SEL@KIT

Photo by Osamu Mizuno

A Fault-Prone Module Detection Using a Spam Filter

KYOTO INSTITUTE OF TECHNOLOGY

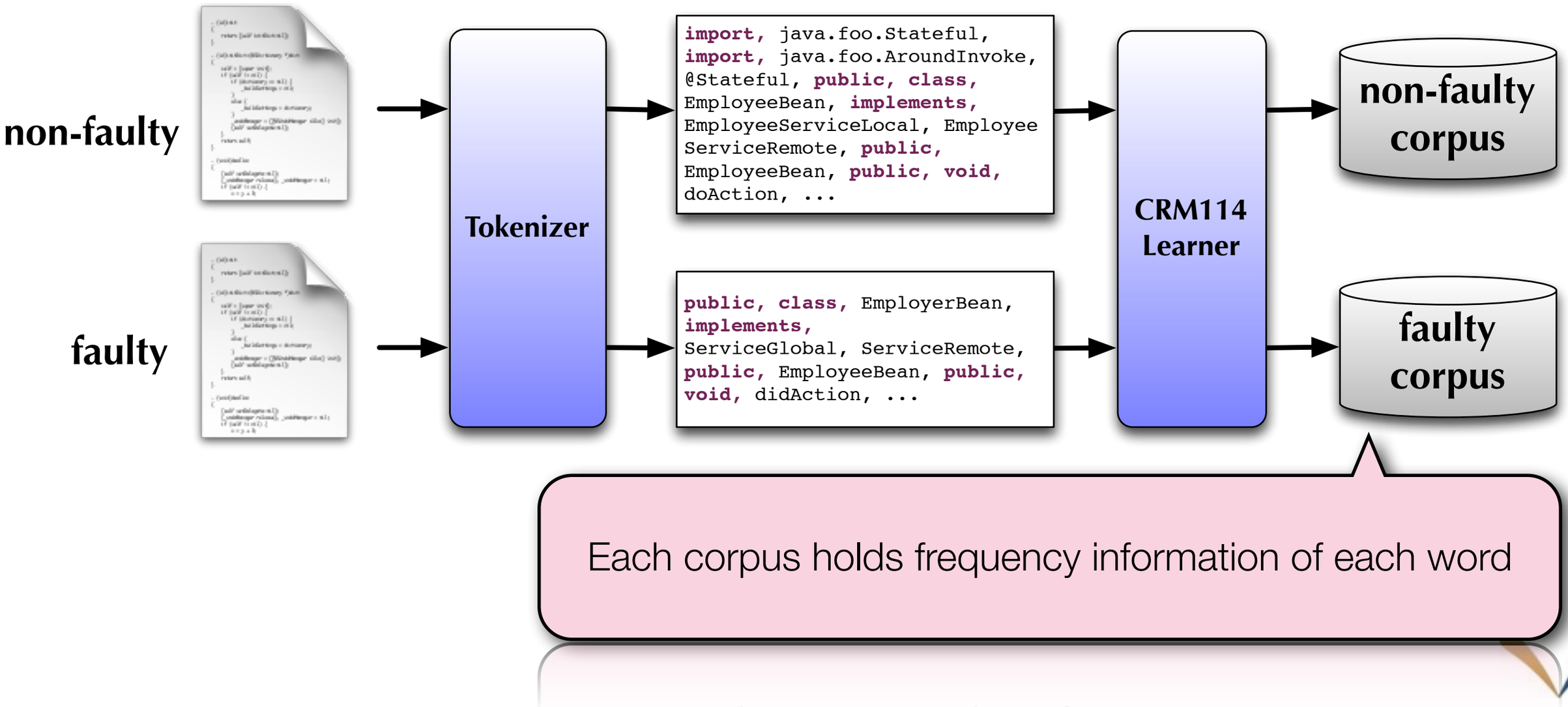
Fault-prone Filtering

- Fault-prone module detection using a spam filtering approach[1][2]
  - Uses frequency of terms like spam e-mail filtering
    - Constructs both faulty and non-faulty corpuses from past modules
  - Classifies an unknown module using two corpuses

[1] O. Mizuno and T. Kikuno, "Training on Errors Experiment to Detect Fault-Prone Software Modules by Spam Filter," In The 6th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE2007), pp. 405-414, September 2007. (Dubrovnik, Croatia)
[2] O. Mizuno, S. Ikami, S. Nakaichi, and T. Kikuno, "Spam Filter Based Approach for Finding Fault-Prone Software Modules," In Proc. of Fourth International Workshop on Mining Software Repositories (MSR'07), p. 4, May 2007. (Minneapolis, MN, USA)

