Using program slicing data to predict code faults

David Bowes
University of Hertfordshire

February 10, 2010

Why?

- Defect prediction 70% using machine learning
- Slicing Metrics rarely used for defect prediction
- Slicing metrics have some relationship of cohesion
- Slicing metrics do not tend to be a proxy for LOC

Code example

```
public class Fib {

   int start=1;//may be err?

   public static void main(String[] args) {
      Fib f = new Fib();
      for (int i = 1; i < 10; i++) {
         System.out.println(i+"    "+f.fib(i));
      }
   }

   public int fib(int n) {
      int a = 0, b = 1;
      int c = start, d = 1;//fix me?
      while (c < n) {
         System.out.printf(" debug %d\r\n", d);
         d = a + b;
         a = b;
         b = d;
         c++;
      }
      return b;
   }
}
```

Slicing Metrics

Weiser ,Ott and Thuss defined a set of slice based metrics including:

- Tightness :The number of statements which are in every slice. High tightness values suggest that the code is cohesive.
- Overlap : Indicates how many statements in a slice are found only in that slice
- Coverage : Compares the length of slices to the length of the entire program
- Min Coverage :The length of the shortest slice as a proportion of the program length
- Max Coverage : Length of the longest slice as a proportion of the program length

New metric Counsel et al

- NHD

Which variables to choose?

Previous studies exploring the efficacy of slice-based metrics have tended to use different sets of variables in specifying the slices:

Categories	Description	Studies
Formal ins (V_i)	Input parameters for the function specified in the module declaration	6
Formal outs (V_o)	The set of return variables	8
Global variables (V_g)	The set of variables which are used or may be affected by the module	9
printfs (V_p)	Variables which appear as formal outs in the list of parameters in an output statement (e.g. printf)	7

Code example

```
public class Fib {

   int start=1;//may be err?

   public static void main(String[] args) {
      Fib f = new Fib();
      for (int i = 1; i < 10; i++) {
         System.out.println(i+"    "+f.fib(i));
      }
   }

   public int fib(int n) {
      int a = 0, b = 1;
      int c = start, d = 1;//fix me?
      while (c < n) {
         System.out.printf(" debug %d\r\n", d);
         d = a + b;
         a = b;
         b = d;
         c++;
      }
      return b;
   }
}
```

What impact does the choice of variables have?

- Studied barcode, open source barcode printing utility.
  - http://ar.linux.it/software/barcode/barcode.html
- For 15 variants of variables:

V_i	V_o	V_g	V_p	Overlap	Tightness	Coverage	Min C	Max C
+	+	+	+	0.649	0.481	0.691	0.523	0.901
+	+	+		0.643	0.482	0.705	0.524	0.901
+	+		+	0.712	0.551	0.717	0.588	0.898
+		+	+	0.759	0.563	0.712	0.587	0.892
	+	+	+	0.745	0.519	0.671	0.543	0.845
+	+			0.728	0.560	0.743	0.590	0.898
		+	+	0.772	0.518	0.653	0.538	0.820
+			+	0.839	0.672	0.764	0.694	0.885
	+	+		0.767	0.521	0.653	0.544	0.761
+		+		0.728	0.560	0.743	0.590	0.898
	+		+	0.820	0.591	0.688	0.610	0.792
+				0.944	0.823	0.856	0.832	0.885
	+			1.000	0.612	0.612	0.612	0.612
		+		0.851	0.538	0.639	0.547	0.717
			+	0.749	0.464	0.597	0.496	0.778

Relating slicing metrics to 'fault' data:Getting data

Technique:

