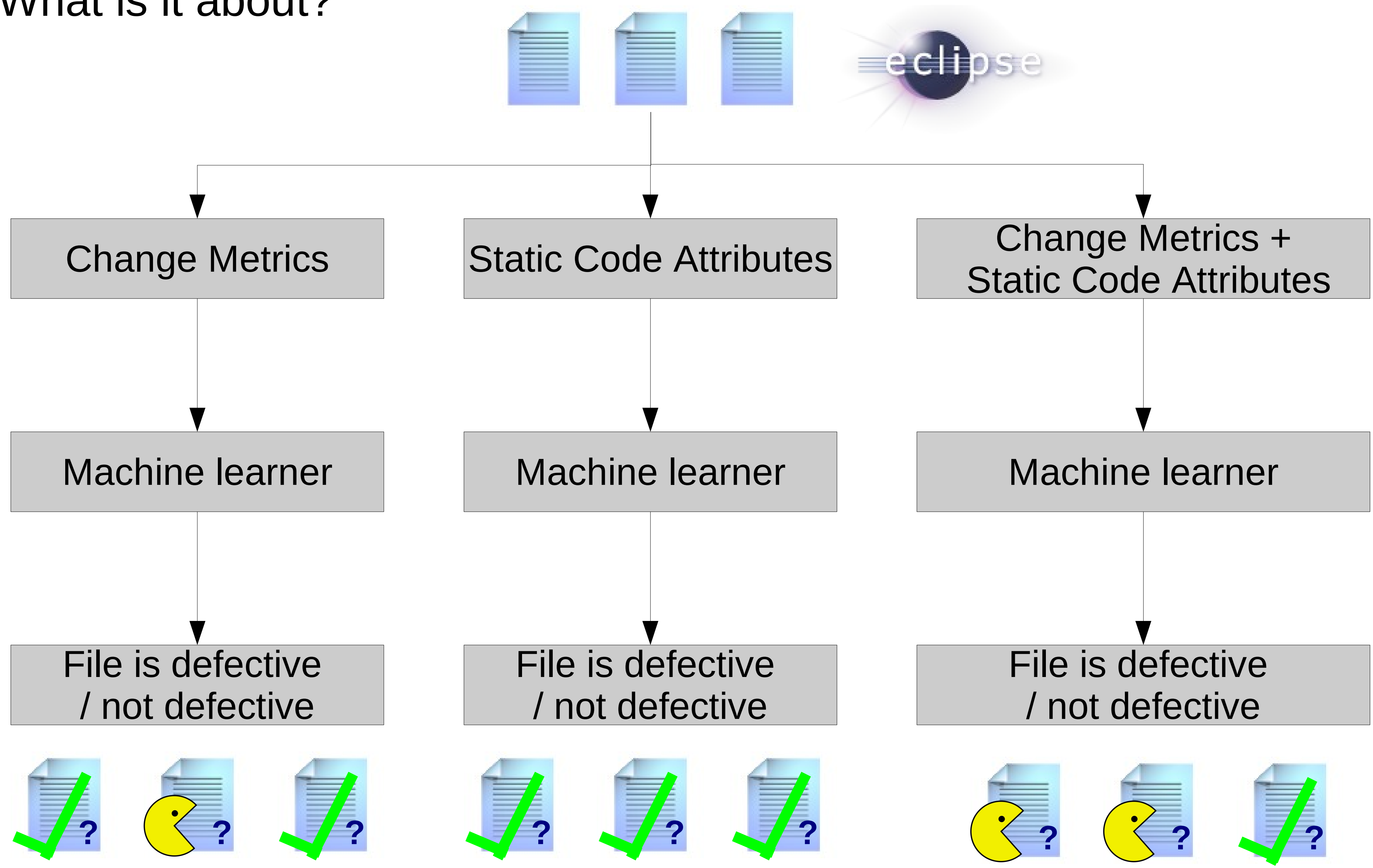
A Comparative Analysis of the Efficiency of Change Metrics and Static Code Attributes for Defect Prediction

Raimund Moser, Witold Pedrycz, Giancarlo Succi
ICSE 08

Andres Bühlmann, April 2009

What is it about?



Static Code Attributes

- Describe how a file is at the moment

Example:

Methods: Number of method calls, Nested bock depth, ...
Classes: Number of methods, ...
Files: Number of interfaces, Total lines of code, ...