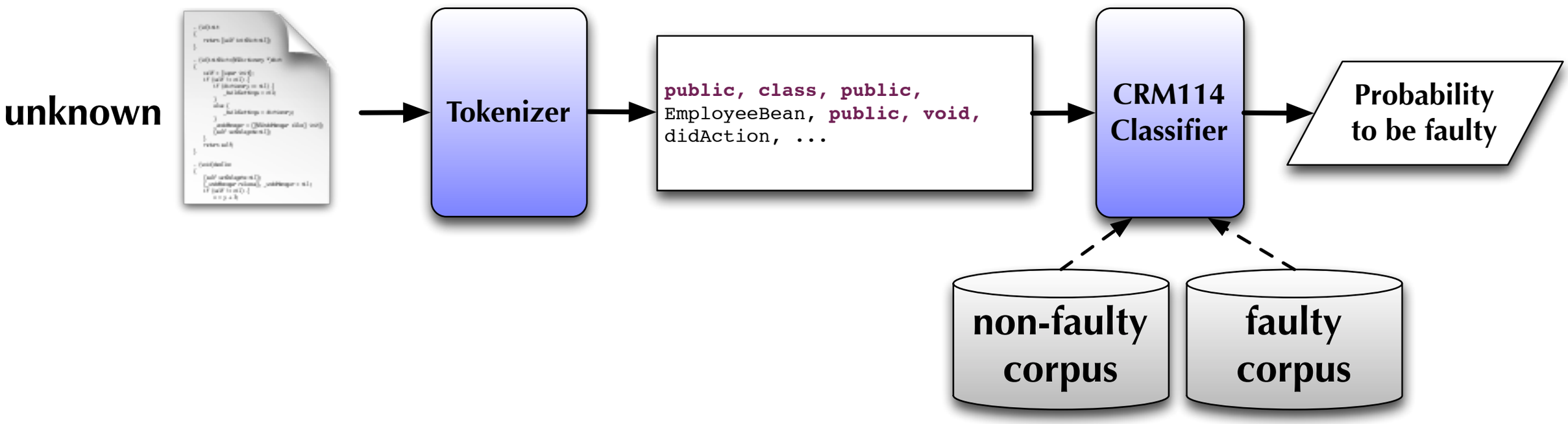
How It Works (Training)

1. Training faulty and non-faulty modules using Tokenizer.



How It Works (Prediction)

2. Calculating probability by the Bayesian filter.



Fault-proneness is determined by the probability and a threshold.

KYOTO INSTITUTE OF TECHNOLOGY

Experiment:

Result of Prediction

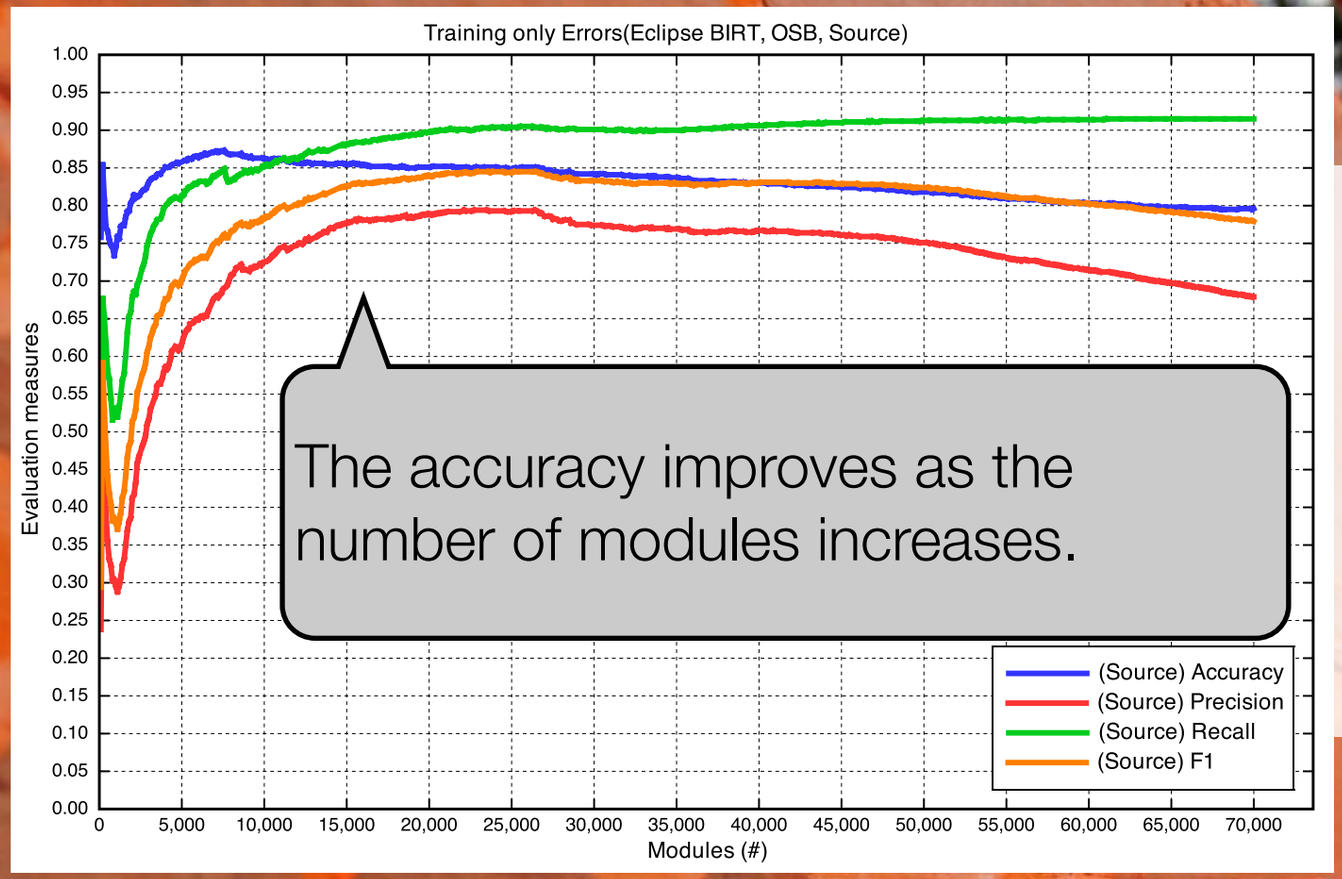
Eclipse (random sample)		Predicted	
		NFP	FP
Actual	not faulty	12,249	7,093
	faulty	2,972	16,243

