Classifying Software Changes : Clean or Buggy?

IEEE Transactions on Software Engineering (TSE 2008)

Sunghun Kim, E. James Whitehead Jr., Yi Zhang

2012-07-17
Gwangui Hong

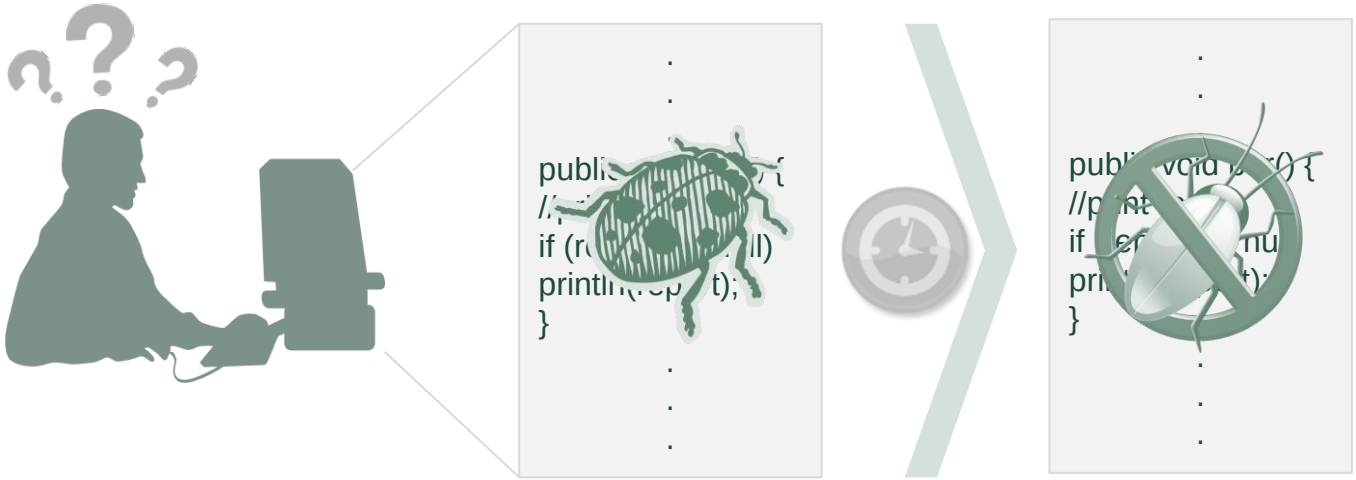
KAIST SE LAB
KAIST Software Engineering Laboratory

Contents

- Introduction
- Background
- Overall approach
- Change classification
- Case study
- Related work
- Conclusion
- Discussion

Introduction (1/3)

- Developers make software changes to add new features.
- Sometimes the changes introduce bugs.
- It is time consuming that spend time to relearn source codes to fix latent bugs.



Introduction (2/3)

- Previous work on bug prediction
  - Gyimothy et al.'s model to predict buggy classes with class level of granularity

    ➡ Granularity is too huge.
    Developers need to examine small amount of LOC.

  - BugMem to capture bug patterns in previous fixes
  - Brun and Ernst's model to find hidden code errors
  - Other models to predict bug

    ➡ They focus on the source code only to predict.
    Other properties (e.g. log, metadata, file name) have to be considered.

Introduction (3/3)

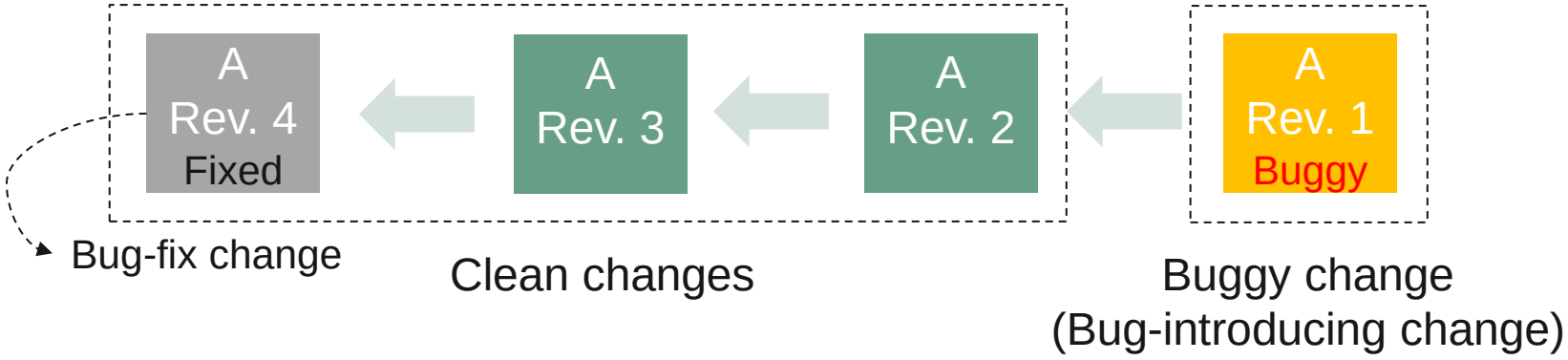
- Motivation
  - A tool which predicts bugs after changes made with small granularity and other properties is needed.
  - The tool will reduce the time required to find bugs and reduce the bug-stay time before fixing.