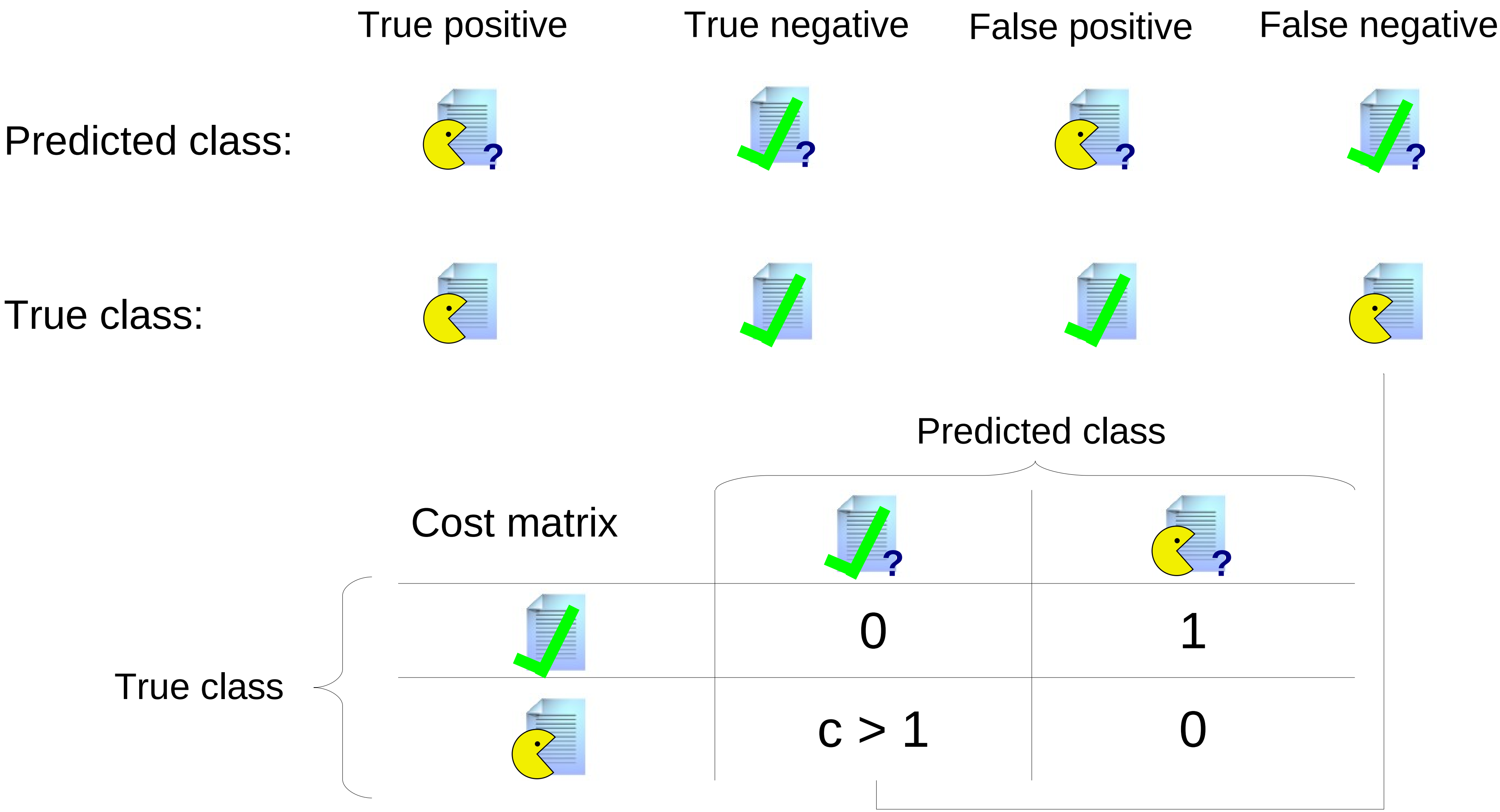
Change Metrics

- Describe how a file changed in the past

Example:

REVISIONS, REFACTORINGS, BUGFIXES, LOC_ADDED, ...

The cost of making wrong decisions



Assessing classification accuracy

		
	True Negative (TN)	False Positive (FP)
	False Negative (FN)	True Positive (TP)

True positive rate (recall): $$\frac{TP}{TP+FN}$$

False positive rate: $$\frac{FP}{FP+TN}$$

Percentage of correctly predicted files: $$\frac{TP+TN}{TP+TN+FP+FN}$$

Experimental set-up

31 Static Code Attributes

18 Change Metrics

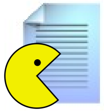
31 Static Code Attributes
+ 18 Change metrics

Release	#Files	Defective (Post-release)
2.0	3851	31%
2.1	5341	23%
3.0	5347	32%

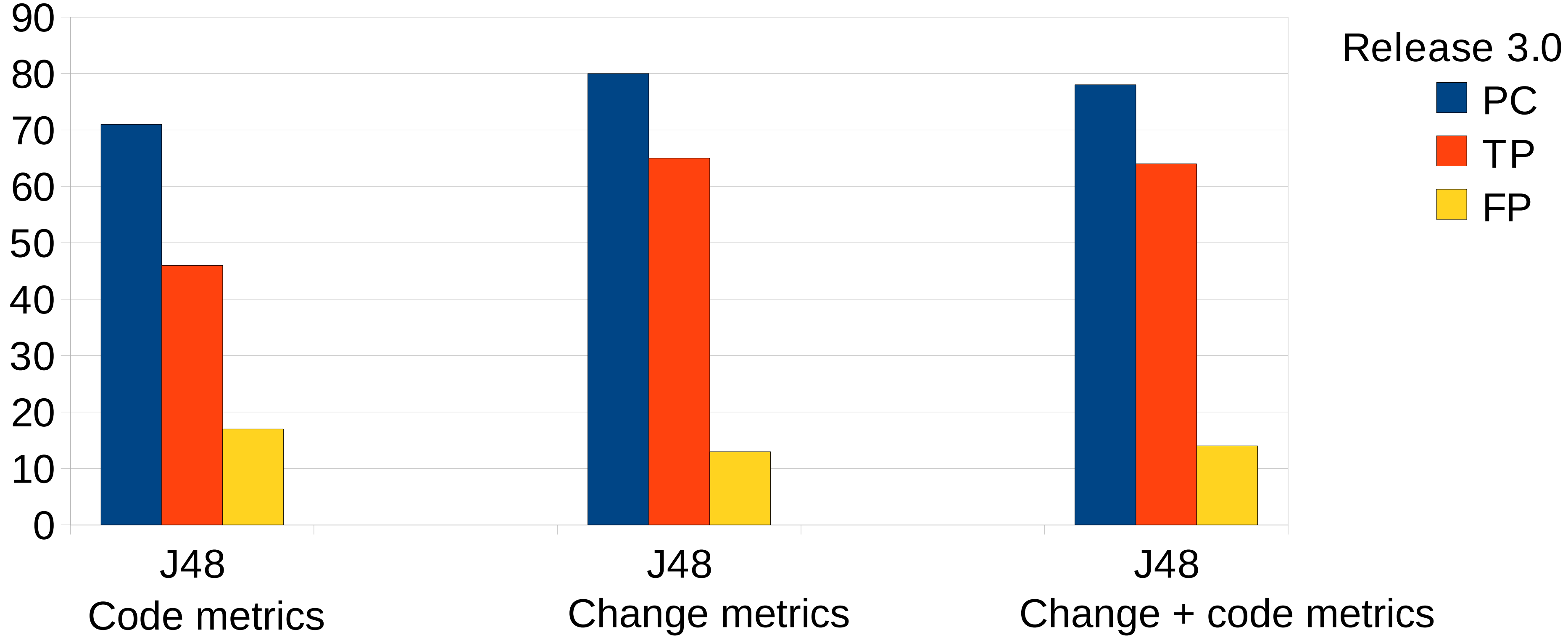
Machine learners
- Naive Bayes
- Logistic Regression
- Decision Tree (J48)

H_0: Code metrics have the same prediction accuracy as change metrics for (cost-sensitive) defect prediction.