Precision: 0.696 Recall: 0.845

Eclipse BIRT (all modules)		Predicted	
		NFP	FP
Actual	not faulty	70,349	16,011
	faulty	2,039	7,501

Precision: 0.319 Recall: 0.786

KYOTO INSTITUTE OF TECHNOLOGY



The accuracy improves as the number of modules increases.

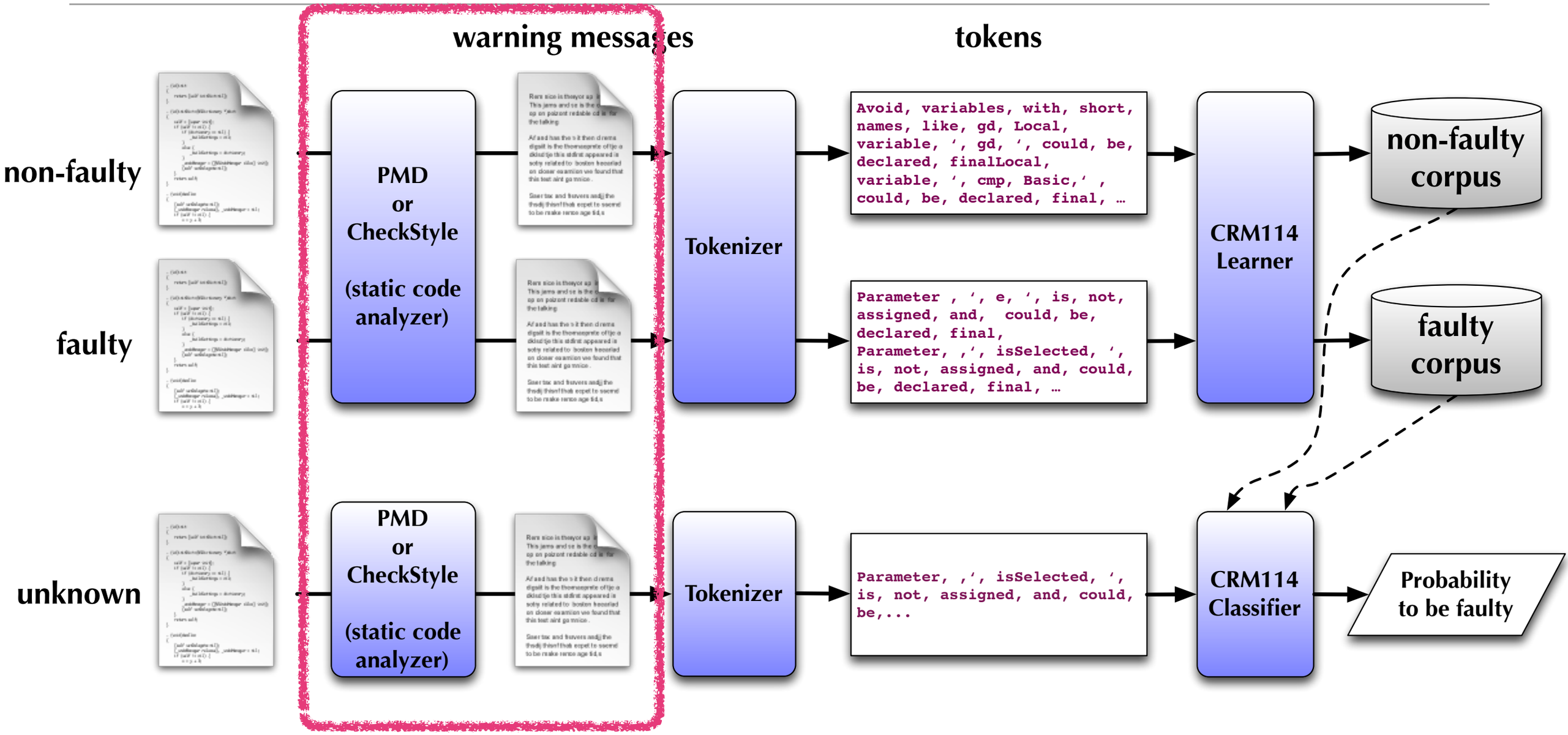
Apply software modules to fault-prone filter in order of construction and modification.

