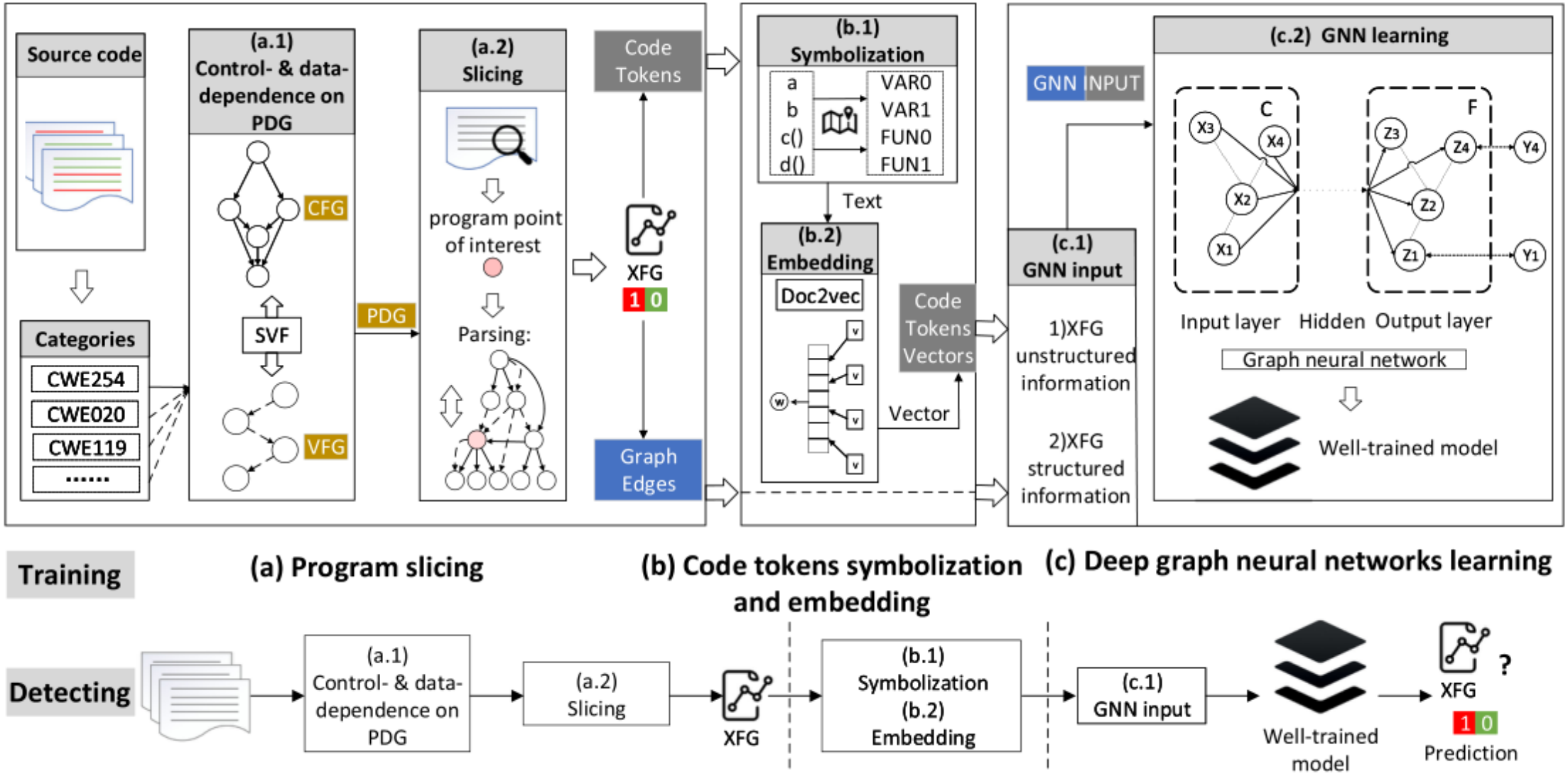
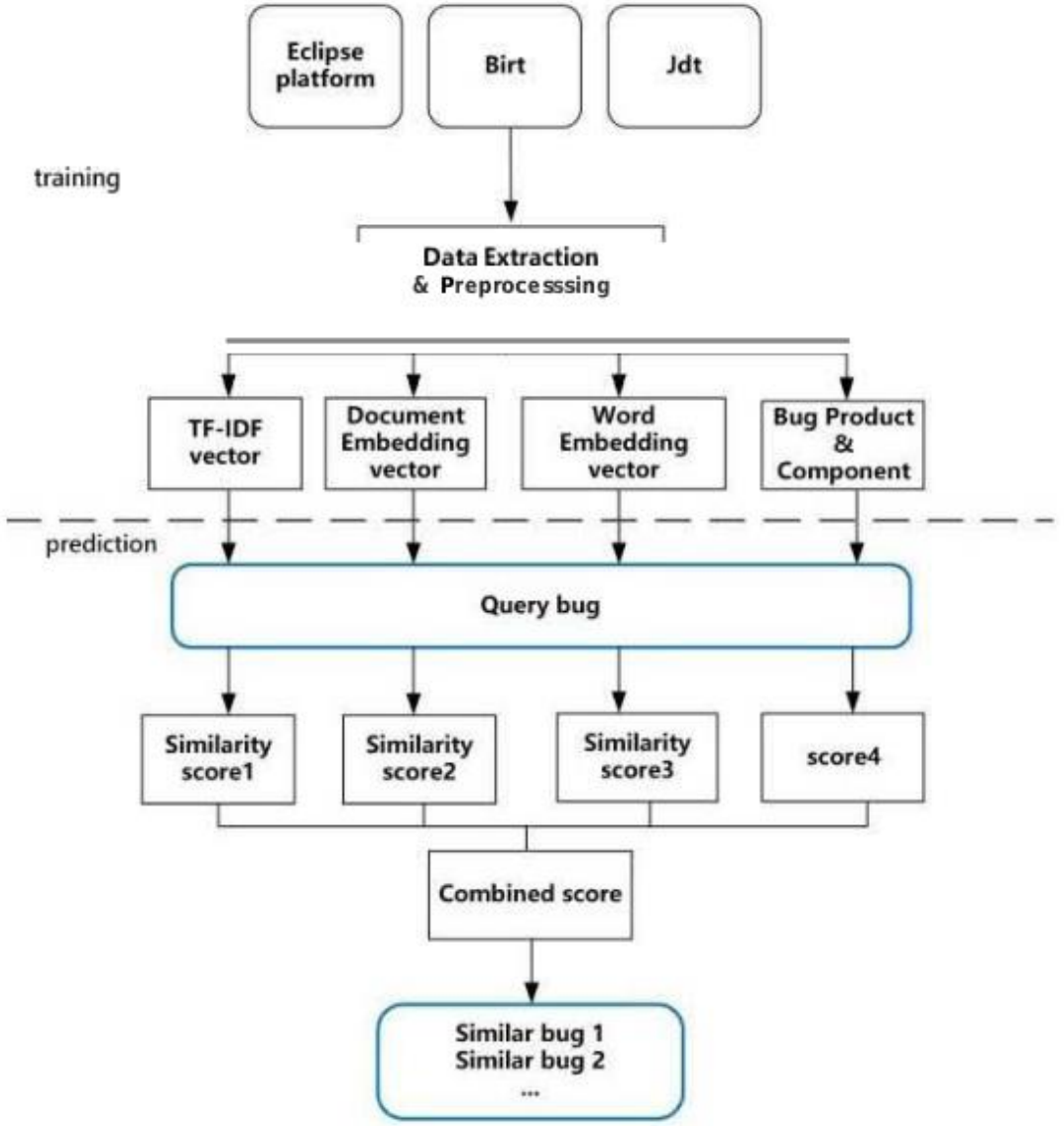
5 Papers