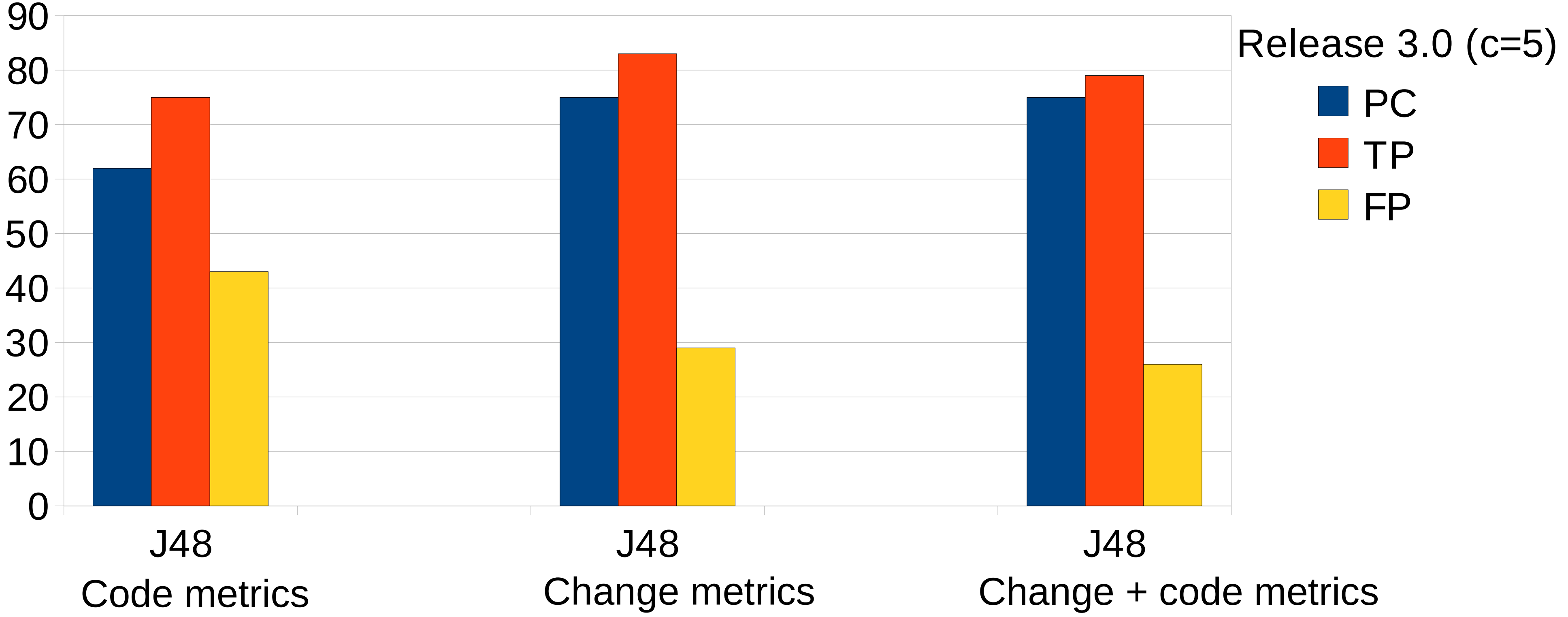
The cost of making wrong decisions

Cost matrix	Predicted class: 	Predicted class: 
True class: 	0	1
True class: 	c > 1	0

Results (cost insensitive)



Results (cost sensitive)



Their results and conclusions

~~H_0: Code metrics have the same prediction accuracy as change metrics for (cost-sensitive) defect prediction.~~

- Change metrics have better prediction accuracy than code metrics
- Use cost sensitive classification to improve recall

- May not generalize
- Not sure whether or not used right metrics
- Doesn't perform well for an iterative approach
- Potential errors in code and change metrics

Discussion

What is nice

- Different machine learners
- 3 Releases
- 10 fold cross validation
- Significance analysis
- Cost sensitive analysis

What I didn't like

- No results for iterative defect prediction
- Not clear how change metrics were extracted
- Change metrics not available

Further Thoughts

The machine learners tell us which metrics are good indicators for defects

Example:

Files involved in a lot of bug fixing activities are most likely to be defective

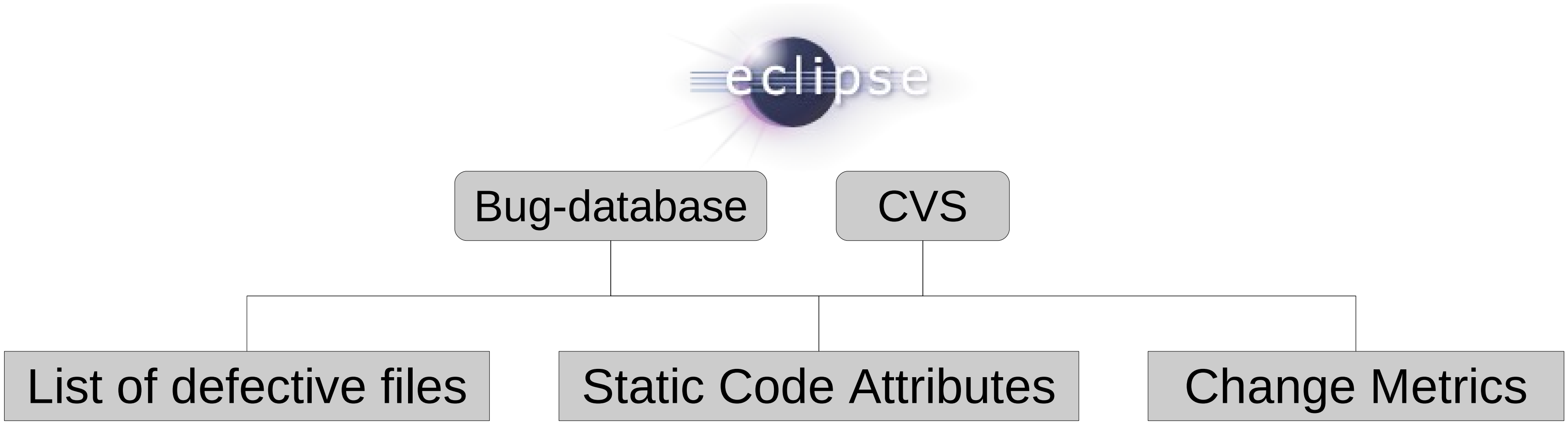
Can we conclude something from this statement?