- Goal
  - Building a new bug prediction model with file level granularity called change classification

Background (1/3)

- Change
  - Each revision of file is called change.

- Clean/Buggy change
  - Clean change : Revision which does not cause a bug
  - Buggy change : Revision which causes a bug



* Assumption : A bug is repaired in a single bug-fix change.

Background (2/3)

- SZZ algorithm

  (Sliwerski, Zimmermann, and Zeller)

  - It is used for finding the bug-introducing change.
  - It does backward search in the revision history.
  - It identifies source code change and developer.

Rev.	Dev.	Code
2	ejw	1: public void foo() {
1	kim	2: //print report
3	kai	3: if (report != null)
1	kim	4: println(report);
1	kim	5: }

Rev.	Dev.	Code
1	kim	1: public void bar() {
1	kim	2: //print report
1	kai	3: if (report == null)
1	kim	4: println(report);
1	kim	5: }

Revision 3
Bug-fix change

Revision 1
Bug-introducing change

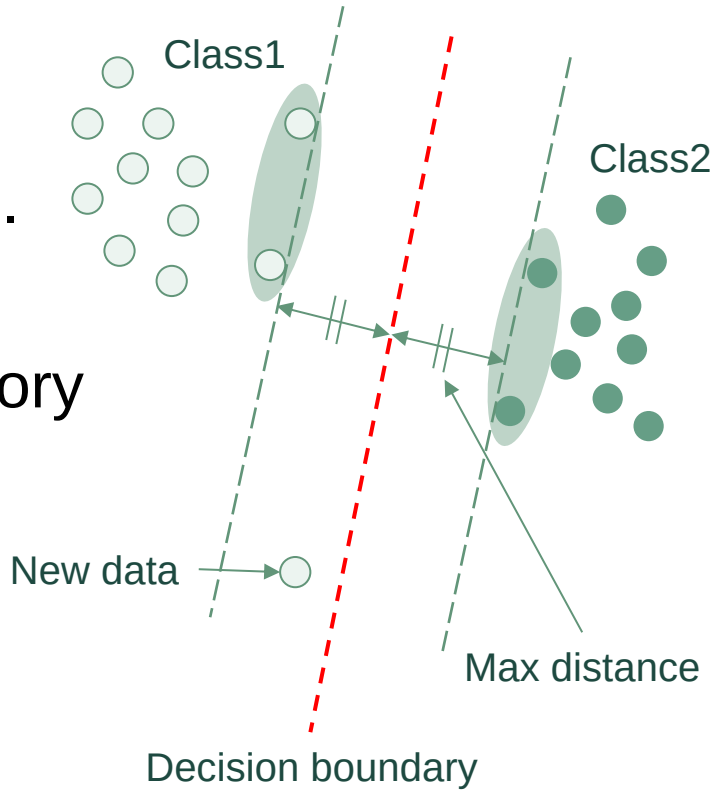
Rev.	Dev.	Code
2	ejw	1: public void foo() {
1	kim	2: //print report
1	kim	3: if (report == null)
2	ejw	4: println(report.str);
1	kim	5: }