Title	Publication source	Year
DeepWukong: Statically Detecting Software Vulnerabilities Using Deep Graph Neural Network	TOSEM	2021
Using Document Embedding Techniques for Similar Bug Reports Recommendation	ICSE	2018
Automatically Predicting Bug Severity Early in the Development Process	ICSE	2020
A Study of Bug Resolution Characteristics in Popular Programming Languages	TSE	2020
Toward a Smell-Aware Bug Prediction Model	TSE	2019

DeepWukong: Statically Detecting Software Vulnerabilities Using Deep Graph Neural Network, TOSEM (2021)



Using Document Embedding Techniques for Similar Bug Reports Recommendation，ICSE (2018)



Automatically Predicting Bug Severity Early in the Development Process，ICSE (2020)

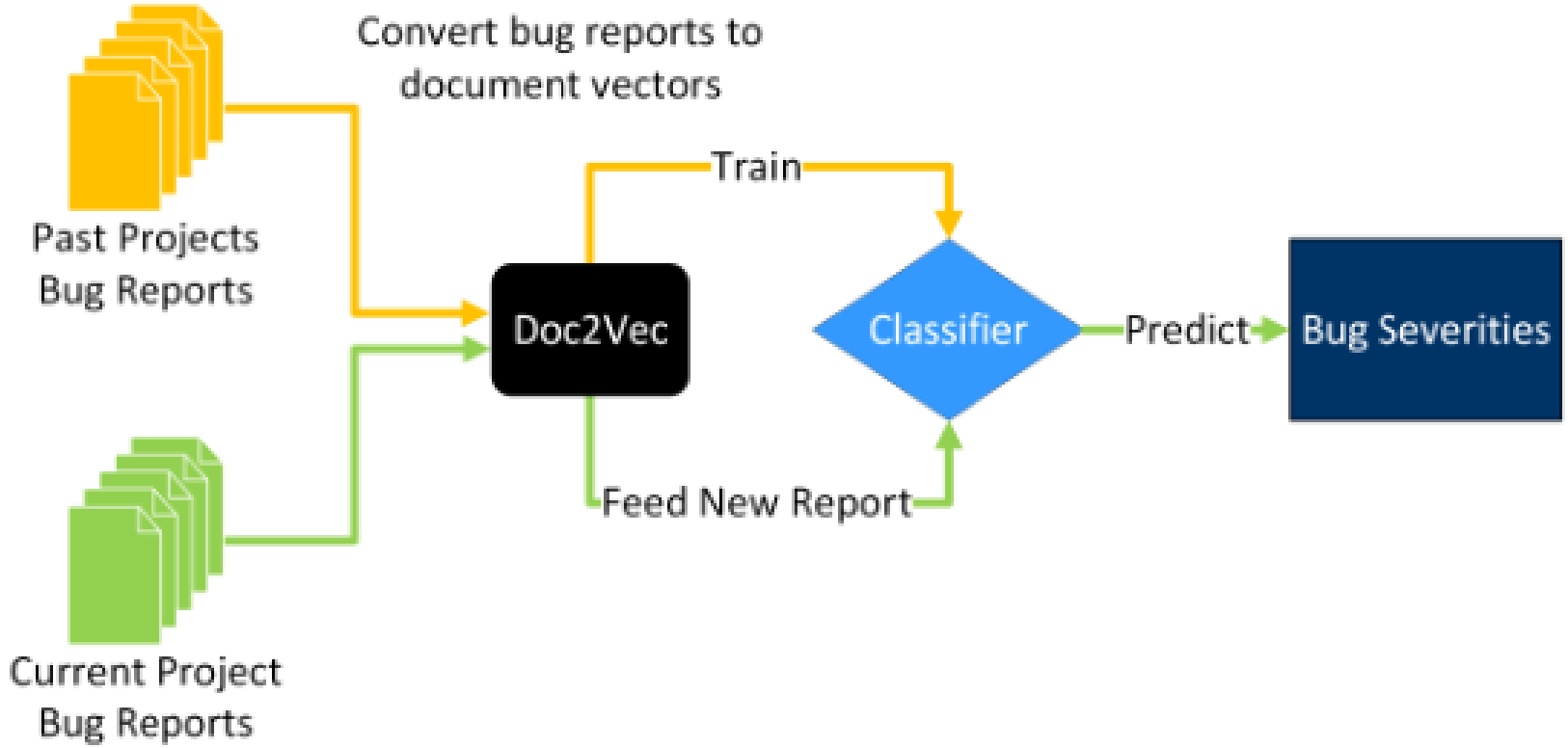


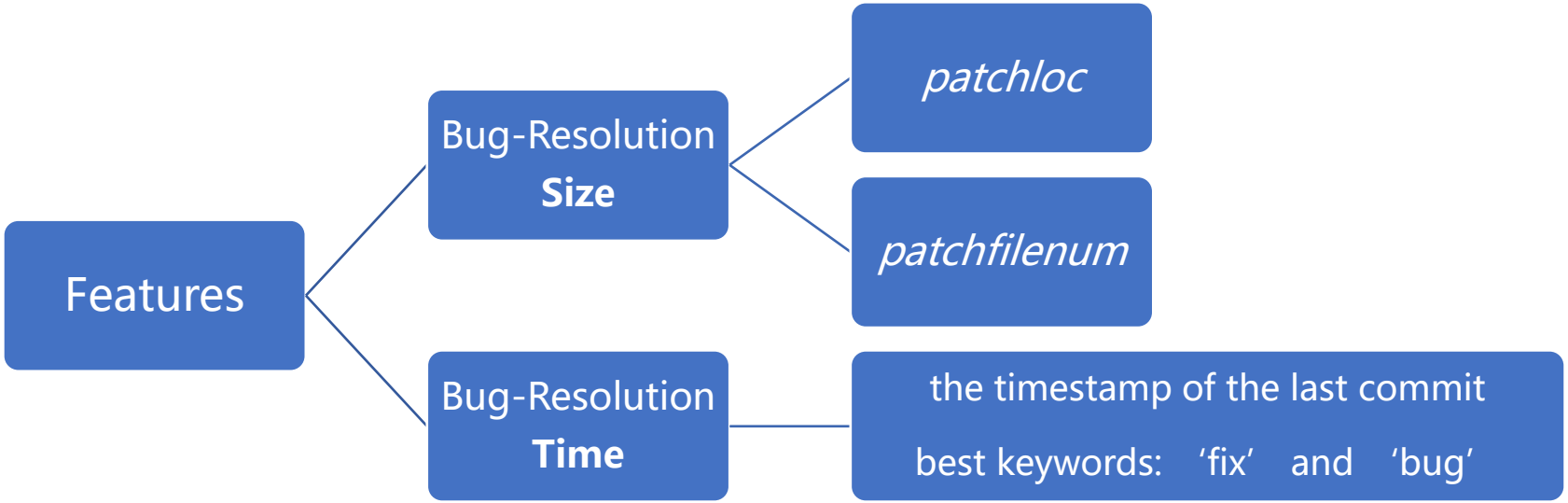
Figure 1: Bug report severity prediction process

A Study of Bug Resolution Characteristics in Popular Programming Languages，TSE (2020)

A systematic and extensive study on bug-resolution effort among different programming languages and categories.