Only misclassified modules are trained in corpuses.

Successive Prediction (TOE)

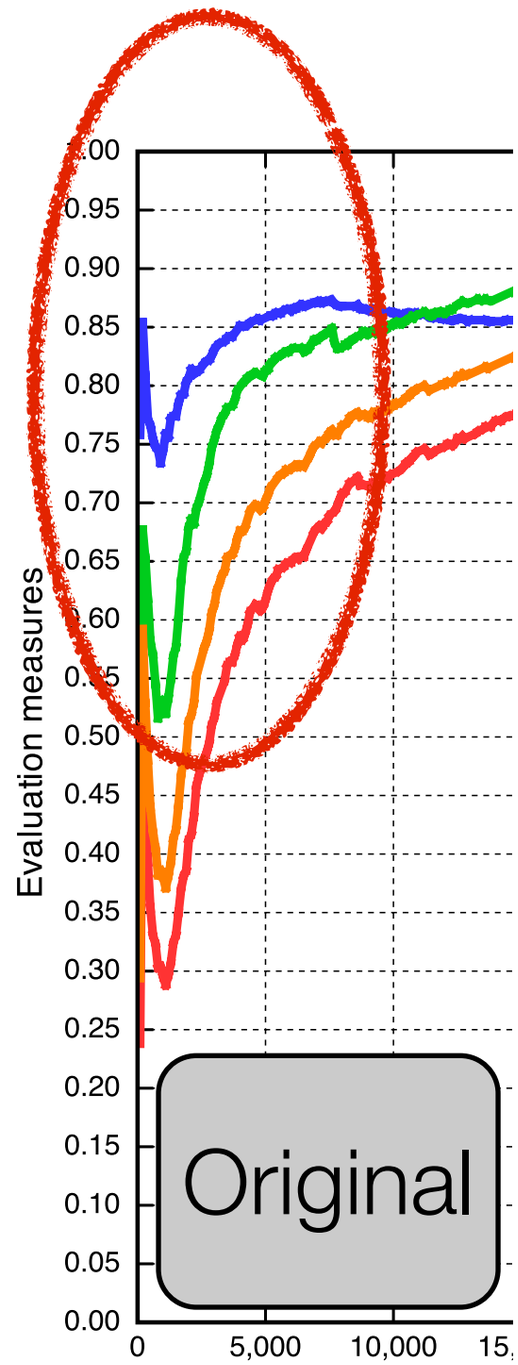
KYOTO INSTITUTE OF TECHNOLOGY

Fault-proneness filtering with static code analysis



KYOTO INSTITUTE OF TECHNOLOGY

Result


Evaluation measures
1.00
0.95
0.90
0.85
0.80
0.75
0.70
0.65
0.60
0.55
0.50
0.45
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
0
5,000
10,000
15,
Original

