MUTATION-AWARE FAULT PREDICTION

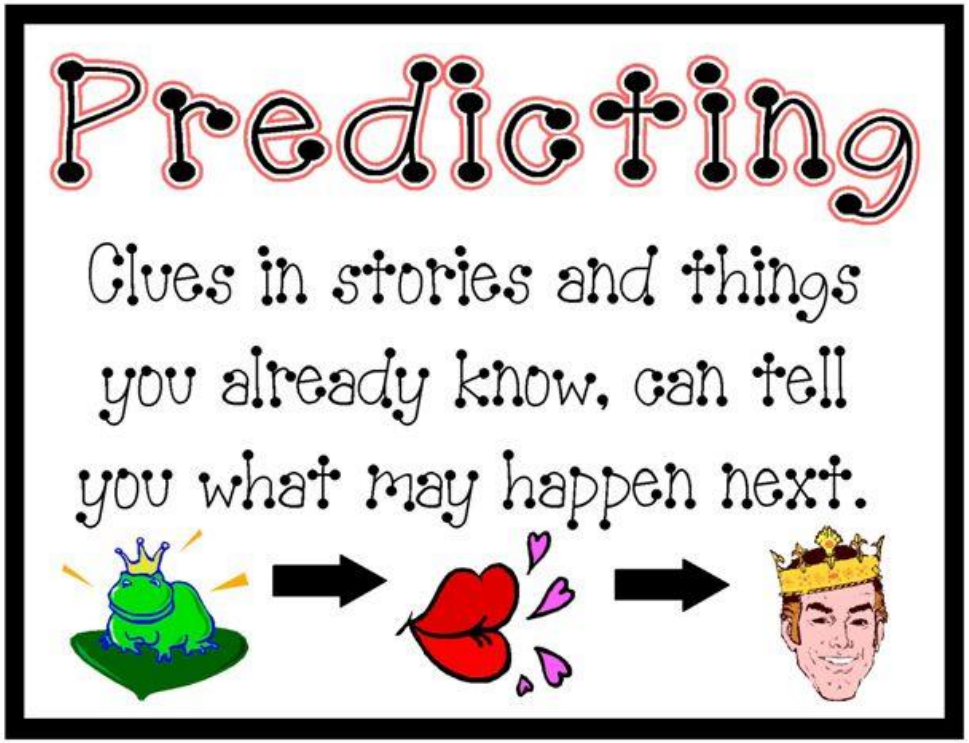
David Bowes: Hertfordshire University, UK
Tracy Hall: Brunel University London, UK
Mark Harman: University College London, UK
Yue Jia: University College London, UK
Federica Sarro: University College London, UK
Fan Wu: University College London, UK

Agenda

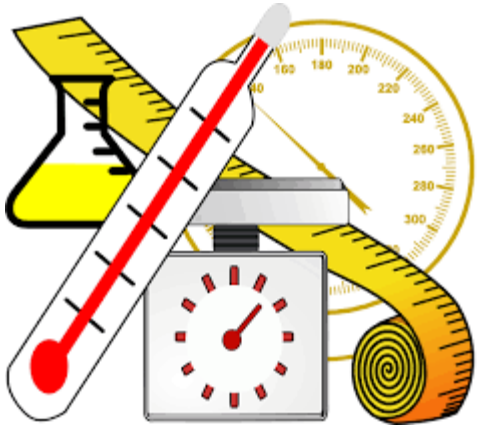
- Introduction
- Background
- Methods
- Results
- Conclusions

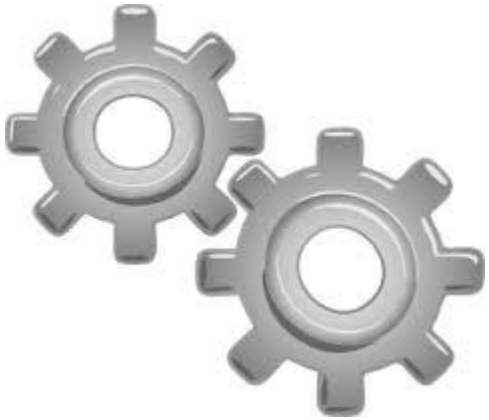
Introduction

- Predicting defects is important but hard.