At least we can ask further questions:

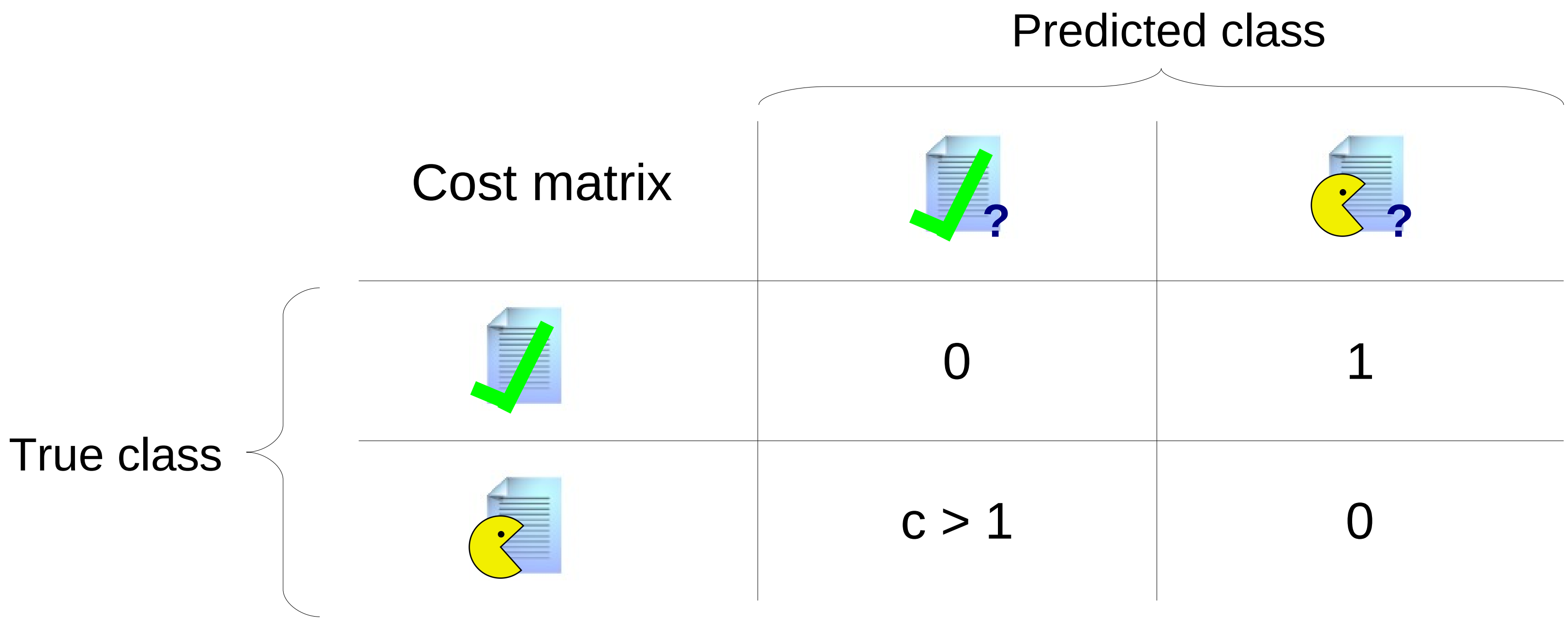
- Are this file new?
- Do this files contain a lot of complex code?
- Are bugs fixed by the initial author?
- Is our documentation and/or comments insufficent?

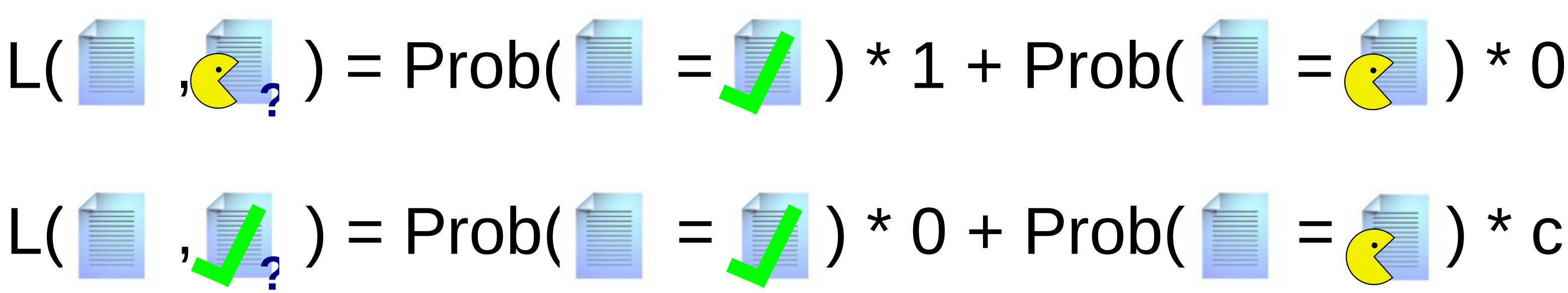
-> We should investigate how statistical analysis of detected bugs could be used to improve our process and design decisions

How to calculate the metrics?

