Predicting how Difficult Bugs are to Detect Using Source Code Metrics

Kevin Jalbert, Jeremy S. Bradbury ● Faculty of Science ● University of Ontario Institute of Technology ● Oshawa, Ontario, Canada
{kevin.jalbert, jeremy.bradbury}@uoit.ca

1. Motivation

- The ability to localize faults is highly desirable in software systems.
- **Why?** It is possible to predict problematic areas in the source code to focus testing resources.

  Research Goal: Predict the difficulty* of detecting bugs within a Java class/package using static source code metrics and a support vector machine.
  *** Mutation score is used as a proxy to quantify the difficulty of detecting a bug.**

  - Static source code metrics might provide insight on how difficult a bug might be to detect.
  - A Support Vector Machine (SVM) can predict how difficult bugs are to detect using only source code metrics.
- **Related work** with other machine learning techniques and classification criterion [1, 2, 3].

2. Approach

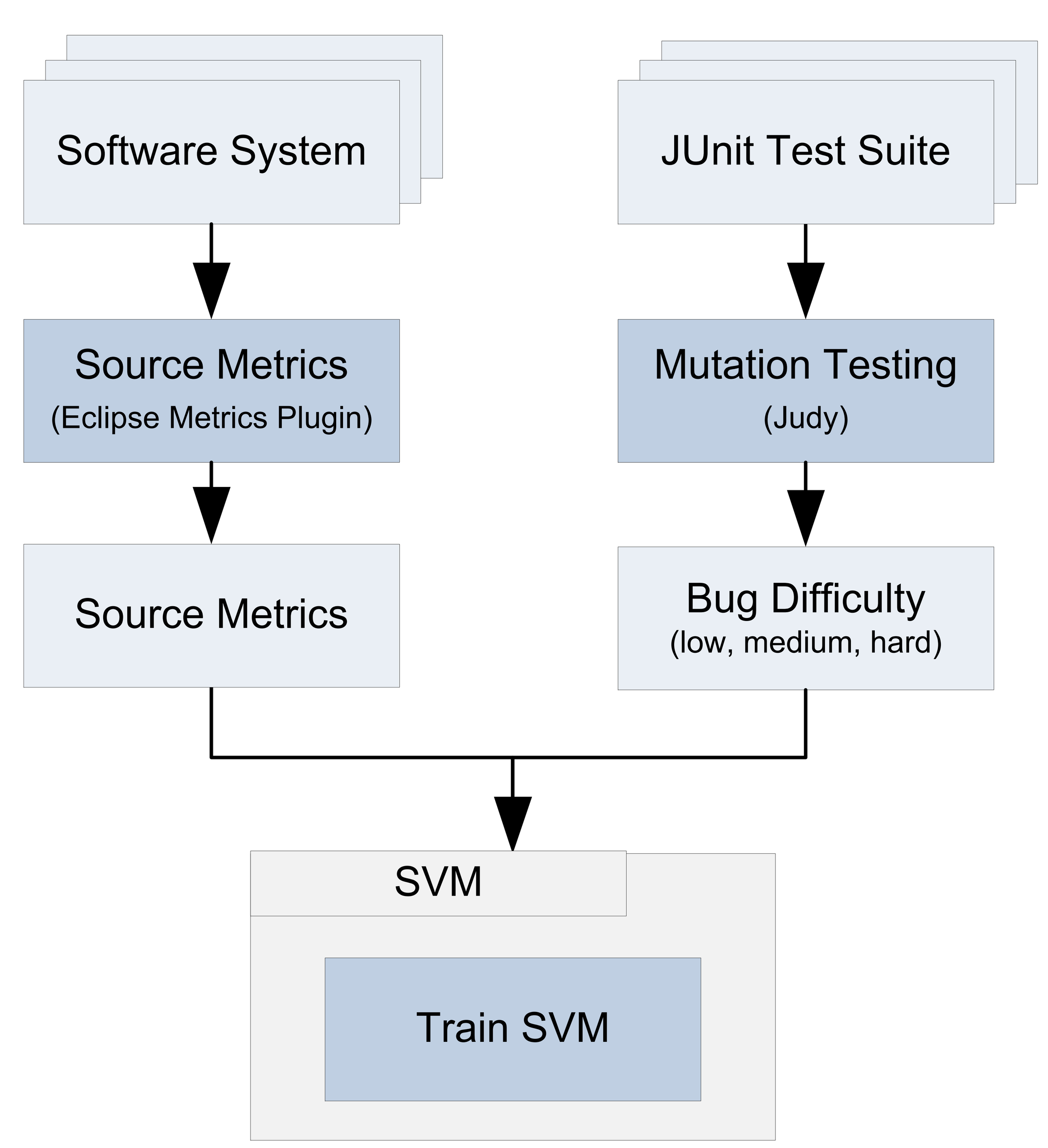

Figure 1: Overview of the first stage: the training process of a SVM.