Background: Defect prediction





Data about the code → Data analysis method → Defective code predictions

Background: mutation testing

- Creates a mutant of a program.
- A mutant is `killed' (identified) by a test case.
- Mutants are created by mutation operators

Methods: Mutation Operators

Index	Mutation Operator	Description
1	Return Value Mutator	Mutates the return values of method calls.
2	Negate Conditionals Mutator	Mutates conditionals between `==` and `!=`, `<` and `>=`, `>` and `<=`.
3	Increments Mutator	Replaces increments (`++`) with decrements (`--`) and vice versa.
4	Void Method Calls Mutator	Removes void method calls.
5	Conditionals Boundary Mutator	Replaces the relational operators between `<` and `<=`, `>` and `>=`.
6	Invert Negatives Mutator	Inverts negation of integer and floating point numbers.
7	Math Mutator	Swaps operations within groups: {`+`,`-`}, {`*`, `/`, `%`}, {`&`, `|`, `^`}, {`<<`, `>>`, `>>>`}
A	All of the above	Used to denote mutation metrics that count all the above metric instance

Research Questions

RQ1: Is the performance of predictive modelling techniques improved when they have mutation metrics available?

RQ2: What is the effect size of improvements in predictive model performance due to mutation metrics?

RQ3: What is the relative performance of static and dynamic mutation metrics?

Methods

- Subject systems…

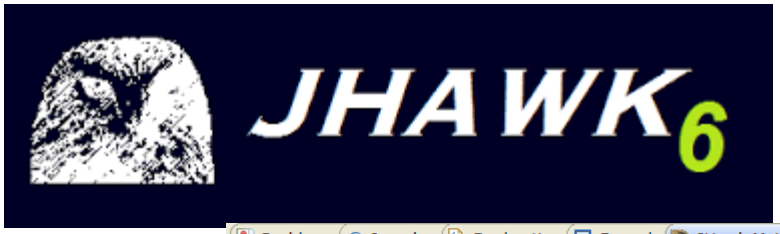
Methods: Subject Systems

	Apache	Eclipse	TelCom
LOC Total	22,430	113,106	84,731
No. Classes	74	727	1295
No. Faulty Classes	30	191	42
% Faulty Classes	40.54%	26.27%	3.24%
No. Tests	245	8,393	2,846

Methods: Data collected

- Defect data collected using SZZ Algorithm
- Commonly used source code metrics collected…

Methods: Metrics



Problems | Javadoc | Declaration | Console | JHawk Metrics

Dashboard | System Details | System Packages | Package Details | Class Details | All Classes | All Methods | Export

Packages in this system

Name	No. Classes	No. Methods	NOS	TCC	AVCC	HBUG	HEFF	HLTH	HVOL	MI	MINC
org.junit.tests.running.methods	62	211	736	71	0.34	7.83	165681.06	5487	23479.26	130.66	130.66
junit.framework	17	149	557	41	0.28	6.16	167133.46	4266	18474.07	150.10	129.48
junit.tests.framework	37	129	544	41	0.32	4.60	107279.83	3498	13800.33	125.81	125.81
org.junit.tests.junit3compatibility	46	99	495	47	0.47	5.84	146767.79	3861	17520.04	120.13	120.13
org.junit.tests.experimental.rules	51	89	448	48	0.54	5.03	121537.06	3447	15085.39	120.38	120.38
org.junit.tests.experimental.theo...	36	79	383	39	0.49	5.87	191947.37	3758	17596.12	122.04	122.04
org.junit.tests.assertion	3	71	303	5	0.07	4.17	88755.88	2812	12513.39	124.76	124.76
org.junit.internal.runners	9	40	291	26	0.65	3.99	151453.48	2350	11962.96	111.40	111.40
org.junit.runners	9	62	276	17	0.27	3.77	94737.46	2383	11315.09	124.00	123.77
org.junit.tests.running.classes	29	67	263	24	0.36	3.41	81358.89	2311	10239.09	129.15	129.15

