Construção de um Dataset para Análise da Evolução dos *bugs* de Sistemas

Pedro A. Pires[1]

[1]Departamento de Ciência da Computação – Universidade Federal de Minas Gerais (UFMG)
Belo Horizonte – MG – Brazil

ppires@dcc.ufmg.br

Abstract. *This paper documents a time series dataset on the evolution of bugs and seventeen object-oriented metrics extracted from nine open- source systems. This dataset is an extension to the dataset created by Marco Dambros to assess bug prediction techniques.*

Resumo. *Este artigo documenta um dataset de séries temporais sobre a evolução de bugs e de dezessete métricas orientadas a objetos, extraídas de nove sistemas open-source. Este conjunto de dados é uma extensão para o conjunto de dados criado por Marco Dambros para avaliar técnicas de predição de bugs.*

1. Introdução

Estudar a evolução e entender a estrutura de sistemas legados são atividades conectadas pois:

- examinar a estrutura de subsistemas nos permite entender melhor a evolução do sistema como um todo.
- as informações das entidades de software podem revelar relacionamentos escondidos entre elas.
- analisar várias versões de um sistema melhora o nosso entendimento sobre ele.

Baseado nessas informações, este artigo propõe um dataset para análise da evolução dos *bugs* de sistemas, além de dezessete métricas orientadas a objetos. Este dataset é uma extensão do dataset criado por [D'Ambros et al. 2010], aumentando não só o número de sistemas, mas também o número de versões analisadas de cada sistema.

2. Solução Proposta

O dataset construído é composto pelas informações sobre *bugs* e código fonte dos 9 sistemas listados na Tabela 1. É fornecido, para cada sistema: os *bugs* extraídos dos sistemas de issue tracking e os logs de commits extraídos dos repositórios de código fonte; Versões bissemanais dos sistemas representados de acordo com um modelo orientado a objetos; Valores da quantidade de *bugs* em cada classe, para cada versão do sistema; Valores de métricas de código fonte para cada classe, para cada versão dos sistemas. Todos os sistemas utilizam a linguagem Java, e as informações foram extraídas pelo mesmo parser, para evitar problemas devido a diferenças de comportamento entre os parser, um problema conhecido em ferramentas de engenharia reversa [R. Kollmann and Stroulia 2002].

Table 1. Sistemas no dataset

Sistema	Período	# Versões
Eclipse JDT Core	2001-07-01 – 2008-06-14	183
Eclipse PDE UI	2001-05-24 - 2008-04-03	180
Equinox Framework	2003-11-25 – 2010-10-05	180
Lucene	2002-06-22 – 2009-05-02	180
Hibernate	2007-06-13 – 2012-10-10	140
TV-Browser	2003-04-23 – 2011-08-27	228
Hadoop MapReduce	2008-08-22 – 2012-08-03	77
Geronimo	2003-08-20 – 2012-08-08	235
Axis 2	2004-09-02 – 2011-08-25	182

2.1. Processo de Extração dos Dados

Para que fosse possível construir as séries temporais dos dados foram necessárias as seguintes informações: (1) logs de *commits* dos repositórios de código fonte, para obter informações sobre arquivos modificados na resolução de um *bug*; (2) informações sobre *bugs* dos *issue trackers*, para saber quais foram os *bugs* que ocorreram nos sistemas no período analisado; e (3) código fonte dos sistemas, para computar as métricas.

Para descobrir quais arquivos foram modificados durante a resolução de um bug, foram extraídos os logs dos commits de cada sistema em seus respectivos repositórios de código fonte. De cada commit foram extraídos os arquivos modificados, e a mensagem deixada pelo desenvolvedor no momento da transação. Todos os sistemas utilizam como repositório os sistemas SVN ou Git, e em cada um deles é uma tarefa trivial extrair esses dados.

As informações sobre os *bugs* dos sistemas foram buscadas nos sistemas de issue tracking utilizados. Todos os sistemas do dataset utilizam o Jira ou o Bugzilla, e nenhum desses sistemas oferece um web service para download das informações. Devido a isso, as informações tiveram de ser baixadas manualmente de cada website.

Por último, o código fonte dos sistemas foi baixado dos repositórios, e o aplicativo VerveineJ foi utilizado para extrair um modelo orientado a objetos de acordo com o FAMIX, um meta-modelo independente de linguagem para códigos orientados a objetos. Como são necessárias várias versões dos sistemas, esse processo foi repetido em intervalos bi-semanais durante todo o período analisado dos sistemas.