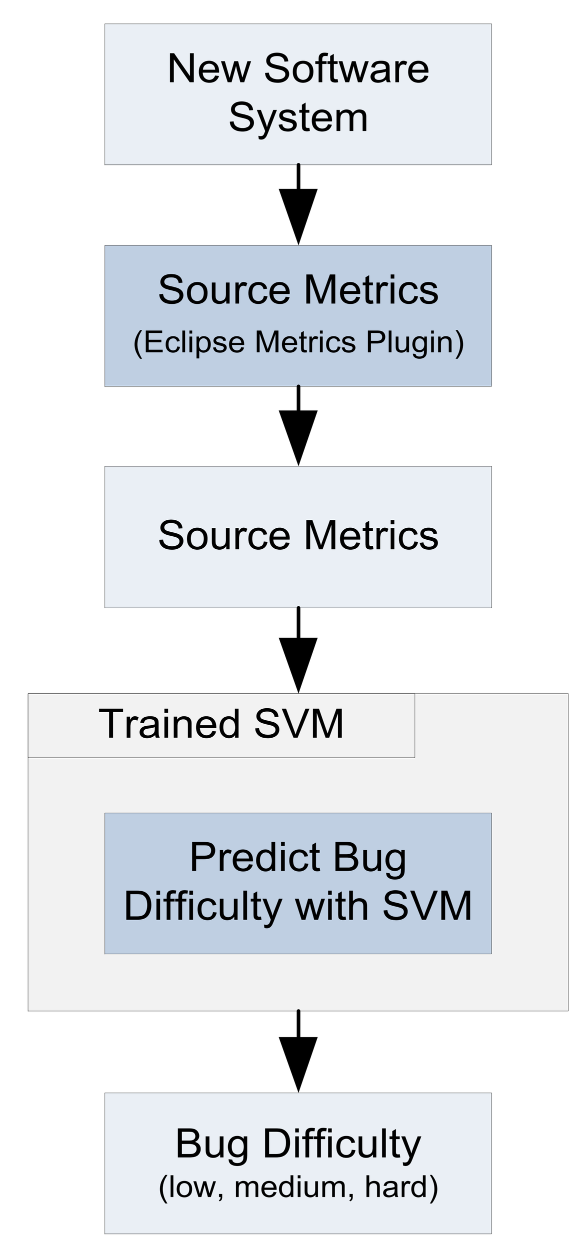

Figure 2: Overview of the second stage: the prediction process of a SVM.

- The approach used to predict the difficulty of detecting bugs within a software system is a two stage process.
  - Tools used : LIBSVM [4], Judy [5] and Eclipse Metrics Plugin [6].
- Figure 1 illustrates the training stage. The source metrics and bug difficulty of source code units (packages, classes, etc…) are collected and used to train a SVM
- Figure 2 illustrates the predicting stage. The prediction of bug difficulty in the new software system is based on that system's, along with the trained SVM.

[1] L. Briand, J. Wust, J. W. Daly, and D. V. Porter. "Exploring the relationships between design measures and software quality in object-oriented systems". *Journal of Systems and Software*, 51(3):245–273, May 2000.
[2] D. Gray et al. "Using the Support Vector Machine as a Classification Method for Software Defect Prediction with Static Code Metrics". *Engineering Applications of Neural Networks, Communications in Computer and Information Science*, 43, (2009), 223-234.
[3] Y. Singh, A. Kaur, R. Malhorta. "Application of Support Vector Machine to Predict Fault Prone Classes". *In ACM SIGSOFT Software Engineering Notes*, 34(1), (2009)
[4] C.-C. Chang and C.-J. Lin. "LIBSVM: a library for support vector machines", 2001. Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm.
[5] L. Madeyski and N. Radyk. "Judy - a mutation testing tool for java", *Software, IET* , vol.4, no.1, pp.32-42, Feb. 2010.
[6] Eclipse Metrics Plugin available at http://metrics.sourceforge.net/.
[7] K. Meffert et al. "JGAP - Java Genetic Algorithms and Genetic Programming Package". Software available at http://jgap.sf.net.

3. Mutation Testing & Source Code Metrics

- **Mutation testing** inserts common faults into source code using mutation operators (see Figure 3).
- The percentage of mutants caught in the testing result in a mutant score.
- Mutant score estimate the difficulty of detecting bugs.
  - **Easy** (0.67 to 1.00)
  - **Medium** (0.34 to 0.66)
  - **Hard** (0.00 to 0.33)

Abbre-viation	Description	Scope
ABS	Absolute value insertion	Method
AOR	Arithmetic operator replacement	Method
LCR	Logical connector replacement	Method
ROR	Relational operator replacement	Method
UOI	Unary operator insertion	Method
UOD	Unary operator deletion	Method
SOR	Shift operator replacement	Method
LOR	Logical operator replacement	Method
COR	Conditional operator replacement	Method

Abbre-viation	Description	Scope
ASR	Assignment operator replacement	Method
EOR	Reference assignment and content assignment replacement	Class
EOC	Reference comparison and content comparison replacement	Class
JTD	*this* keyword deletion	Class
JTI	*this* keyword insertion	Class
EAM	Accessor method change	Class
EMM	Modifier method change	Class

Figure 3: Mutation operators used from the Mutation testing tool Judy.