Name of package	org.junit	**Total number of Classes**	5
Total number of methods	67	**Total number of Java statements**	251
Total Cyclomatic Complexity	12	**Average Cyclomatic Complexity for**	0.18
Cumulative Halstead bugs	3.26	**Cumulative Halstead effort**	98323.41
Cumulative Halstead length	2248	**Cumulative Halstead volume**	9789.64
Maintainability Index	84.53	**Maintainability Index (Not including con**	128.66
Abstractness	0.00	**Fan In**	0
Fan Out	4	**Instability**	1.00
Distance	0.00	**Maximum Cyclomatic Complexity for me**	5

Methods: Data collected

- Defect data collected using SZZ Algorithm
- Commonly used source code metrics collected using JHawk.
- PITest used to mutate the code and identify methods where the unit tests do not detect/kill the mutation
- Our tool then computes five types of Static and dynamic mutation metrics

Methods: Types of Mutation Metrics

Metric	Description	Type
MuNOM^S	Number of generated mutants	Static
MuNOC	Number of mutants covered by tests	Dynamic
MuNNC	Number of mutants not covered by tests	Dynamic
MuMS	Mutation score of generated mutants	Dynamic
MuMSC	Mutation score of covered mutants	Dynamic

Methods: Data collected

- Defect data collected using SZZ Algorithm
- Commonly used source code metrics collected using JHawk.
- PITest used to mutate the code and identify methods where the unit tests do not detect/kill the mutation
- Our tool then computes five types of Static and dynamic mutation metrics
- PITEST has 7 mutations operators
- We used 40 mutation metrics in total

Methods: Machine Learning Techniques

- Techniques used to predict if a class is defective:
  - Naïve Bayes
  - Logistic Regression
  - J48
  - Random Forest
- Used the WEKA Wrapper Subset Selection Filter.
- Performed 10-fold cross validation repeated 50 times
- Evaluated the results using Matthews Correlation Coefficient
  - MCC values range between -1 and +1.