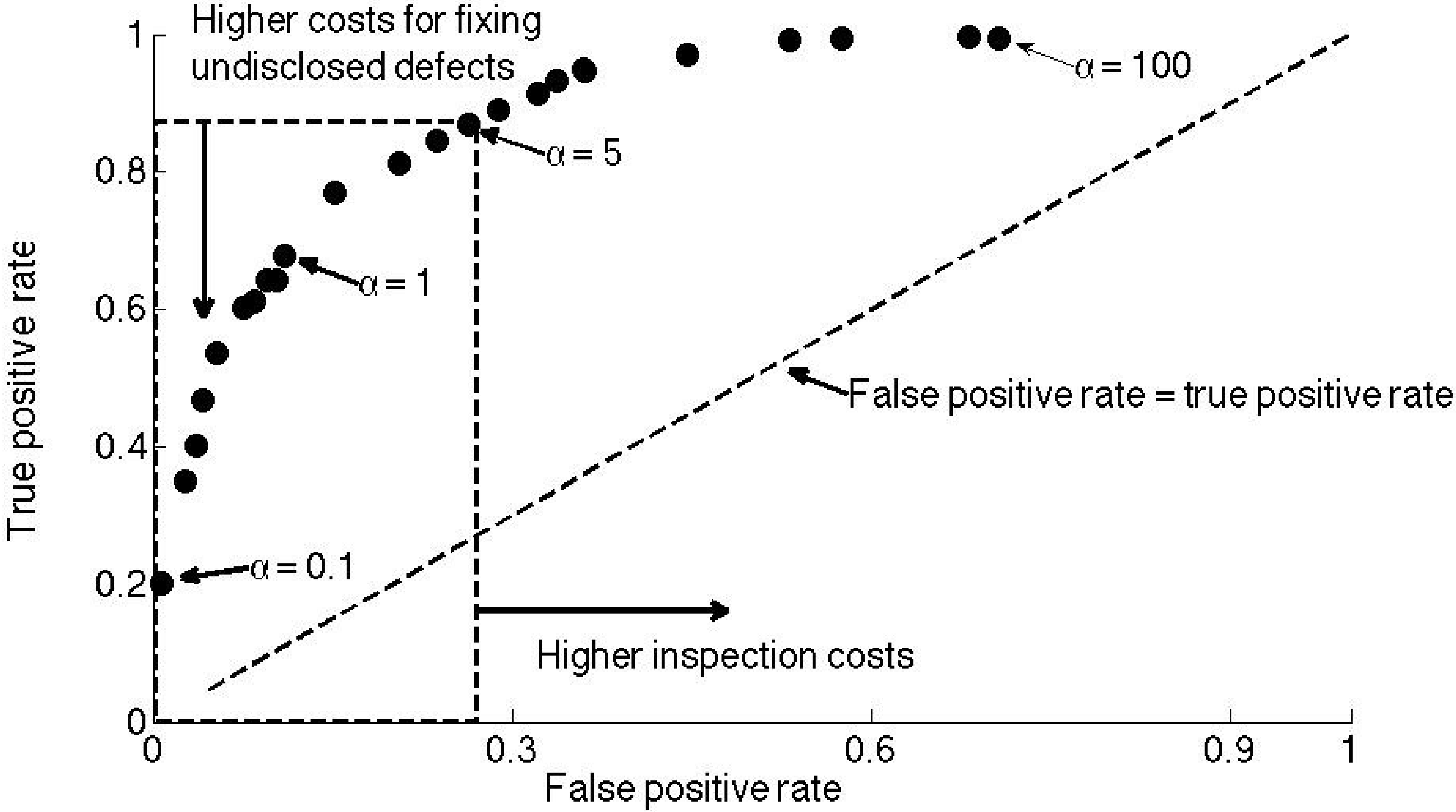


The cost of making wrong decisions





How does the cost affect the prediction?



Which metrics have the most predictive power?

Powerful defect indicators:

- High number of revisions
- High number of bug-fixing activities
- Small sized CVS commits
- Small number of refactorings

See Zimmermann p 2

Static Code Attributes

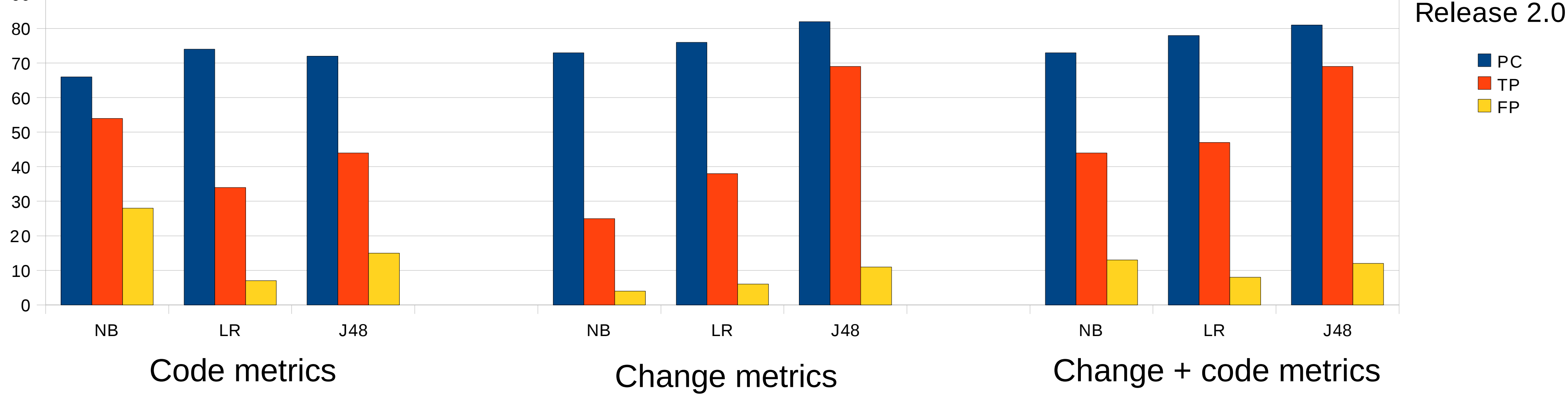
	Metric name	**Definition**
methods	FOUT	Number of method calls (fan out)
	MLOC	Method lines of code
	NBD	Nested block depth
	PAR	Number of parameters
	VG	McCabe cyclomatic complexity
classes	NOF	Number of fields
	NOM	Number of methods
	NSF	Number of static fields
	NSM	Number of static methods
files	ACD	Number anonymous type declarations
	NOI	Number of interfaces
	NOT	Number of classes
	TLOC	Total lines of code