This paper performs a comprehensive study of **10 popular languages:**

C, C#, C++, Go, Java, JavaScript, ObjectiveC, PHP, Python, and Ruby

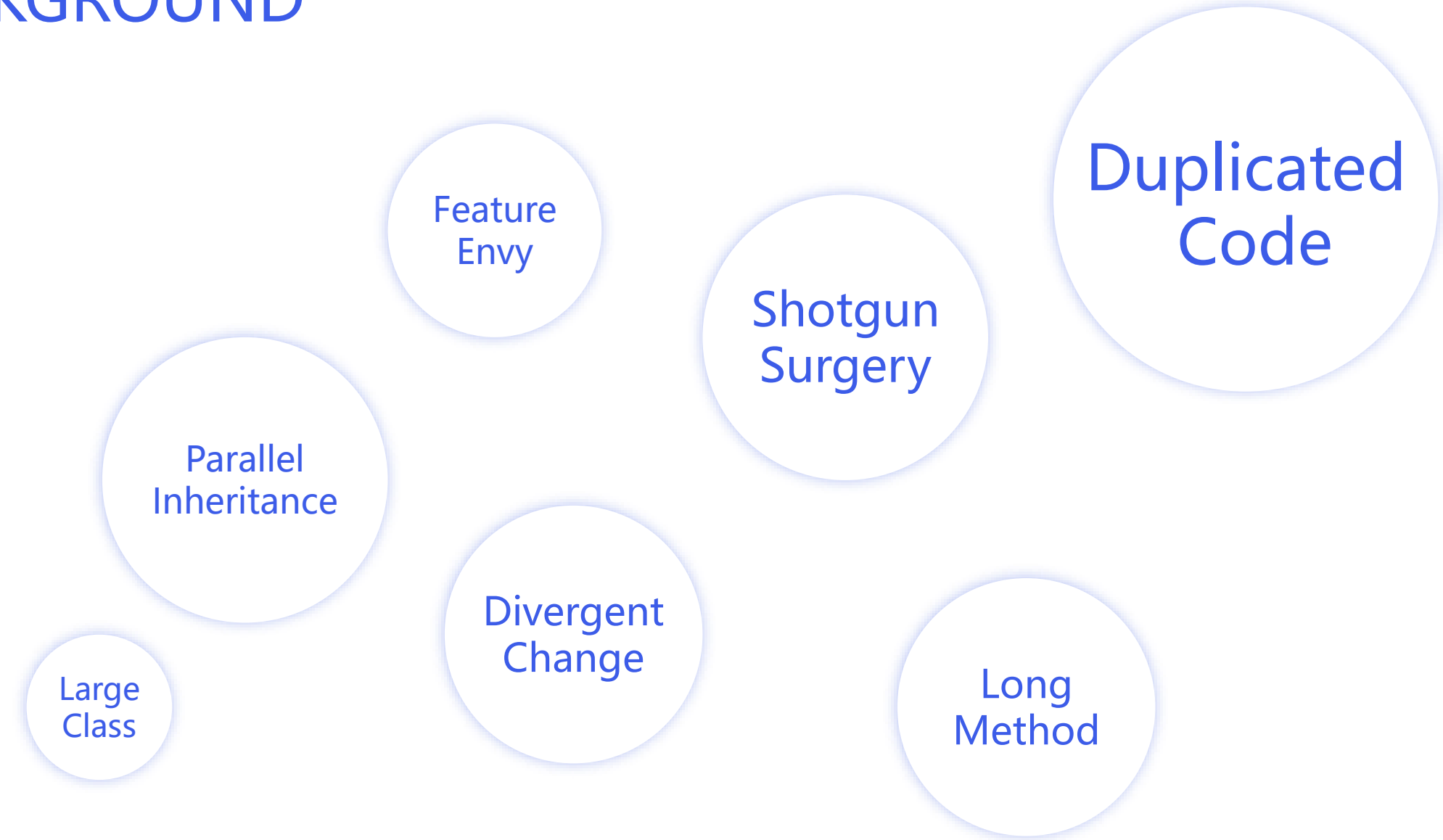


Toward a Smell-Aware Bug Prediction Model

汇报人：王昭丹

CONTENTS

BACKGROUND


Duplicated Code
Feature Envy
Shotgun Surgery
Parallel Inheritance
Divergent Change
Large Class
Long Method

BACKGROUND

1. Hall et al. found that some code scents were only associated with the presence of errors under certain circumstances, but these scents had little effect on errors.

2. Taba et al. report on the use of historical measures (called anti-pattern measures) for class computations affected by design defects as an additional source of information for predicting defects. They found that such an indicator could improve the performance of a mis-prediction model by up to 12.5 percent.

In contrast to the above work, this paper recommend using a measure of code smell intensity rather than calculating a historical measure of classes/methods affected by smell.

PROPOSED MODEL

The **intensity index** is an estimation of the severity of a code smell, and its value is defined as a real number in the range [1,10].

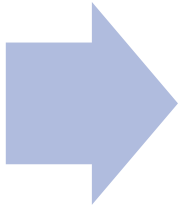
TABLE 1: Code Smell Detection Strategies (the complete names of the metrics are given in Table 2)

Code Smells	Detection Strategies: LABEL(n) → LABEL has value n for that smell
God Class	LOCNAMM ≥ HIGH(176) ∧ WMCNAMM ≥ MEAN(22) ∧ NOMNAMM ≥ HIGH(18) ∧ TCC ≤ LOW(0.33) ∧ ATFD ≥ MEAN(6)
Data Class	WMCNAMM ≤ LOW(14) ∧ WOC ≤ LOW(0.33) ∧ NOAM ≥ MEAN(4) ∧ NOPA ≥ MEAN(3)
Brain Method	(LOC ≥ HIGH(33) ∧ CYCLO ≥ HIGH(7) ∧ MAXNESTING ≥ HIGH(6)) ∨ (NOLV ≥ MEAN(6) ∧ ATLD ≥ MEAN(5))
Shotgun Surgery	CC ≥ HIGH(5) ∧ CM ≥ HIGH(6) ∧ FANOUT ≥ LOW(3)
Dispersed Coupling	CINT ≥ HIGH(8) ∧ CDISP ≥ HIGH(0.66)
Message Chains	MaMCL ≥ MEAN(3) ∨ (NMCS ≥ MEAN(3) ∧ MeMCL ≥ LOW(2))

$$z = [\frac{x - min(x)}{max(x) - min(x)}] \cdot 10$$

Two steps to compute the intensity of code smells

In the first step the tool aims at detecting code smells in the system given as input, relying on the detection strategies reported in Table 1.



"exceeding amount"

min-max normalization process

PROPOSED MODEL

TABLE 2: Metrics used for Code Smells Detection

Short Name	Long Name	Definition
ATFD	Access To Foreign Data	The number of attributes from unrelated classes belonging to the system, accessed directly or by invoking accessor methods.
ATLD	Access To Local Data	The number of attributes declared by the current classes accessed by the measured method directly or by invoking accessor methods.