Revision 2

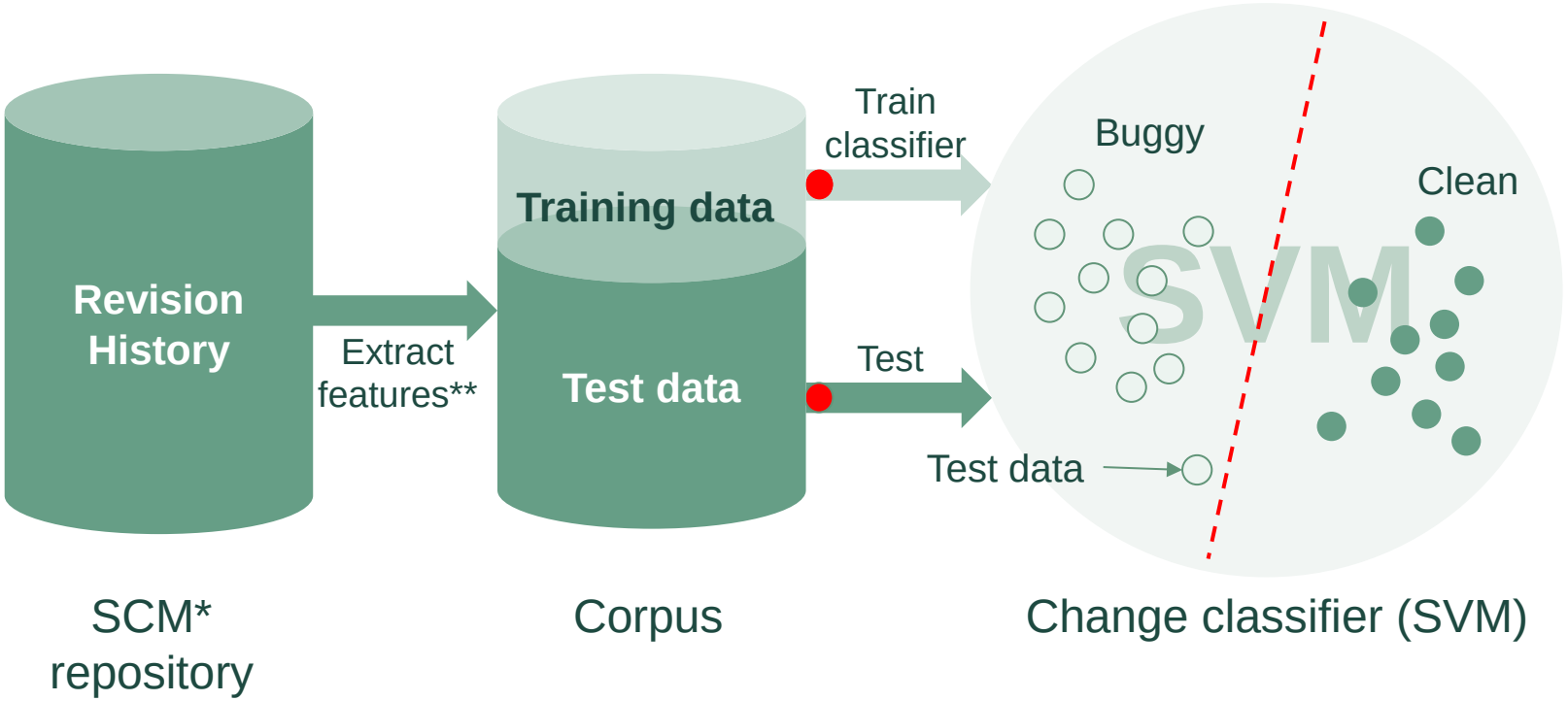
Revision 2

Background (3/3)

- SVM (Support Vector Machine)
  - It is machine learning binary classifier.
  - Maximum distance boundary exists between the two classes.
  - Each Class represents a category of data.
  - Boundary is used for making decision for new data.



Overall approach



*SCM : **S**oftware **C**onfiguration **M**anagement
**Feature : Properties of the change

Creating corpus

1. Change history extraction
2. Identifying bug-introducing changes
3. Feature extraction

Change history extraction (1/2)

- Target programs
  - 12 open source projects

Project			
Apache HTTP	1.3, Bugzilla	Columba	Gaim
GForge	JEdit	Mozilla	Plone
Eclipse	PostgreSQL	Scarab	Subversion

  - 500-1,000 revisions (500-750 for large projects)

- Extracted change information from SCM

  (**S**oftware **C**onfiguration **M**anagement e.g. CVS, SVN)
  - Change log, author, change date, source code, change delta, and change metadata