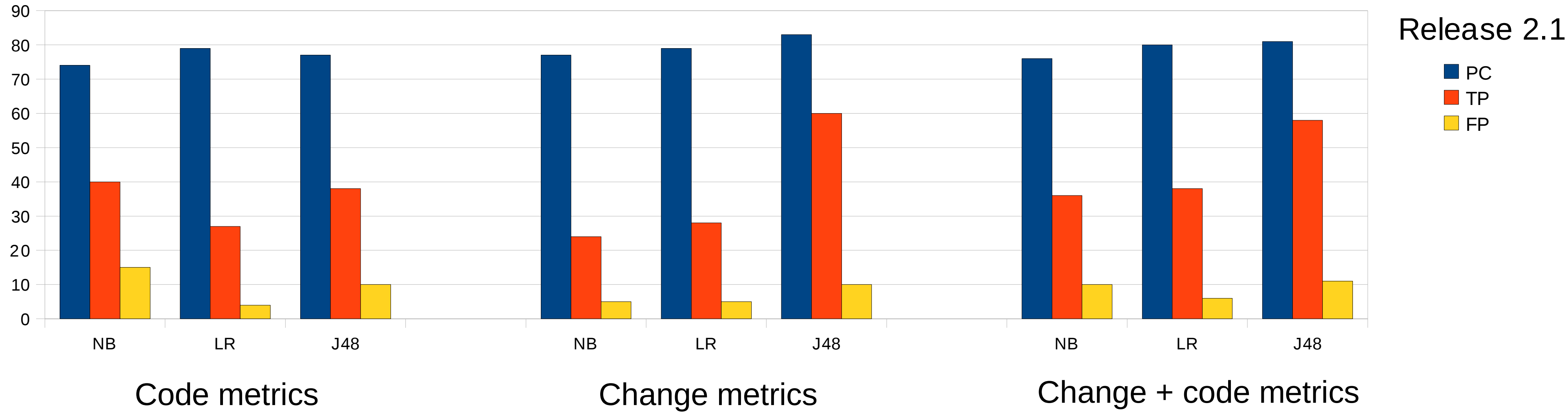
See p 185

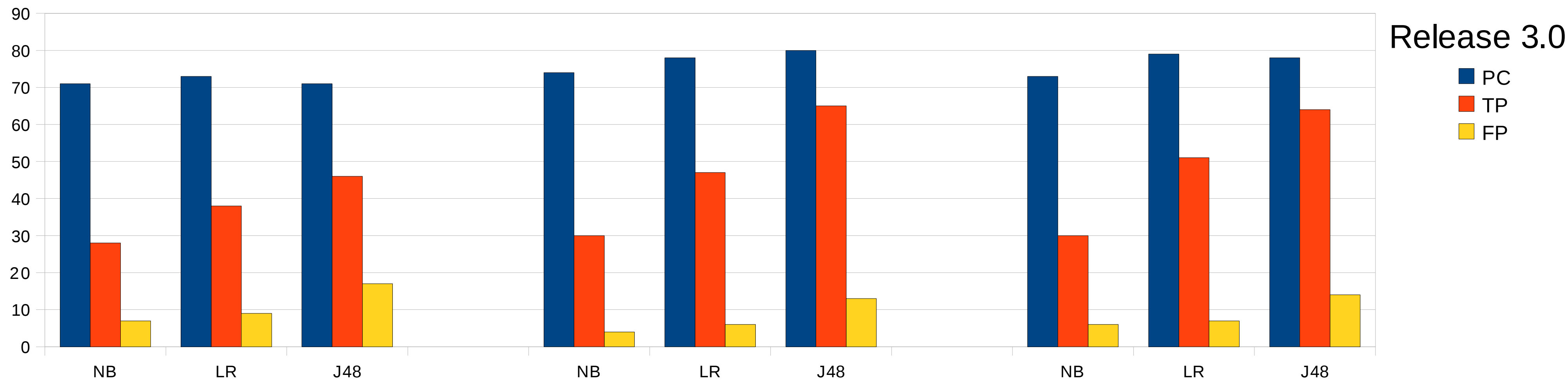
Change Metrics

Metric name	Definition
REVISIONS	Number of revisions of a file
REFACTORINGS	Number of times a file has been refactored
BUGFIXES	Number of times a file has been involved in bug-fixing
AUTHORS	Number of distinct authors
LOC_ADDED	Sum over all revisions of the lines of code added to a file
LOC_DELETED	
CODECHURN	Sum of added lines of code – deleted lines of code over all revisions
MAX_CHANGESET	Maximum number of files committed together
AGE	Age of a file in weeks counted counted backwards from the release