CC	Changing Classes	The number of classes in which the methods that call the measured method are defined in.
CDISP	Coupling Dispersion	The number of classes in which the operations called from the measured operation are defined, divided by CINT.
CINT	Coupling Intensity	The number of distinct operations called by the measured operation.
CM	Changing Methods	The number of distinct methods that call the meas-ured method.
CYCLO	McCabe Cyclomatic Complexity	The maximum number of linearly independent paths in a method. A path is linear if there is no branch in the execution flow of the corresponding code.
FANOUT		Number of called classes.
LOC	Lines Of Code	The number of lines of code of an operation or of a class, including blank lines and comments.
LOCNAMM	Lines of Code Without Accessor or Mutator Methods	The number of lines of code of a class, including blank lines and comments and excluding accessor and mutator methods and corresponding comments.
MaMCL	Maximum Message Chain Length	The maximum length of chained calls in a method.
MAXNESTING	Maximum Nesting Level	The maximum nesting level of control structures within an operation.
MeMCL	Mean Message Chain Length	The average length of chained calls in a method.
NMCS	Number of Message Chain Statements	The number of different chained calls in a method.
NOAM	Number Of Accessor Methods	The number of accessor (getter and setter) methods of a class.
NOLV	Number Of Local Variables	Number of local variables declared in a method. The method's parameters are consid-ered local variables.
NOMNAMM	Number of Not Accessor or Mutator Methods	The number of methods defined locally in a class, counting public as well as private methods, exclud-ing accessor or mutator methods.
NOPA	Number Of Public Attributes	The number of public attributes of a class.
TCC	Tight Class Cohesion	The normalized ratio between the number of methods directly connected with other methods through an instance variable and the total number of possible connections between methods. A direct connection between two methods exists if both access the same instance variable directly or indirectly through a method call. TCC takes its value in the range [0,1].
WMCNAMM	Weighted Methods Count of Not Ac-cessor or Mutator Methods	The sum of complexity of the methods that are defined in the class, and are not accessor or mutator methods. We compute the complexity with the Cyclomatic Complexity metric (CYCLO).
WOC	Weight Of Class	The number of "functional" (*i.e.*, non-abstract, non-accessor, non-mutator) public methods divided by the total number of public members.