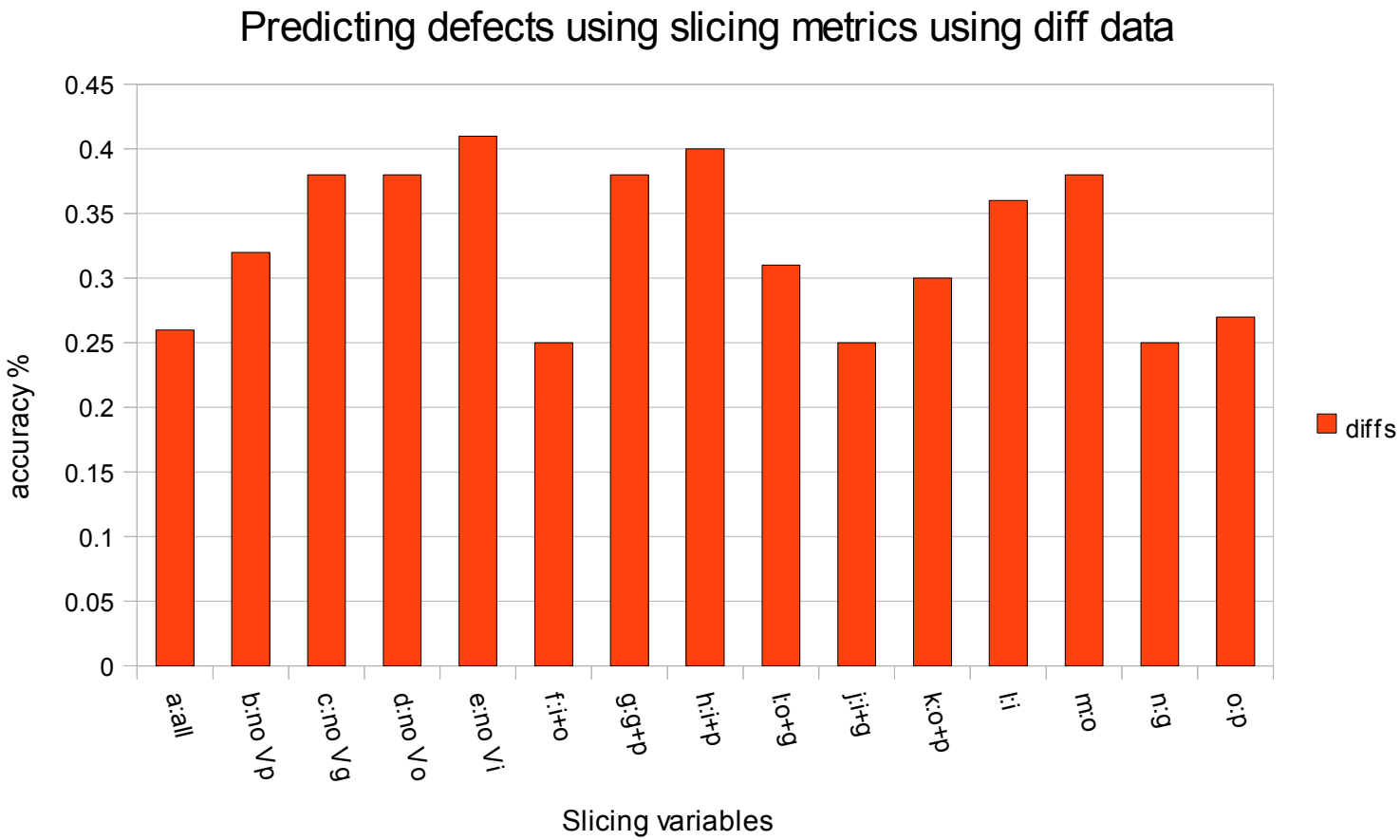
- Find a bug fix
- Assume before (α) was defective and after (β) was less defective.
- do the metrics of α predict a change to less defective state β?[1]

[1] This technique produces balanced data so accuracy can be used to compare results.