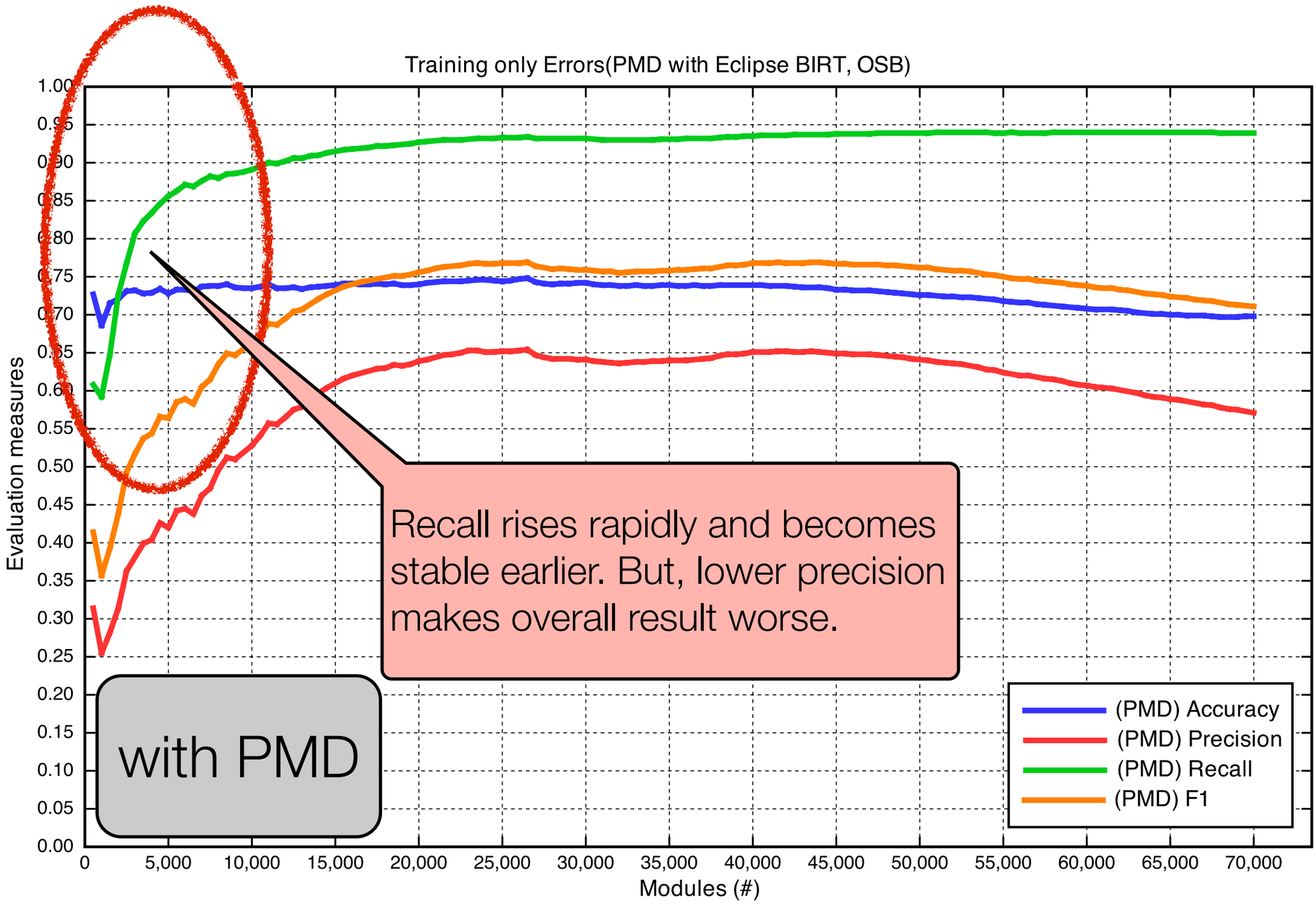

Training only Errors(PMD with Eclipse BIRT, OSB)
Evaluation measures
1.00
0.95
0.90
0.85
0.80
0.75
0.70
0.65
0.60
0.55
0.50
0.45
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
0
5,000
10,000
15,000
20,000
25,000
30,000
35,000
40,000
45,000
50,000
55,000
60,000
65,000
70,000
Modules (#)
with PMD
Recall rises rapidly and becomes stable earlier. But, lower precision makes overall result worse.
(PMD) Accuracy
(PMD) Precision
(PMD) Recall
(PMD) F1

Photo by thenys
www.freeimages.com

Analysis of Identifiers

KYOTO INSTITUTE OF TECHNOLOGY

What we did

We investigated the frequency of appearance of identifiers in source code modules from the viewpoint of the length of identifiers in Eclipse and NetBeans.

We modelled the relationship between the length of identifiers and the fault-proneness.

We used the random forest for the modelling.