PROPOSED MODEL

Firstly split the training set by considering **smelly** and **non-smelly** classes

Assign to smelly classes an intensity index according to the evaluation performed by JCodeOdor, while set the intensity of **non-smelly** classes to **0**

Finally, build a **prediction model** using as predictors the intensity index and a set of other product/process metrics.

DESIGN OF THE CASE STUDY

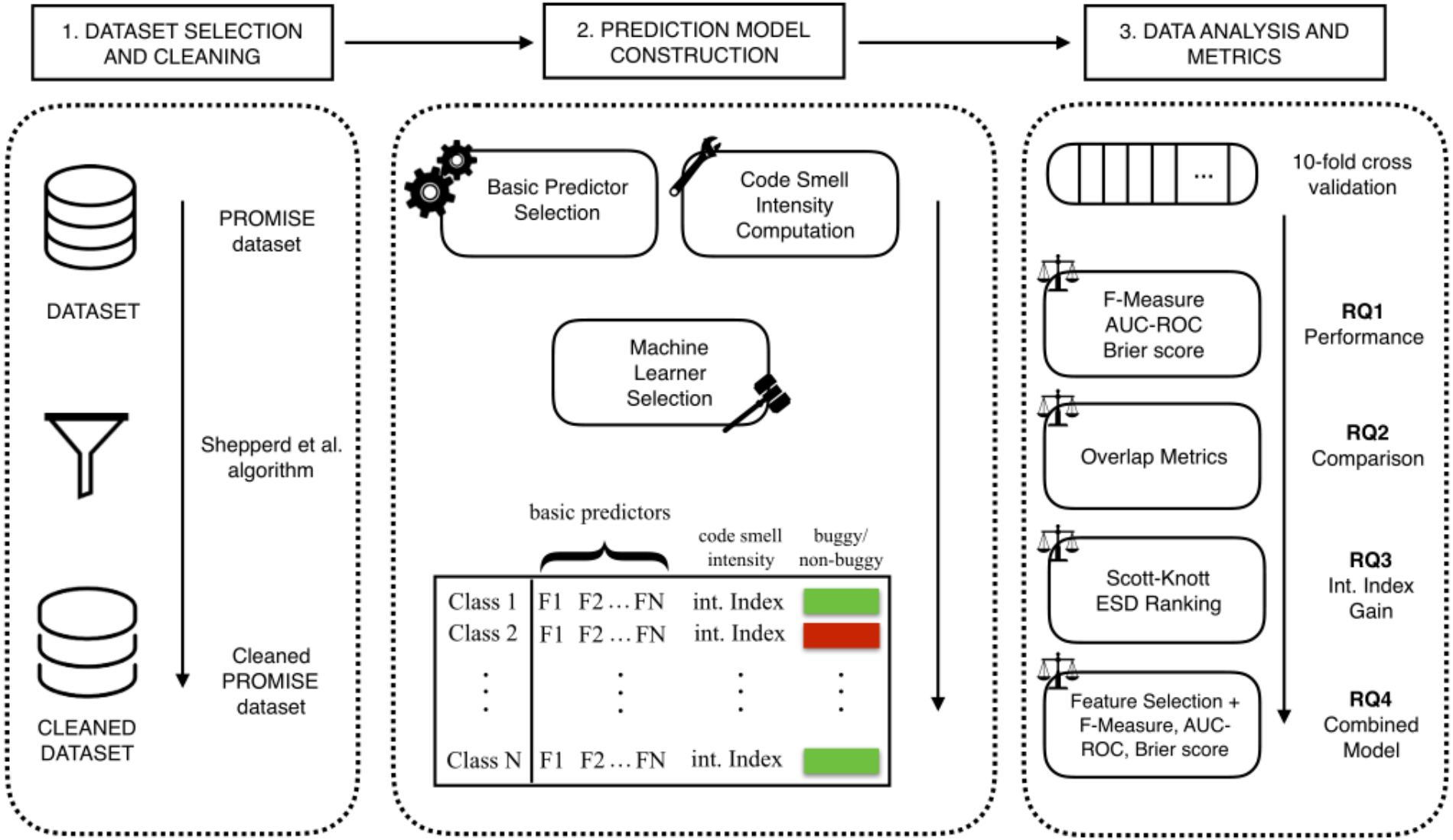


Fig. 1: Overview of the Empirical Study Design.

DESIGN OF THE CASE STUDY

Dataset Selection and Cleaning

Data cleaning

- following the algorithm proposed by Shepperd et al.
- includes a list of 13 corrections aimed at removing identical features, features with conflicting values or missing values etc..

1

TABLE 6: Software Projects in Our Dataset

System	Releases	Classes	KLOCs	% Buggy Cl.	% Smelly Cl.	EPV
Apache Ant	5	83-813	20-204	68-72	11-16	12-15
Apache Camel	4	315-571	70-108	30-38	9-14	16-21
Apache Forrest	3	112-628	18-193	37-39	11-13	14-17
Apache Ivy	1	349	58	29	12	14
JEdit	5	228-520	39-166	36-43	14-22	11-12
Apache Velocity	3	229-341	57-73	15-23	7-13	16-18
Apache Tomcat	1	858	301	6	4	15
Apache Lucene	3	338-2,246	103-466	59-63	10-22	11-18
Apache Pbeans	2	121-509	13-55	29-33	21-25	14-16
Apache POI	4	129-278	68-124	62-68	15-19	15-22
Apache Synapse	3	249-317	117-136	17-26	13-17	14-19

EPV (the number of events per variables)

- the ratio between the number of occurrences of the least frequently occurring class of the dependent variable and the number of independent variables used to train the model.
- datasets having **EPVs lower than 10** are particularly susceptible to unstable results.

DESIGN OF THE CASE STUDY —— Prediction Model Construction

Basic Predictors

① a bug prediction model composed of structural predictors, and in particular the 20 quality metrics exploited by Jureczko et al. .

