romain Robbes. An extensive comparison of bug prediction approaches. In *7th Working Conference on Mining Software Repositories (MSR)*, pages 31–41, 2010.

[6] C. W. J. Granger. Investigating causal relations by econometric models and cross-spectral methods. *Econometrica*, 37(3):424–438, 1969.

[7] Ahmed E. Hassan. Predicting faults using the complexity of code changes. In *31st International Conference on Software Engineering (ICSE)*, pages 78–88, 2009.

[8] Tilman Holschuh, Markus Pauser, Kim Herzig, Thomas Zimmermann, Rahul Premraj, and Andreas Zeller. Predicting defects in sap java code: An experience report. In *31st International Conference on Software Engineering (ICSE)*, pages 172–181, 2009.

[9] Sunghun Kim, Thomas Zimmermann, E. James Whitehead Jr., and Andreas Zeller. Predicting faults from cached history. In *29th International Conference on Software Engineering (ICSE)*, pages 489–498, 2007.

[10] Raimund Moser, Witold Pedrycz, and Giancarlo Succi. A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction. In *30th International Conference on Software Engineering (ICSE)*, pages 181–190, 2008.

[11] Nachiappan Nagappan and Thomas Ball. Static analysis tools as early indicators of pre-release defect density. In *27th International Conference on Software Engineering (ICSE)*, pages 580–586, 2005.

[12] Nachiappan Nagappan and Thomas Ball. Use of relative code churn measures to predict system defect density. In *27th International Conference on Software Engineering (ICSE)*, pages 284–292, 2005.

[13] Nachiappan Nagappan, Thomas Ball, and Andreas Zeller. Mining metrics to predict component failures. In *28th International Conference on Software Engineering (ICSE)*, pages 452–461, 2006.

[14] Dewayne E. Perry, Adam A. Porter, and Lawrence G. Votta. A primer on empirical studies (tutorial). In *Tutorial presented at 19th International Conference on Software Engineering (ICSE)*, pages 657–658, 1997.

[15] Ramanath Subramanyam and M. S. Krishnan. Empirical analysis of CK metrics for object-oriented design complexity: Implications for software defects. *IEEE Transaction on Software Engineering*, 29(4):297–310, 2003.

[16] Thomas Zimmermann, Nachiappan Nagappan, and Andreas Zeller. *Predicting Bugs from History*, chapter 4, pages 69–88. Springer, 2008.