Data used:

Frequency of identifier's occurrence for each file grouped by the length of identifier + faulty status by SZZ

Kimiaki Kawamoto and Osamu Mizuno, "Do Long Identifiers Induce Faults in Software? --- a Repository Mining Based Investigation ---," In Proc. of 22nd International Symposium on Software Reliability Engineering (ISSRE2011), Supplemental proceedings, 3-1, November 2011. (Hiroshima, Japan)

Results of analysis

Table 2 Evaluation of prediction (average of 10 times)

	Accuracy	Precision	Recall	F-Measure
Eclipse	0.888	0.866	0.913	0.889
Netbeans	0.850	0.765	0.865	0.812

Table 3 Top 10: Mean Decrease Gini (average of 10 times)

	Eclipse		Netbeans	
Rank	Length	Mean decrease Gini	Length	Mean decrease Gini
1	7	7767.47	3	7505.82
2	4	6501.14	4	6537.75
3	8	6146.32	10	6505.30
4	3	5643.99	7	6187.07
5	13	5378.26	6	6109.80
6	6	5073.66	11	5342.15
7	9	4883.39	9	5176.19
8	5	4664.90	8	5156.06
9	11	4200.25	14	4900.95
10	14	4140.85	13	4653.10

Lengths of 3, 4, and 7 have impact for fault-proneness.

[FYI] The longest identifiers

In Eclipse: 237 characters, appears 26 times

WorkingDirectoryStatusHandler_Eclipse_is_not_able_to_set_the_working_directory_specified_by_the_program_being_launched_as_the_current_runtime_does_not_support_working_directories__nContinue_launch_without_setting_the_working_director y__2

In Netbeans: 210 characters, appears 76 times

getCustomResourceOrExternalJndiResourceOrJdbcResourceOrMailResourceOrPersistenceManagerFactoryResourceOrAdminObjectResourceOrConnectorResourceOrResourceAdapterConfigOrJdbcConnectionPoolOrConnectorConnectionPool

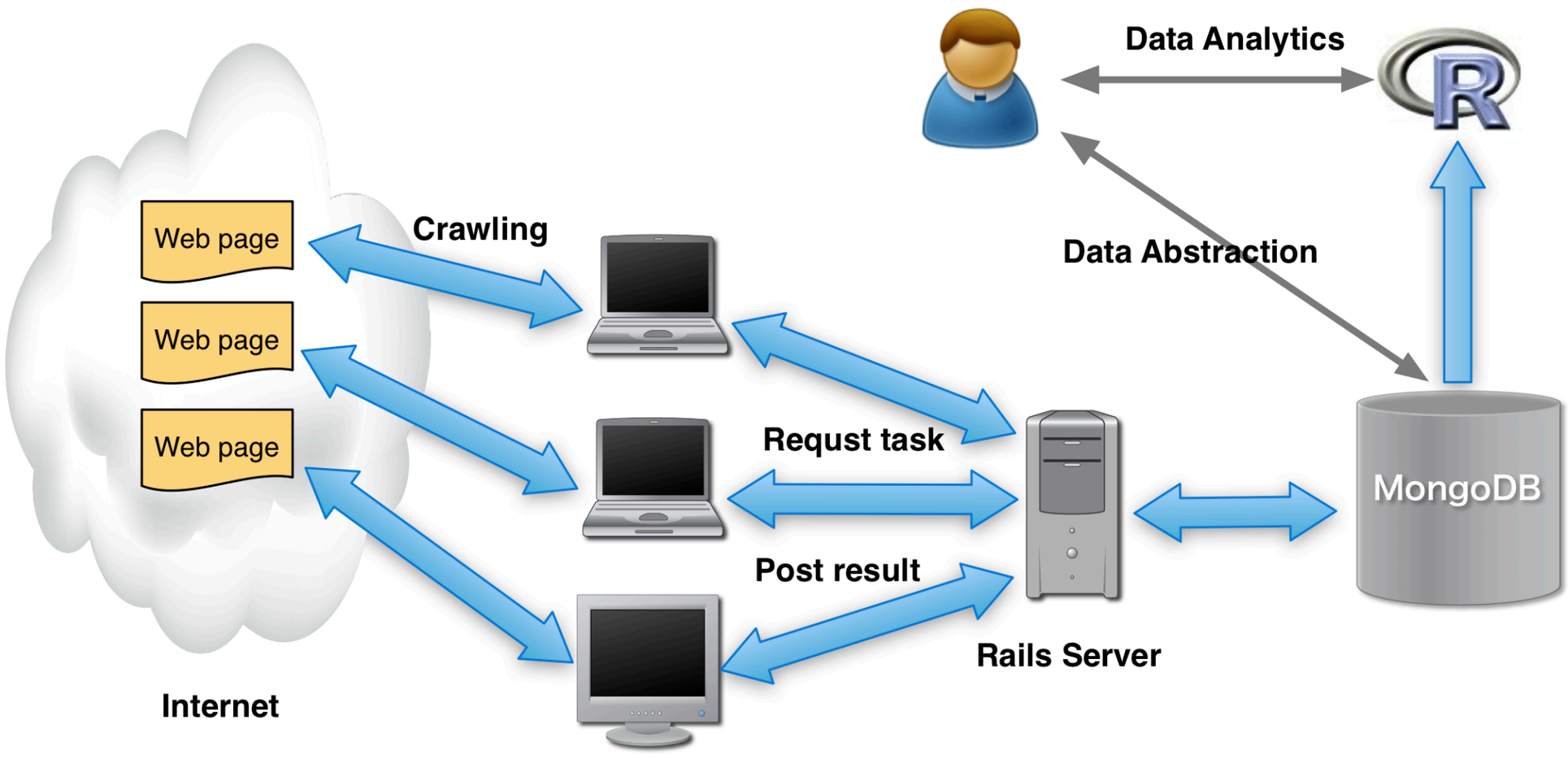
Do code review activities become more productive in Gerrit-based projects as a result of evolution?