Results

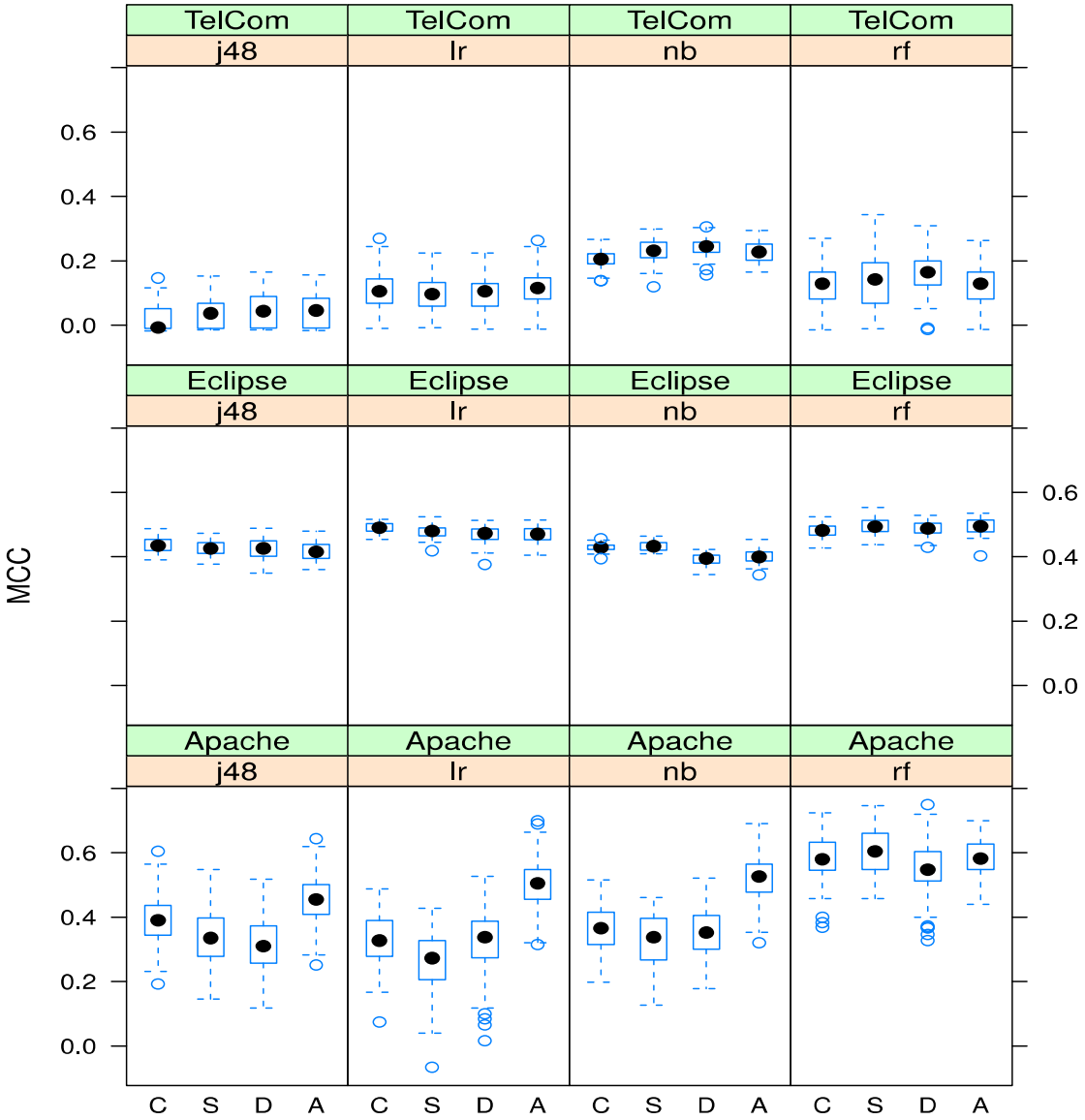
RQ1: Is the performance of predictive modelling techniques improved when they have mutation metrics available?

RQ2: What is the effect size of improvements in predictive model performance due to mutation metrics?

RQ3: What is the relative performance of static and dynamic mutation metrics?

RQ1: Do Mutation Metrics Improve Performance?