② three baseline prediction models based on process metrics

③ experimented

- the baseline models described above
- the same baseline models where the intensity index is plugged as additional metric.

three baseline prediction models

- the Basic Code Change Model (BCCM)
- the DM Model
- the Developer Changes Based Model (DCBM)

Code Smell Detection

- relied on the detection performed by JCodeOdor
- computes the value of the intensity index on the detected code smells
- discard the false positive instances from the set of candidate code smells given by the detection tool
- trained the prediction model taking into account the actually smelly classes only.

Machine Learning Technique

- choose a machine learning classifier to use
- Multilayer Perceptron, ADTree, Naive Bayes, Logistic Regression, Decision Table Majority, **Simple Logistic**

DESIGN OF THE CASE STUDY —— Data Analysis and Metrics

Validation Methodology

- **10-fold cross-validation strategy**

RQ1 - The contribution of the Intensity Index

Metrics:

- **Precision** and **Recall**

$$precision = \frac{TP}{TP + FP} \qquad recall = \frac{TP}{TP + TN}$$

- **F-Measure** —— harmonic mean of precision and recall

$$F\text{-}Measure = 2 * \frac{precision * recall}{precision + recall}$$

Metrics:

- Area Under the Curve (**AUC**)

The closer the AUC to 1, the higher the ability of the classifier to discriminate classes affected and not by a bug.

- **Brier score**

Low Brier scores indicate good classifier performances, while high scores indicate low performances.

$$Brier\text{-}score = \frac{1}{N} \sum_{i=1}^{N} (p_c - o_c)$$

p_c: probability predicted by the model on a class **c**

o_c: actual outcome for class **c**

DESIGN OF THE CASE STUDY —— Data Analysis and Metrics

RQ2 - Comparison between Intensity Index and Antipattern Metrics

- The antipattern metrics defined by Taba et al.
- Plugged the antipattern metrics into the product- and process-based baseline models.
- Compared the performances of the resulting models with the ones achieved by the model built using the intensity index using the same set of accuracy metrics
- **Metrics**: precision, recall, F-Measure, AUC-ROC, and Brier score

- analyzed to what extent the two models are complementary in the classification of the bugginess of classes affected by code smells

$$TP_{m_{int} \cap m_{ant}} = \frac{|TP_{m_{int}} \cap TP_{m_{ant}}|}{|TP_{m_{int}} \cup TP_{m_{ant}}|}\%$$

$$TP_{m_{int} \setminus m_{ant}} = \frac{|TP_{m_{int}} \setminus TP_{m_{ant}}|}{|TP_{m_{int}} \cup TP_{m_{ant}}|}\%$$

$$TP_{m_{ant} \setminus m_{int}} = \frac{|TP_{m_{ant}} \setminus TP_{m_{int}}|}{|TP_{m_{ant}} \cup TP_{m_{int}}|}\%$$

model built by considering the antipattern metrics

model built plugging in the intensity index

DESIGN OF THE CASE STUDY —— Data Analysis and Metrics

RQ3 - Gain Provided by the Intensity Index

- An information gain algorithm to quantify the gain provided by adding the intensity index in each prediction model.

$$InfoGain(M, p_i) = H(M) - H(M|p_i)$$

$H(M)$: indicates the entropy of the model that **includes** the predictor **pi**

$H(M|p_i)$: measures the entropy of the model that does **not include pi**

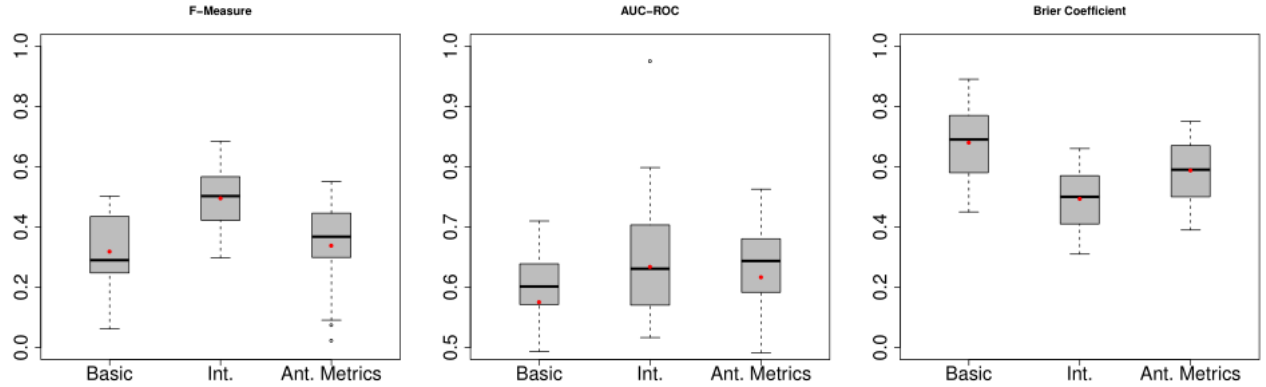
RQ4 - Combining Basic Predictors and Smell related Metrics

- **goal**: to find a combined set of metrics that uses smell-related information together with product and process metrics to achieve better performances.