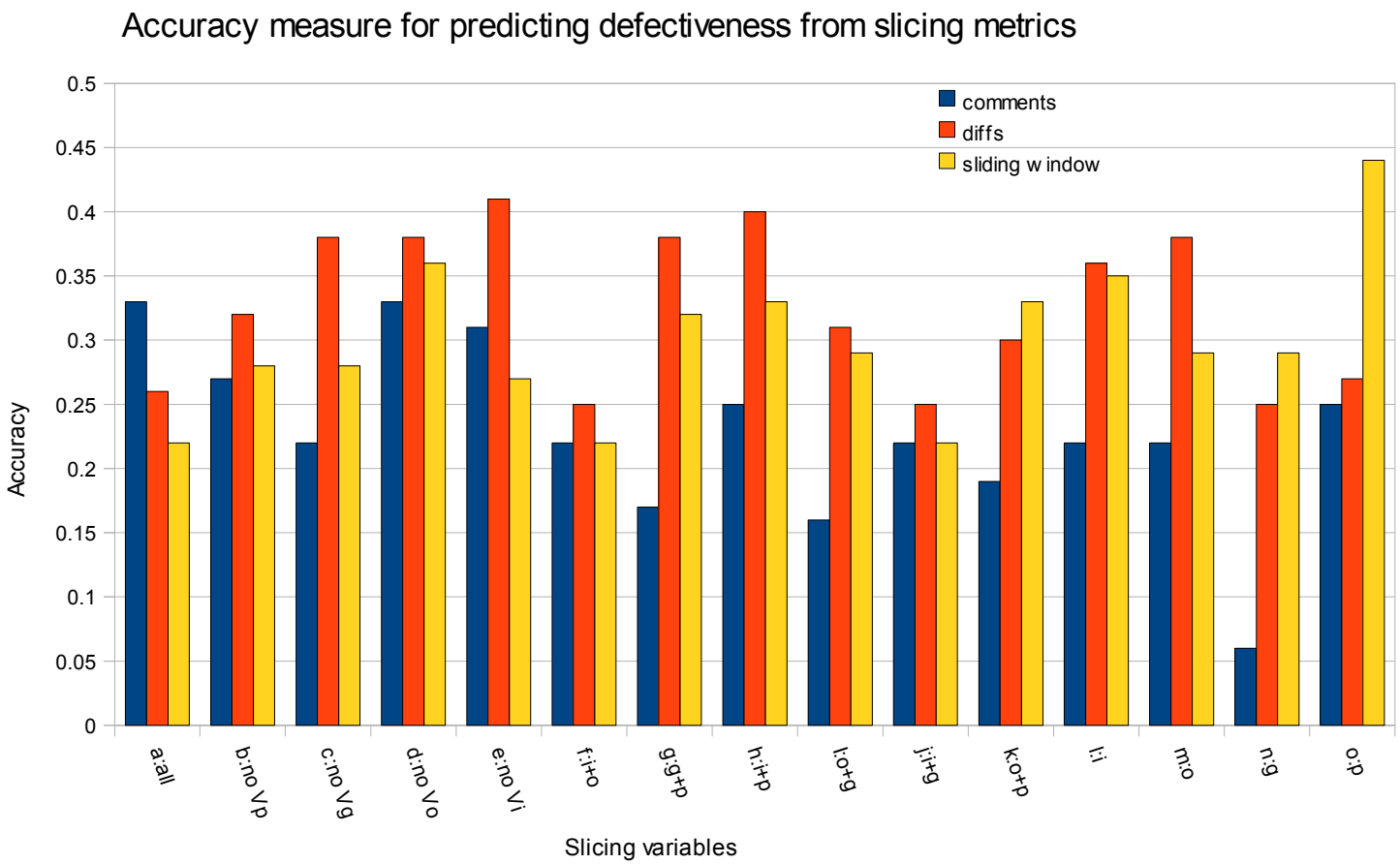
Wack it into Weka

- For each variant of slicing variable:
  - format the data for Weka
  - use Naive Bayesian Classifier
  - 10 fold cross validation
  - report accuracy

Results using diff data



Results


Accuracy measure for predicting defectiveness from slicing metrics
comments
diffs
sliding w indow
Accuracy
0.5
0.45
0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
a:all
b:no Vp
c:no Vg
d:no Vo
e:no Vi
f:i+o
g:g+p
h:i+p
i:o+g
j:i+g
k:o+p
l:i
m:o
n:g
o:p
Slicing variables

Conclusion/Analysis

- Choice of slicing variables has an impact on slicing metrics
- Learning defects from slicing metrics may be domain specific
- Slicing metrics on their own do not predict defects 'better' than other studies. Or even picking a classification at random
- There aren't enough bug fixes!
- Looking at defect boundaries may not be the best approach.
  - A patch is likely to need patching.... does the quality of code improve with patching?
  - defect mining with defect boundaries may predict if the patch was good if we study the pattern of patching after.

Metric	Formula
Tightness	$= \frac{\lvert SL_{int} \rvert}{\text{length}(M)}$
Overlap	$= \frac{1}{\lvert V_o \rvert} \sum_{i=1}^{V_o} \frac{\lvert SL_{int} \rvert}{\lvert SL_i \rvert}$
Coverage	$= \frac{1}{\lvert V_o \rvert} \sum_{i=1}^{V_o} \frac{\lvert SL_i \rvert}{\text{length}(M)}$
Min Coverage	$= \frac{\min_i \lvert SL_i \rvert}{\text{length}(M)}$
Max Coverage	$= \frac{\max_i \lvert SL_i \rvert}{\text{length}(M)}$

Key : M Set of program vertices in a method, NB $length(M) \equiv |M|$

V_0 Set of variables used to slice a method.

SL_i Set of program vertices in the slice of the i'th variable in V_0

SL_{int} Intersection of all slices formed from each V_0