C- code
S- static
D- dynamic
A- All



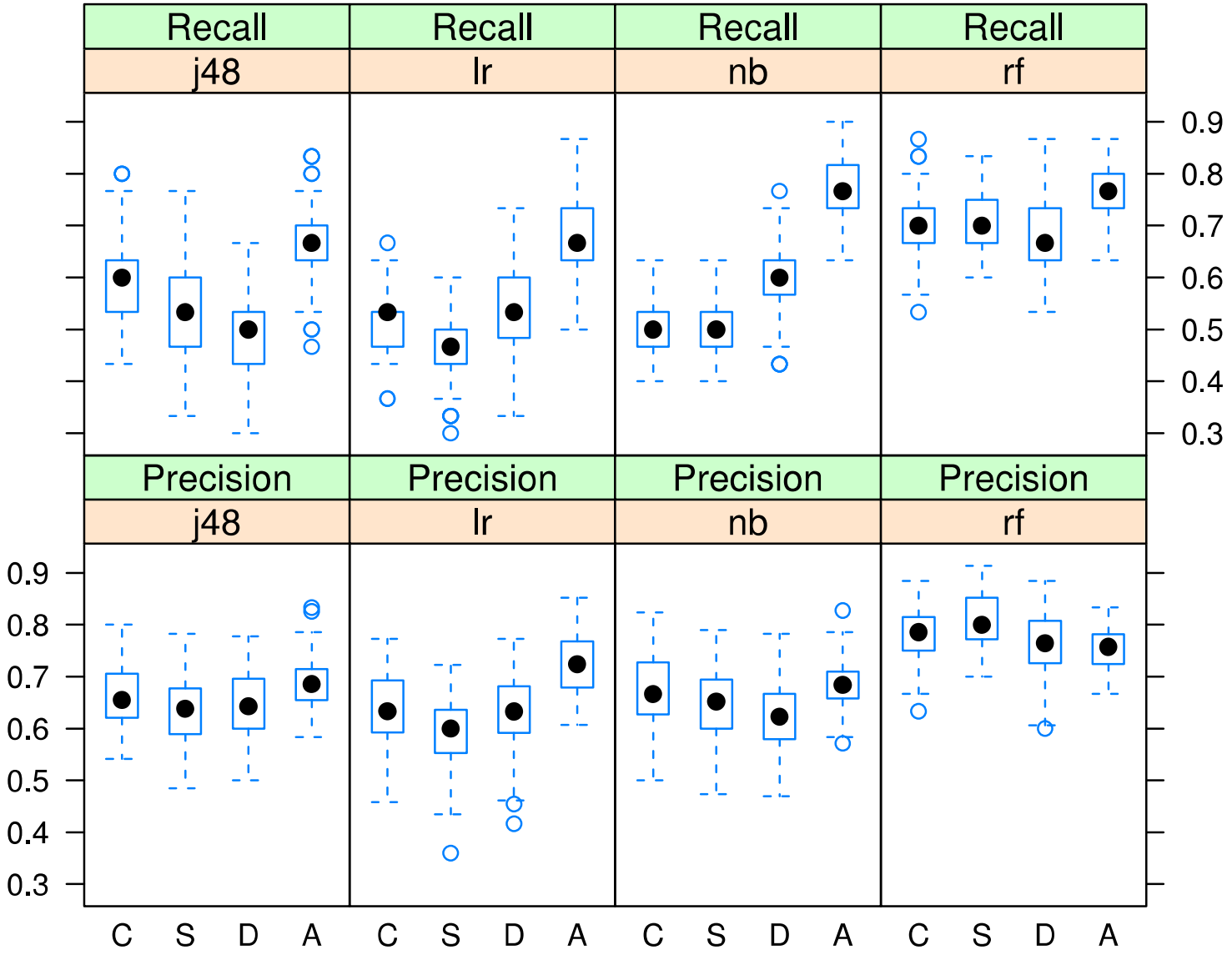
RQ1: Do Mutation Metrics Improve Performance?

(a) MCC

	Apache	Eclipse	TelCom
Naïve Bayes	<0.001 (0.182)	<0.001 (-0.030)	0.018 (0.022)
Logistic Regression	<0.001 (0.164)	<0.001 (-0.020)	0.338 (0.005)
Random Forest	0.858 (-0.008)	0.005 (0.013)	0.920 (0.000)
J48	<0.001 (0.070)	<0.001 (-0.019)	0.007 (0.057)

RQ1: Do Mutation Metrics Improve Performance?

Apache



RQ2: What is the effect size of improvements in predictive model performance due to mutation metrics?

Top 10 most frequently used metrics

	Apache	Eclipse	TelCom
Naïve Bayes	HEFF(492)	FOUT(500)	FOUT(465)
	MuMS2(488)	MINC(500)	**MuMS7**(458)
	MuMSA(340)	NOM(500)	**MuMS4**(427)
	R-R(306)	**MuMSC4**(471)	**MuNOM^{S}7**(418)
	MI(266)	**MuNOC2**(342)	**MuNNC3**(416)
	LCOM(155)	**MuMSC3**(309)	**MuMSA**(398)
	MuNOM^{S}3(120)	**MuMS1**(226)	MI(373)
	SIX(117)	MAXCC(210)	**MuNNC7**(317)
	MuMSC3(85)	**MuMSA**(209)	SIX(268)
	MuNOC7(71)	**MuMSC5**(206)	**MuMSC7**(259)

RQ2: What is the effect size of improvements in predictive model performance due to mutation metrics?