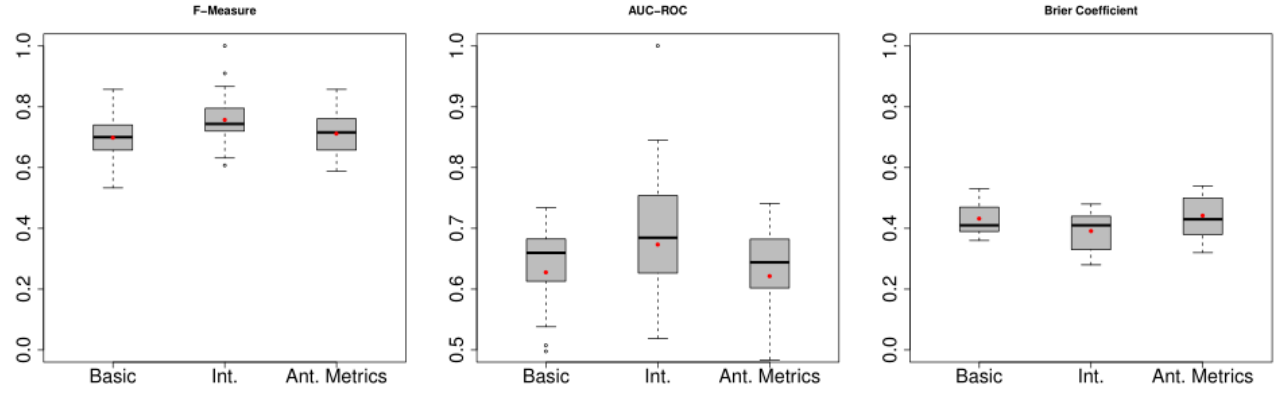
EXPERIMENTAL RESULTS

- The resulting structural model is composed of 16 metrics.

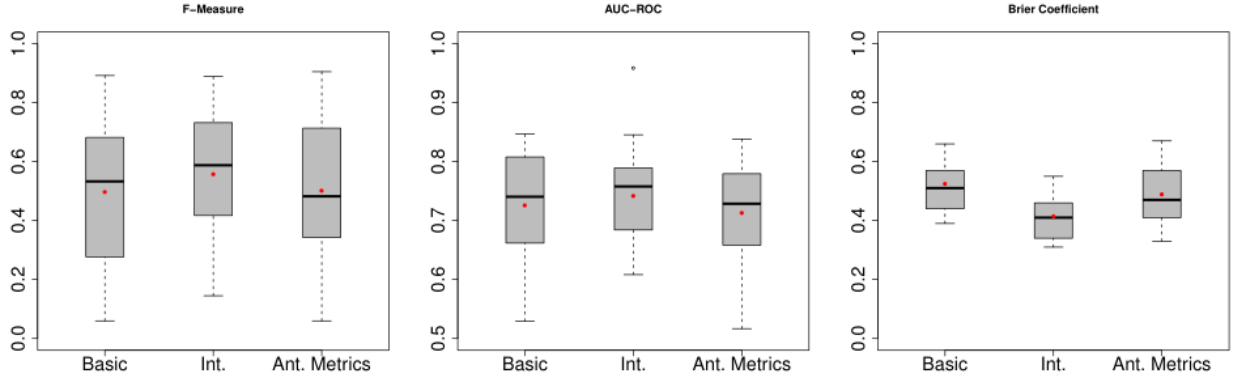
(c) Performances of DM-based Models. *Basic* refers to the model built only using the number of developers, *Int.* refers to the model in which the intensity index is an additional predictor, *Ant. Metrics* refers to the model where the antipattern metrics are included as additional predictors.



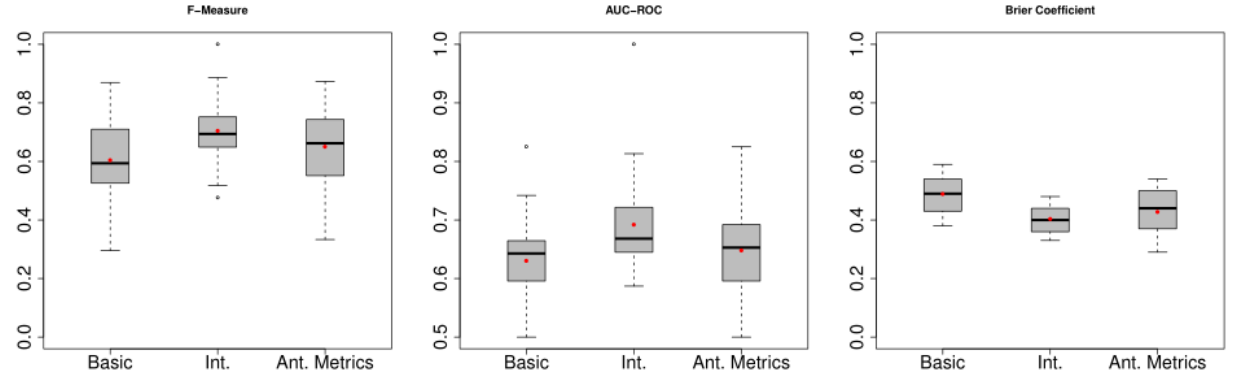
(d) Performances of DCBM-based Models. *Basic* refers to the model built only using scattering metrics, *Int.* refers to the model in which the intensity index is an additional predictor, *Ant. Metrics* refers to the model where the antipattern metrics are included as additional predictors.



ı) Performances of Structural Metric-based Models. *Basic* refers to the model built only using structural ıetrics, *Int.* refers to the model in which the intensity index is an additional predictor, *Ant. Metrics* refers › the model where the antipattern metrics are included as additional predictors.



›) Performances of Entropy-based Models. *Basic* refers to the model built only using the entropy of changes, ıt. refers to the model in which the intensity index is an additional predictor, *Ant. Metrics* refers to the model ‹here the antipattern metrics are included as additional predictors.



EXPERIMENTAL RESULTS

Complementary

The ***Basic + Ant.*** Metrics model always achieved lower performance than the ***Basic + Intensity*** model (considering the median of the distributions, -8% of F-Measure, -3% of AUC-ROC, and +0.06 of Brier score).

Antipattern metrics can provide useful and complementary information with respect to the intensity index.

TABLE 7: Percentage of Smelly and Non-Smelly Classes Correctly Classified by the Experimented Models

Basic Model	Configuration	% Cor. Class. Smelly Instances	Cor. Class. Non-Smelly Instances
Structural Metrics [30]	Basic	45	66
	Basic + Intensity	**69**	**68**
	Basic + Ant. Metrics	48	66
BCCM [5]	Basic	42	55
	Basic + Intensity	**55**	**68**
	Basic + Ant. Metrics	49	56
DM [56]	Basic	31	41
	Basic + Intensity	**63**	**46**
	Basic + Ant. Metrics	42	43
DCBM [33]	Basic	53	87
	Basic + Intensity	**77**	**88**
	Basic + Ant. Metrics	64	85

TABLE 10: Overlap analysis between the model including the intensity index and the model including the antipattern metrics.