Para que possamos analisar a presença de *bugs* em partes específicas do sistema, primeiramente é feita uma ligação entre cada bug extraido no issue tracker e as partes do sistema que foram afetadas por ele. Cada commit nos repositórios possui uma mensagem do desenvolvedor, que geralmente inclui uma referência a uma entrada no issue tracker (p. ex. ”fixed bug 123”). Essas mensagens permitem que seja feita uma ligação entre *bugs* e arquivos no controle de versão (e consequentemente, classes). *bugs* que afetam classes de teste são desconsiderados.

Após esse processo, as datas de abertura e resolução dos *bugs* são analisadas, para que seja definido, além de quais classes foram afetadas, durante quais versões dos sistemas os *bugs* estiveram presentes. Um bug é considerado presente em uma versão se sua data de abertura for anterior à da versão, e a data do commit que o resolveu for

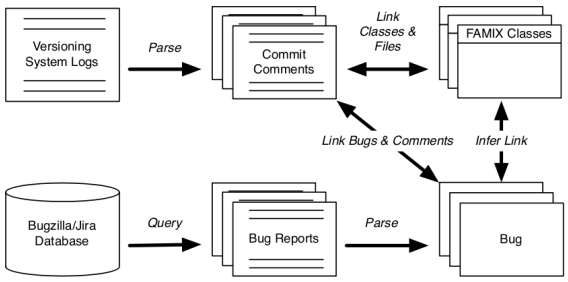

Figure 1. Processo de ligação entre *bugs* e classes.

posterior.

Até este ponto já temos um modelo que inclui informações de várias versões dos códigos fonte, e dados sobre o tempo de vida de *bugs* nos sistemas. O último passo é extrair as métricas de código fonte, para que sejam analisadas em conjunto com as informações geradas nas etapas anteriores. A extração é feita para cada versão de cada sistema, e são criadas séries temporais para elas.

Table 2. Métricas extraídas

Type	Metric	
CK	WMC	Weighted Method Count
CK	DIT	Depth of Inheritance Tree
CK	RFC	Response For Class
CK	NOC	Number Of Children
CK	CBO	Coupling Between Objects
CK	LCOM	Lack of Cohesion in Methods
OO	FanIn	Number of other classes that reference the class
OO	FanOut	Number of other classes referenced by the class
OO	NOA	Number of attributes
OO	NOPA	Number of public attributes
OO	NOPRA	Number of private attributes
OO	NOAI	Number of attributes inherited
OO	LOC	Number of lines of code
OO	NOM	Number of methods
OO	NOPM	Number of public methods
OO	NOPRM	Number of private methods
OO	NOMI	Number of methods inherited

As séries temporais presentes no dataset (*bugs* e métricas) são armazenadas no formato CSV, de forma que cada linha corresponde a uma classe do sistema, e cada coluna é o valor representado (valor da métrica ou quantidade de *bugs*) em uma bissemana.

2.2. Ferramentas

Para a criação do dataset foram utilizadas as seguintes ferramentas:

- *VerveineJ* (disponível em http://www.moosetechnology.org/tools/verveinej) para a conversão de códigofonte Java em modelos FAMIX.
- *Moose* (disponível em http://www.moosetechnology.org) para ler os modelos FAMIX, e extrair as métricas de código fonte.

Além disso, duas outras ferramentas tiveram de ser desenvolvidas, para a geração das séries temporais.

- Uma ferramenta para pegar os valores das métricas fornecidos pela ferramenta Moose e transformá-los em séries temporais.
- Uma ferramenta para transformar as informações dos *bugs* em classes em uma série temporal.

3. Avaliação

Para a avaliação do dataset, os dados gerados foram utilizados para fazer uma análise em um dos sistemas. O sistema escolhido para a análise foi o Eclipse JDT Core. A Figura 2 mostra um gráfico com a série temporal de *bugs* do sistema.