Frequency of mutation metrics in the top 10 metrics

Technique	Apache	Eclipse	TelCom
Naïve Bayes	5	6	7
Logistic Regression	6	1	7
Random Forest	5	4	1
J48	6	5	3

RQ2: What is the effect size of improvements in predictive model performance due to mutation metrics?

Effect of each metric to model performance: global

	Apache	Eclipse	TelCom
Naïve Bayes	HEFF(0.292,492)	FOUT(0.104,500)	FOUT(0.094,465)
	MuMS2(0.252,488)	MINC(0.074,500)	**MuNNC3**(0.056,416)
	MuMSA(0.152,340)	**MuMSC4**(0.046,471)	**MuMS7**(0.055,458)
	MI(0.073,266)	MAXCC(0.034,210)	**MuMSA**(0.055,398)
	R-R(0.063,306)	**MuNOC2**(0.029,342)	**MuMS4**(0.054,427)
	LCOM(0.042,155)	NOM(0.025,500)	SIX(0.051,268)
	MuNOM^{S}3(0.036,120)	**MuMS1**(0.023,226)	**MuNOM^{S}7**(0.049,418)
	MuMSCA(0.033,66)	**MuMSC3**(0.021,309)	MI(0.047,373)
	MuMSC7(0.031,49)	**MuMS6**(0.018,254)	**MuNNC4**(0.042,225)
	MuMSC3(0.021,85)	**MuMSC5**(0.018,206)	CBO(0.036,238)

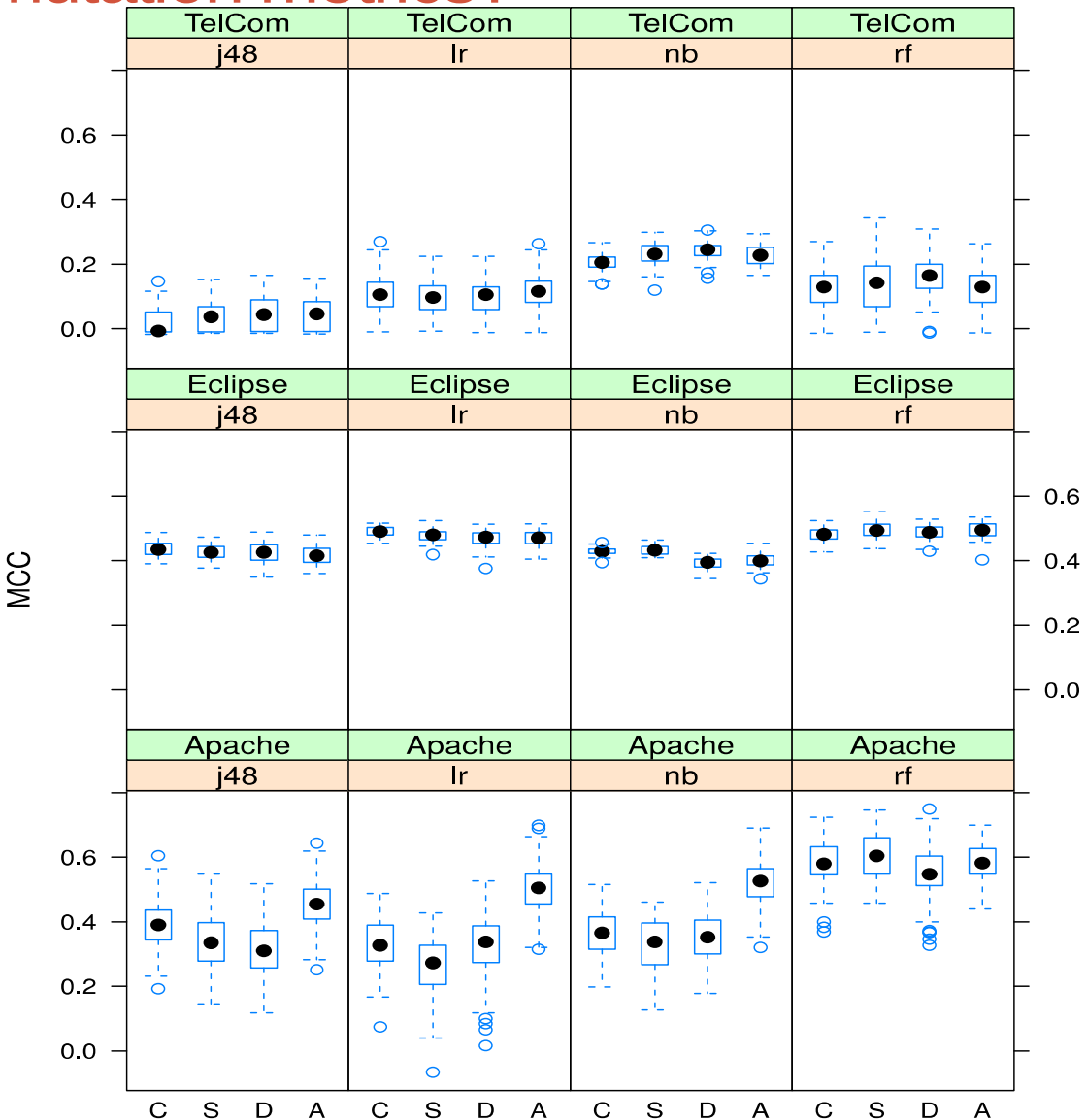
RQ2: What is the effect size of improvements in predictive model performance due to mutation metrics?