Photo by Paul Barker (www.freeimages.com)

Data collection tool (Developed)

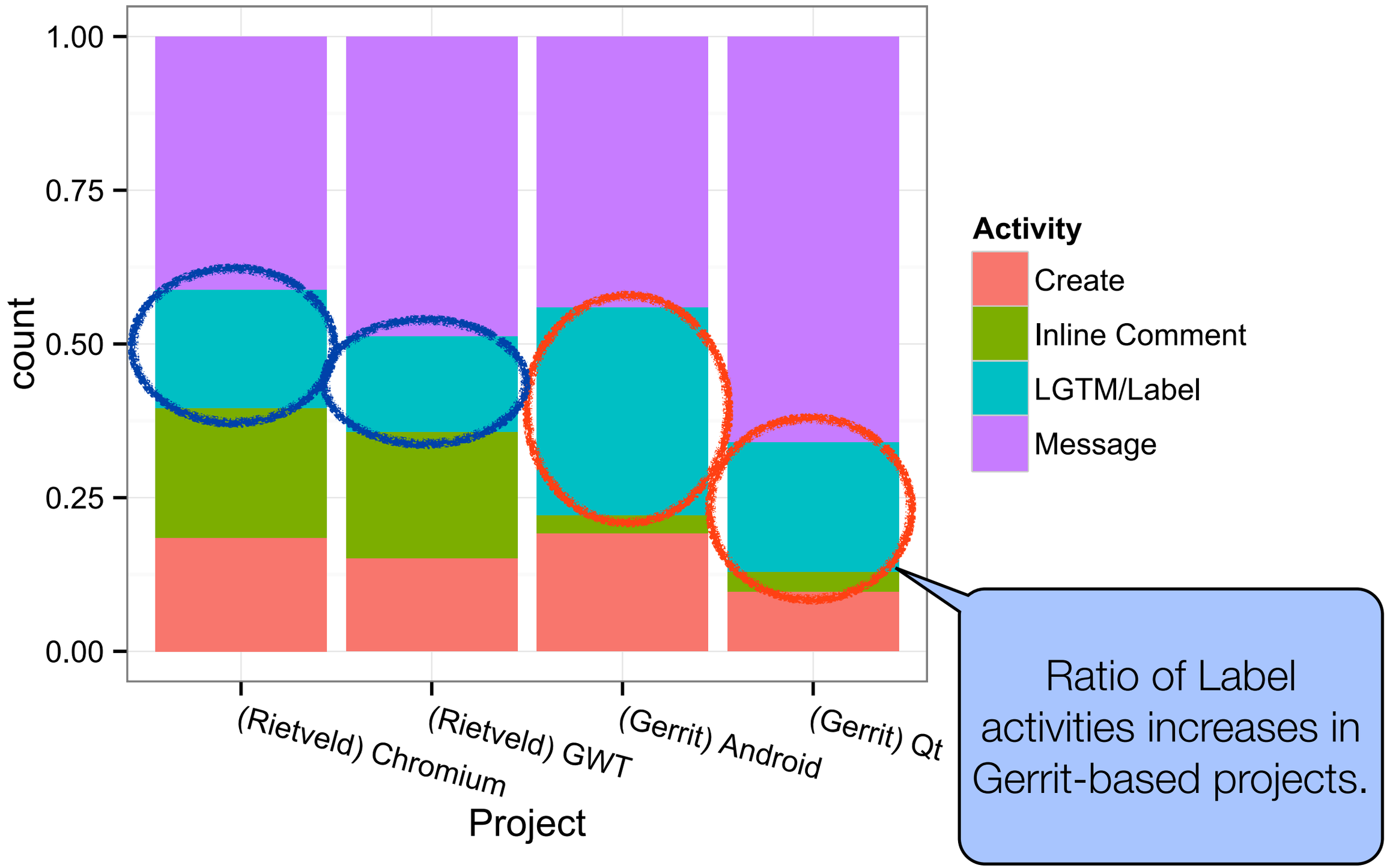
Integrated crawling and mining tool [1].