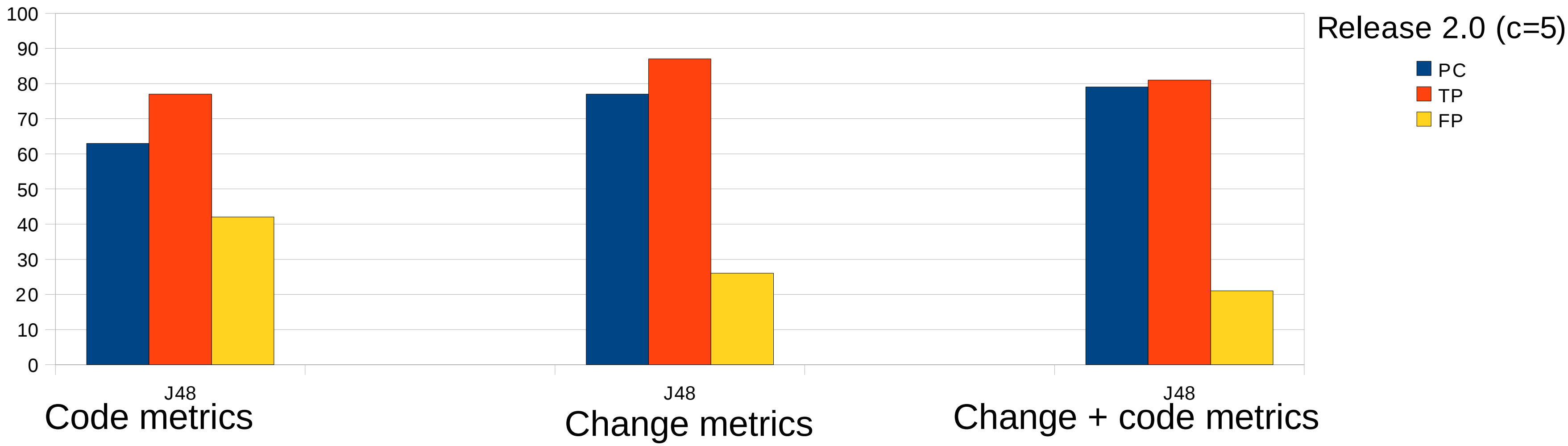
See p 186

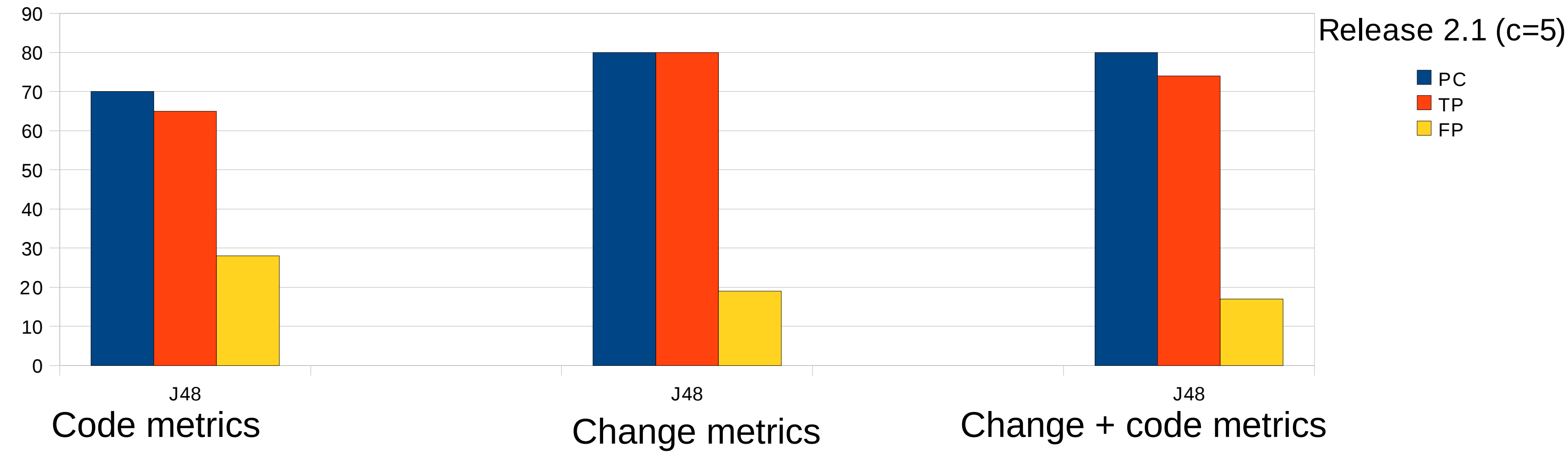

Release 2.0
PC
TP
FP
NB
LR
J48
Code metrics
Change metrics
Change + code metrics

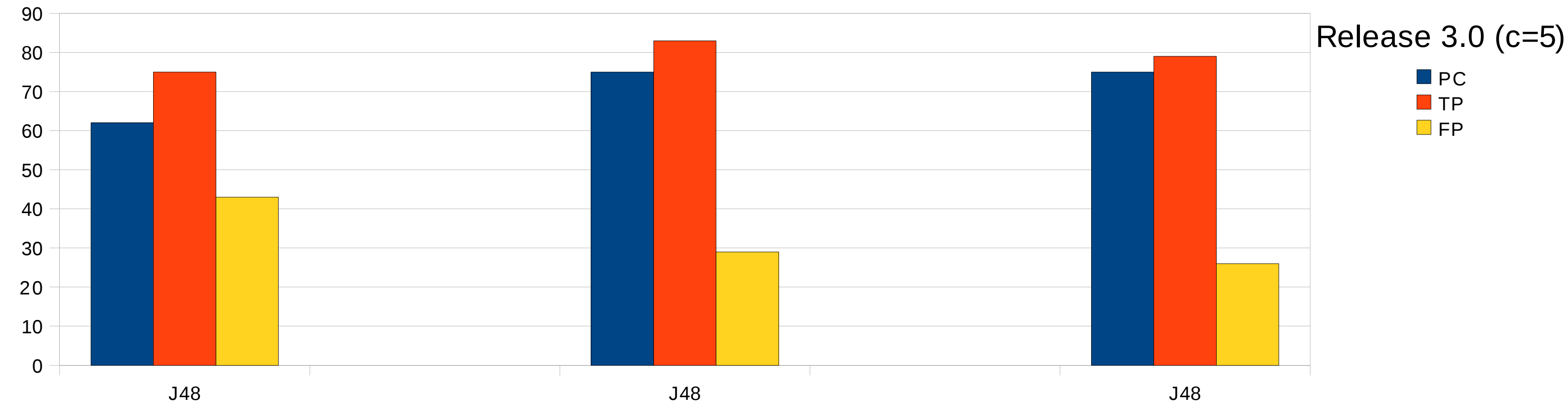

Release 2.1
PC
TP
FP
NB
LR
J48
Code metrics
Change metrics
Change + code metrics


Release 3.0
PC
TP
FP
NB
LR
J48

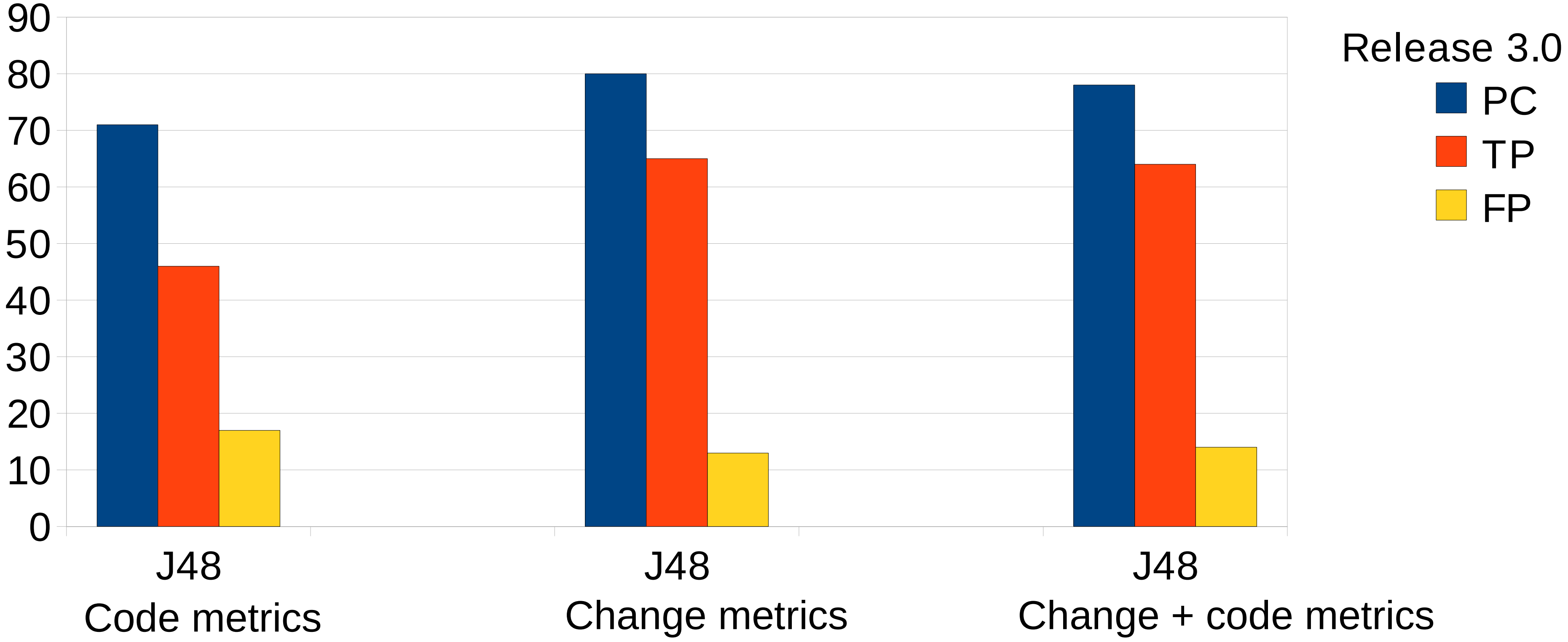
See p 188


Release 2.0 (c=5)
PC
TP
FP
100
90
80
70
60
50
40
30
20
10
0
J48
J48
J48
Code metrics
Change metrics
Change + code metrics


Release 2.1 (c=5)
PC
TP
FP
90
80
70
60
50
40
30
20
10
0
J48
J48
J48
Code metrics