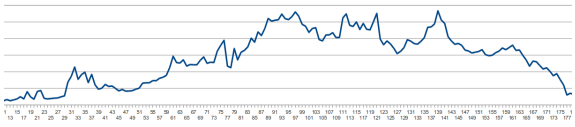

Figure 2. Série temporal completa de *bugs* do Eclipse JDT Core

Para que seja possível avaliar se as informações presentes no dataset são válidas para analisar a evolução dos sistemas, é necessário que essas informações reflitam os eventos ocorridos durante o desenvolvimento desses sistemas. Uma informação já verificada em muitos projetos é o fato de que, logo após o lançamento de uma versão de um software, há um aumento no número de *bugs* reportados no sistema de issue tracking.

A versão 2.0 da JDT foi lançada em 27 de Junho de 2002 (http://archive.eclipse.org/eclipse/downloads/index.php). Essa data corresponde à 26ª bissemana da história do sistema. Na Figura 3 é mostrada a série temporal de *bugs* do JDT entre as bissemanas 24 e 48. Pelo gráfico é possível ver que, aproximadamente um mês depois do lançamento da versão 2.0, houve um aumento no número de *bugs*, de acordo com o esperado.

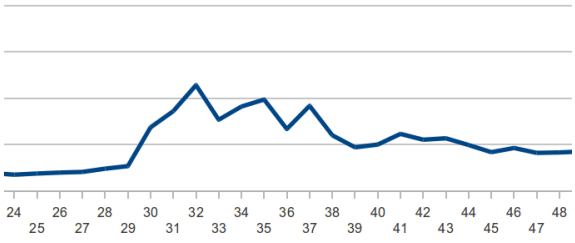

Figure 3. *bugs* do Eclipse JDT Core entre as bissemanas 24 e 48

Após o lançamento da versão 2.0, foram lançadas mais duas versões, 2.0.1 e 2.0.2. Apesar de serem versões com o objetivo de corrigir *bugs* da versão anterior, elas também estão associadas com um crescimento no número de *bugs* do sistema. A versão 2.0.1 foi lançada em 29 de Agosto de 2002, o que corresponde à bissemana 31 da vida do sistema, e a versão 2.0.2 foi lançada em 7 de Novembro de 2002, o que corresponde à bissemana 36. Os aumentos nos números de *bugs* correspondentes aos lançamentos das versões 2.0.1 e 2.0.2 podem ser vistos nas bissemanas 35 e 37, respectivamente.

4. Trabalhos Relacionados

[D'Ambros et al. 2010] faz uma comparação entre várias técnicas de predição de *bugs*, e um dataset foi criado para fazer o estudo. Este dataset estende o dataset de Dambros de várias formas: (1) agregando novamente várias versões dos sistemas, e recalculando as métricas; (2) expandindo o número de versões dos sistemas, e consequentemente a tamanho das séries temporais, de pouco menos que 100 para 180; (3) incluindo informações sobre quatro novos sistemas: Geronimo, Hadoop MapReduce, TV-Browser e Hibernate.

[Rajesh Vasa and Jones 2010] criaram um dataset (Helix, disponível em *http://www.ict.swin.edu.au/research/projects/helix*) com informações temporais sobre valores de métricas de código fonte. Porém, ele não possui informações sobre algumas das métricas inclusas neste dataset, incluindo as métricas CK (com exceção de NOC e DIT).

[Tempero et al. 2010] criou o Qualitas Corpus, um dataset bem conhecido para estudos empíricos em engenharia de software. Ele contém informações sobre 111 sistemas, mas somente provê informações evolucionárias sobre 14 deles, e somente 1 deles possui mais de 70 versões. O Qualitas Corpus também não inclui informações temporais sobre métricas de código fonte.

References

D'Ambros, M., Lanza, M., and Robbes, R. (2010). An extensive comparison of bug prediction approaches. In *Proceedings of MSR 2010 (7th IEEE Working Conference on Mining Software Repositories)*, pages 31 – 41. IEEE CS Press.

R. Kollmann, P. S. and Stroulia, E. (2002). A study on the current state of the art in toolsupported uml-based static reverse engineering. In *Proceedings of WCRE 2002*, pages 22–32.

Rajesh Vasa, M. L. and Jones, A. (2010). Helix - Software Evolution Data Set.

Tempero, E., Anslow, C., Dietrich, J., Han, T., Li, J., Lumpe, M., Melton, H., and Noble, J. (2010). Qualitas corpus: A curated collection of java code for empirical studies. In *2010 Asia Pacific Software Engineering Conference (APSEC2010)*, pages 33 – 345.