Basic Model	Int. $\cap$ Ant.%	Int. $\setminus$ Ant.%	Ant. $\setminus$ Int.%
Structural Metrics [30]	39	34	27
BCCM [5]	45	38	17
DM [56]	54	38	8
DCBM [33]	42	37	21

EXPERIMENTAL RESULTS

TABLE 12: Gain Provided by Each Metric To The BCCM Prediction Model.

Metric	Mean	St. Dev.	Class	SK-ESD Likelihood
Entropy of Changes	0.84	0.08	non-buggy	92
Intensity	**0.44**	**0.11**	**buggy**	**77**
Average Number of Antipatterns	0.29	0.13	buggy	56
Antipattern Complexity Metric	0.07	0.03	non-buggy	18
Antipattern Recurrence Length	0.03	0.06	non-buggy	11

TABLE 13: Gain Provided by Each Metric To The DM Prediction Model.

Metric	Mean	St. Dev.	Class	SK-ESD Likelihood
# Developers	0.75	0.14	non-buggy	85
Intensity	**0.39**	**0.15**	**buggy**	**61**
Average Number of Antipatterns	0.34	0.13	buggy	56
Antipattern Complexity Metric	0.12	0.36	buggy	14
Antipattern Recurrence Length	0.07	0.03	non-buggy	8

TABLE 11: Gain Provided by Each Metric To The Prediction Model based on Structural Metrics.

Metric	Mean	St. Dev.	Class	SK-ESD Likelihood
CBO	0.41	0.23	buggy	67
LCOM3	0.34	0.25	non-buggy	74
WMC	0.33	0.05	buggy	61
Intensity	**0.32**	**0.16**	**non-buggy**	**53**
DAM	0.25	0.06	buggy	48
DIT	0.22	0.05	non-buggy	45
Average Number of Antipatterns	0.21	0.09	buggy	44
AMC	0.15	0.11	non-buggy	31
LOC	0.15	0.07	buggy	25
MFA	0.14	0.02	buggy	23
IC	0.11	0.03	non-buggy	17
CBM	0.09	0.04	non-buggy	14
Antipattern Complexity Metric	0.07	0.02	buggy	9
AVG(CC)	0.05	0.05	buggy	5
CE	0.04	0.02	buggy	5
CAM	0.03	0.02	non-buggy	3
Antipattern Recurrence Length	0.03	0.02	buggy	2
MOA	0.02	0.01	non-buggy	2
NOC	0.01	0.02	non-buggy	2
NPM	0.01	0.01	buggy	2

TABLE 14: Gain Provided by Each Metric To The DCBM Prediction Model.

Metric	Mean	St. Dev.	Class	SK-ESD Likelihood
Structural Scattering	0.67	0.22	buggy	93
Semantic Scattering	0.54	0.26	non-buggy	85
Intensity	**0.33**	**0.15**	**buggy**	**48**
Average Number of Antipatterns	0.25	0.11	buggy	41
Antipattern Complexity Metric	0.10	0.04	buggy	22
Antipattern Recurrence Length	0.06	0.09	non-buggy	11

EXPERIMENTAL RESULTS

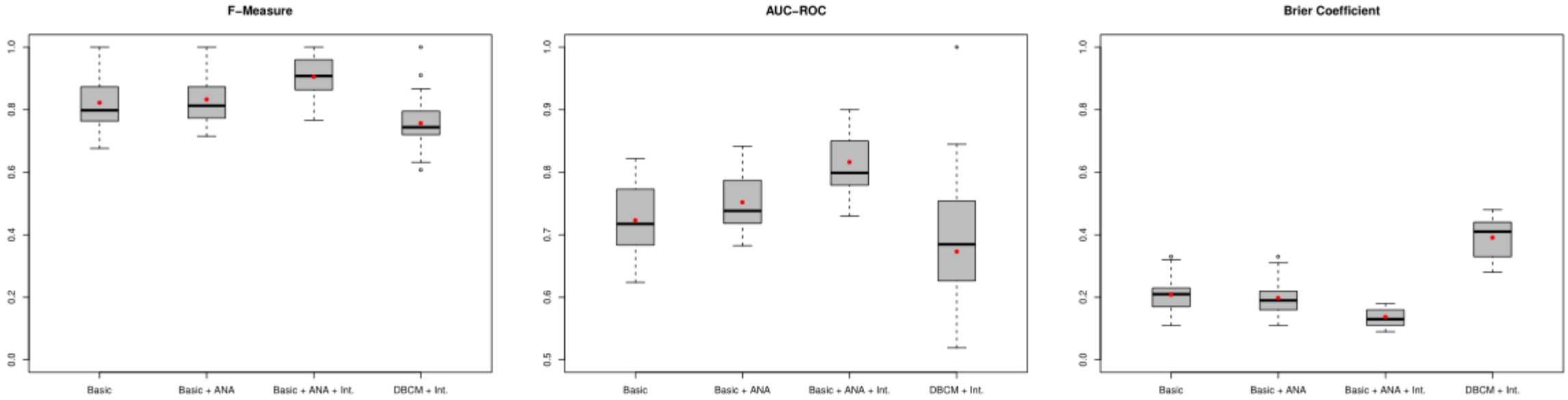


Fig. 3: Performances of the Smell-Aware Bug Prediction Model.

TABLE 16: Percentage of Smelly and Non-Smelly Classes Correctly Classified by the Combined Models

Basic Model	Configuration	% Cor. Class. Smelly Instances	Cor. Class. Non-Smelly Instances
Combined	Basic	81	90
Combined	Basic + ANA	84	90
Combined	**Basic + ANA + Intensity**	**93**	**91**
DCBM [33]	Basic + Intensity	77	88

论　　文　　汇　　报　　展　　示

感 谢 您 的 聆 听

汇报人：王昭丹