Effect of each metric to model performance: local

	Apache	Eclipse	TelCom
Naïve Bayes	MINC(0.507,5)	**MuMSC7**(0.138,7)	S-R(0.394,1)
	HVOL(0.437,4)	HVOL(0.127,4)	NSUB(0.349,1)
	CBO(0.342,2)	FOUT(0.104,500)	RFC(0.155,4)
	MuNOC2(0.317,4)	TCC(0.102,11)	**MuMSC1**(0.143,2)
	MuMSC7(0.316,49)	**MuNOM^{S}5**(0.094,3)	**MuNOC4**(0.126,45)
	MuNOM^{S}7(0.311,1)	**MuMS2**(0.086,28)	**MuNOM^{S}5**(0.121,5)
	MuNOC1(0.303,1)	MAXCC(0.082,210)	Coh(0.118,6)
	HEFF(0.297,492)	**MuMS4**(0.078,11)	HLTH(0.111,152)
	MuMSC1(0.292,21)	**MuNOM^{S}2**(0.074,20)	NCO(0.107,2)
	MuNNC7(0.287,5)	MINC(0.074,500)	NOM(0.107,5)

RQ3: What is the relative performance of static and dynamic mutation metrics?

C- code
S- static
D- dynamic
A- All



RQ3: What is the relative performance of static and dynamic mutation metrics?

(a) Static

	Apache	Eclipse	TelCom
Naïve Bayes	0.008 (-0.028)	0.450 (0.003)	<0.001 (0.027)
Logistic Regression	<0.001 (-0.055)	<0.001 (-0.010)	0.259 (-0.009)
Random Forest	0.036 (0.025)	0.010 (0.012)	0.427 (0.013)
J48	<0.001 (-0.055)	0.030 (-0.009)	0.044 (0.044)