Developed by Junwei Liang (a master course student)



[1] JunWei Liang and Osamu Mizuno, "Analyzing Involvements of Reviewers Through Mining a Code Review Repository," In Proc. of the Joint Conference of the 21th International Workshop on Software Measurement and the 6th International Conference on Software Process and Product Measurement (IWSM/MENSURA2011), pp. 126-132, November 2011. (Nara, Japan)

Ratio of activities in review process



Ratio of Label activities increases in Gerrit-based projects.

Discussion time and approval

- In Rietveld, the number of LGTM is proportional to the discussion time.
- In Gerrit, +2 labeled issues quickly finish discussion.

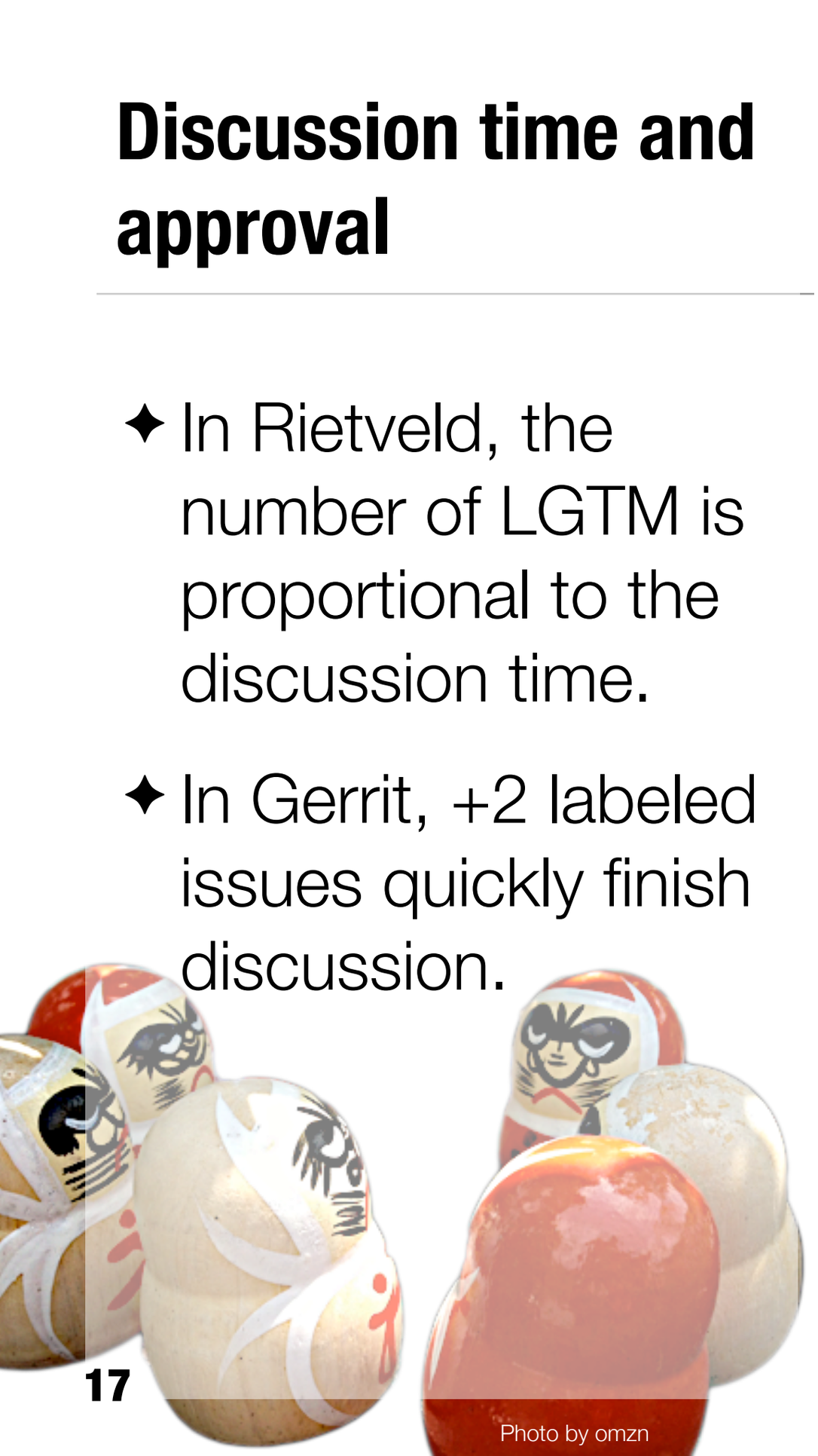


Photo by omzn