- **Static source code metrics** are used to measure informative details about the source code (see Figure 4).
- Object-oriented metrics are used to measure design complexity.
- Various metrics are only applicable on certain scope levels of the source code.

Abbre-viation	Description	Scope
MLOC	Method lines of code	Method
NBD	Nested block depth	Method
VG	McCabe cyclomatic complexity	Method
PAR	Number of parameters	Method
NORM	Number of overridden method	Class
NOF	Number of attributes	Class
NSC	Number of children	Class
DIT	Depth of inheritance tree	Class
LCOM	Lack of cohesion of methods	Class
NSM	Number of static methods	Class
NOM	Number of methods	Class
SIC	Specialization index	Class

Abbre-viation	Description	Scope
WMC	Weighted method per class	Class
NSF	Number of static attributes	Class
NOC	Number of classes	Package
CA	Afferent Coupling	Package
NOI	Number of interfaces	Package
RMI	Instability	Package
CE	Efferent coupling	Package
RMD	Normalized distance	Package
RMA	Abstractness	Package
NOP	Number of packages	Project
TLOC	Total lines of code	Project

Figure 4: Static source metrics used from the Eclipse Metrics Plugin.

4. Preliminary Results

- The software system under observation was JGAP [7], due to its mature Java JUnit Test Suite.
- Figure 5 shows the amount of useful units found in JGAP. It does not make sense to use unit data that did not include any mutations (hence no estimate on bug difficulty).
- Figure 6 shows the cross-validation accuracy (ten folds) using the collected source metrics.

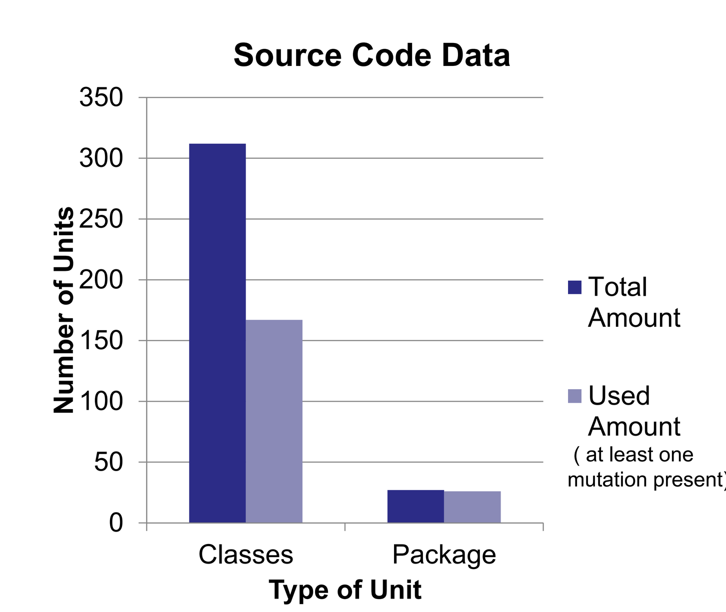

Figure 5: Amount of source code units collected from JGAP.

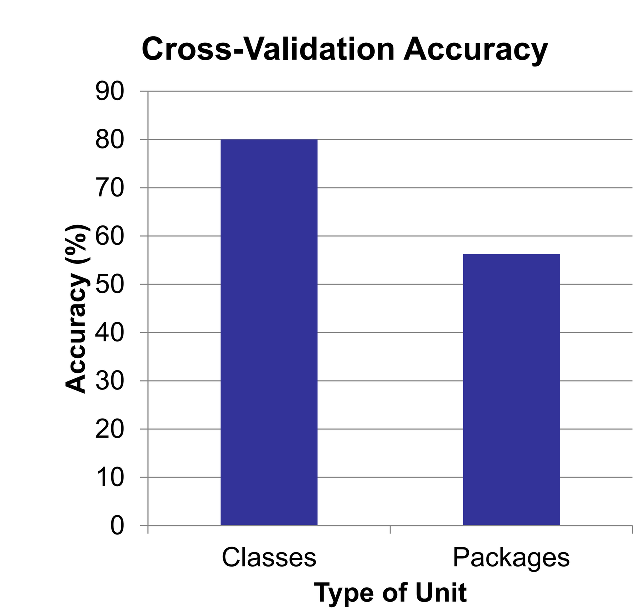

Figure 6: Cross-validation accuracy (ten folds) of the relevant amount of source code units collected from JGAP.

5. Conclusions & Future Work

- Using our approach for predicting the difficulty of detecting bugs within Java source code units we were able to achieve a cross-validation accuracy of **80.00%** for classes and **56.25%** for packages.
- Future work would include:
  - Expand data set to include more open-source software systems.
  - Consider hierarchical metrics (average values for all methods within a class/package, etc…).
  - Expand scope to include methods and project predictions

Consortium for Software Engineering Research (CSER) 2011 Spring Meeting ● Kingston, Ontario, Canada ● June 2011

Funding provided by: NSERC CRSNG