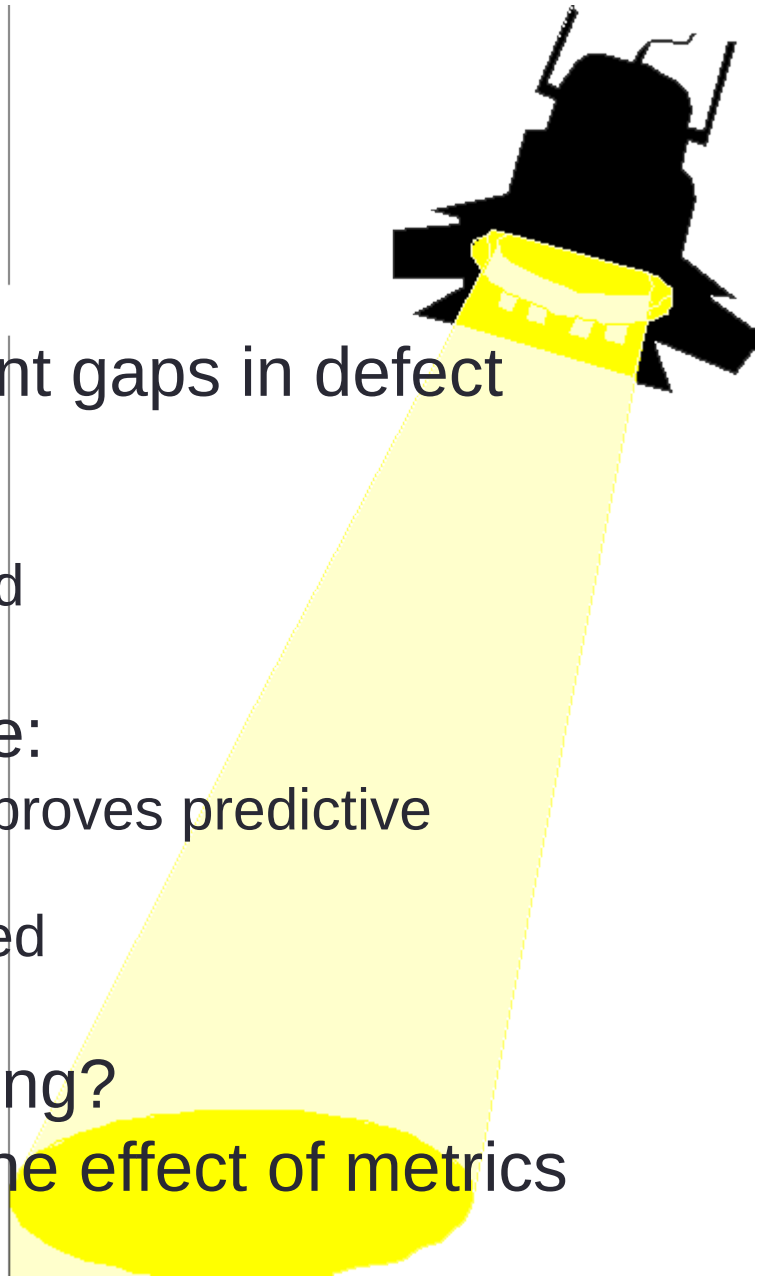
(b) Dynamic

	Apache	Eclipse	TelCom
Naïve Bayes	0.569 (-0.014)	<0.001 (-0.033)	<0.001 (0.039)
Logistic Regression	0.555 (0.017)	<0.001 (-0.019)	0.368 (-0.009)
Random Forest	0.036 (-0.052)	0.101 (0.012)	0.048 (0.036)
J48	<0.001 (-0.088)	0.429 (-0.005)	0.012 (0.050)

Conclusions

- Mutation awareness can significantly improve predictive performance and with worthwhile effect sizes.
- The effects vary from:
  - System to system
  - Algorithm to algorithm
- Combinations of metrics are generally good.
- Our results are good for:
  - Defect prediction
  - Mutation testing
- More work needed….

Questions?



Highlights:

- Our work addresses important gaps in defect prediction:
  - Testing Information rarely used
  - Dynamic Information rarely used
  - Industrial data rarely used
- Findings have lots of promise:
  - Adding mutation information improves predictive performance
  - Models are significantly improved
  - Worthwhile effect sizes occur
- Could improve mutation testing?
- A novel way of working out the effect of metrics