Change metrics
Change + code metrics


Release 3.0 (c=5)
PC
TP
FP
90
80
70
60
50
40
30
20
10
0
J48
J48
J48

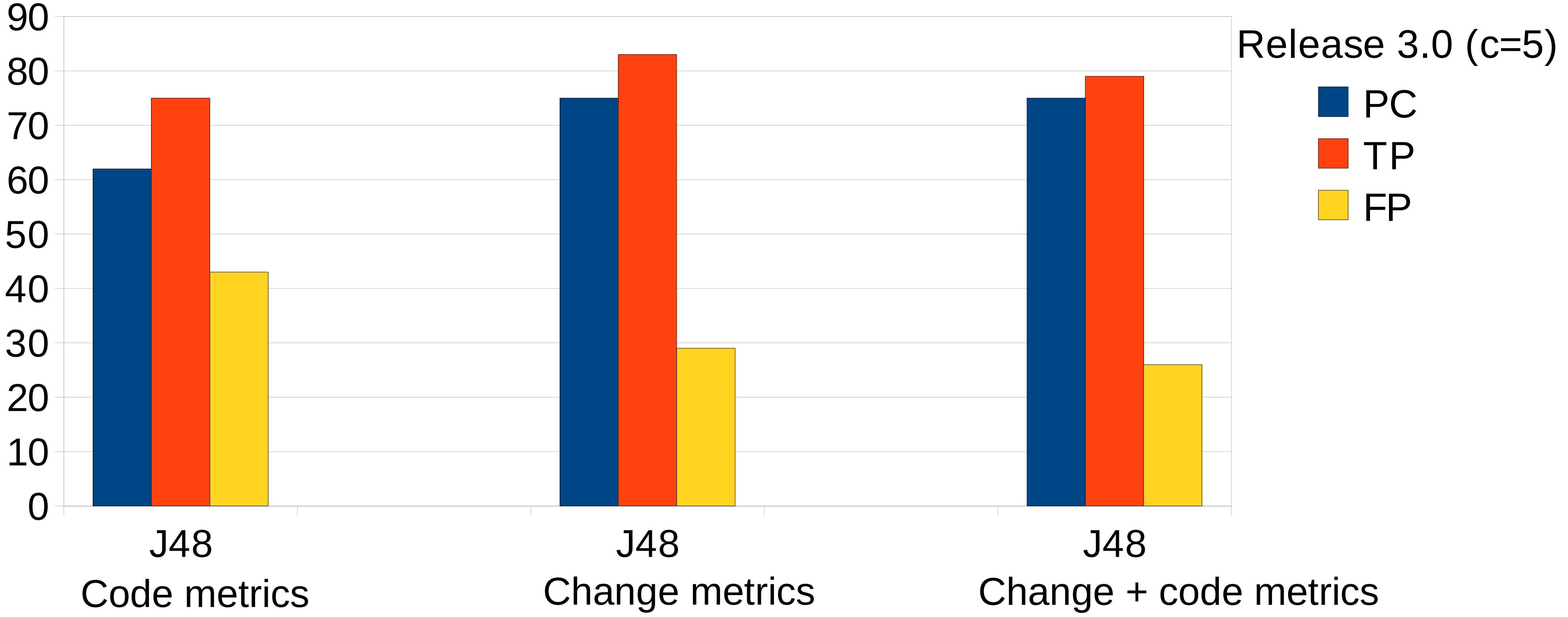
Results (cost insensitive)



Absolute numbers for J48 using change metrics:

Files:	5347	Correctly classified:	4277
Defective:	1725	Correctly classified defective:	1121
Defect free:	3622	Incorrectly classified defective:	471

Results (cost sensitive)



Absolute numbers for J48 using change metrics:

Files:	5347	Correctly classified:	4010
Defective:	1725	Correctly classified defective:	1432
Defect free:	3622	Incorrectly classified defective:	1050