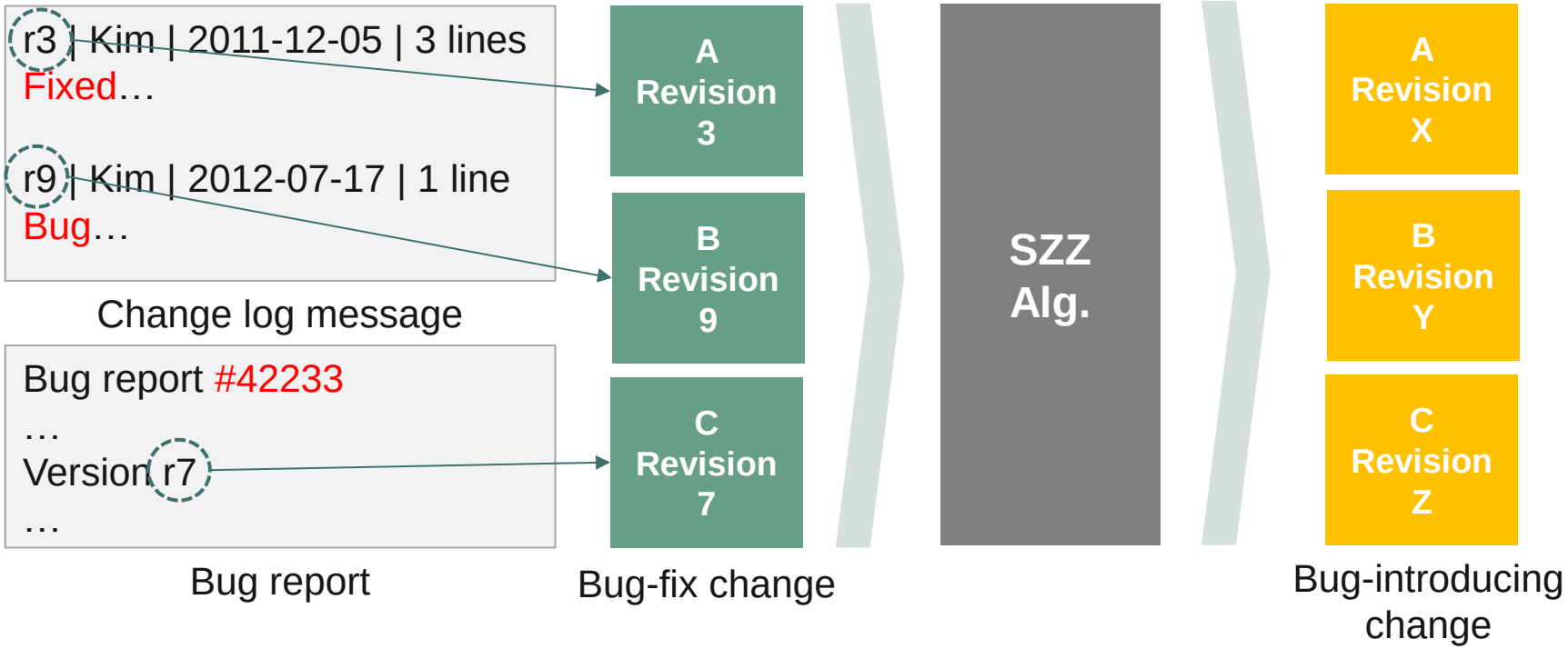
Change history extraction (2/2)

Project	Rev.	Period	# of clean changes	# of buggy changes	% of buggy changes
Apache HTTP 1.3	500-1000	199610-199701	579	121	13.3
Bugzilla	500-1000	200003-200108	149	417	73.7
Columba	500-1000	200305-200309	1,270	530	29.4
Gaim	500-1000	200008-200103	742	451	37.8
Gforge	500-1000	200301-200403	339	334	49.6
JEdit	500-750	200208-200303	626	377	37.5
Mozilla	500-1000	200308-200408	395	169	29.9
Eclipse	500-750	200110-200111	592	67	10.1
Plone	500-1000	200207-200302	457	112	19.6
PostgreSQL	500-1000	199611-199702	853	273	24.2
Scarab	500-1000	200106-200108	358	366	50.5
Subversion	500-1000	200201-200203	1,925	288	13.0
Total	N/A	N/A	8,285	3,505	32.7(Avg.)

<Subject programs and summary of corpus information>

Identifying bug-introducing changes (1/2)

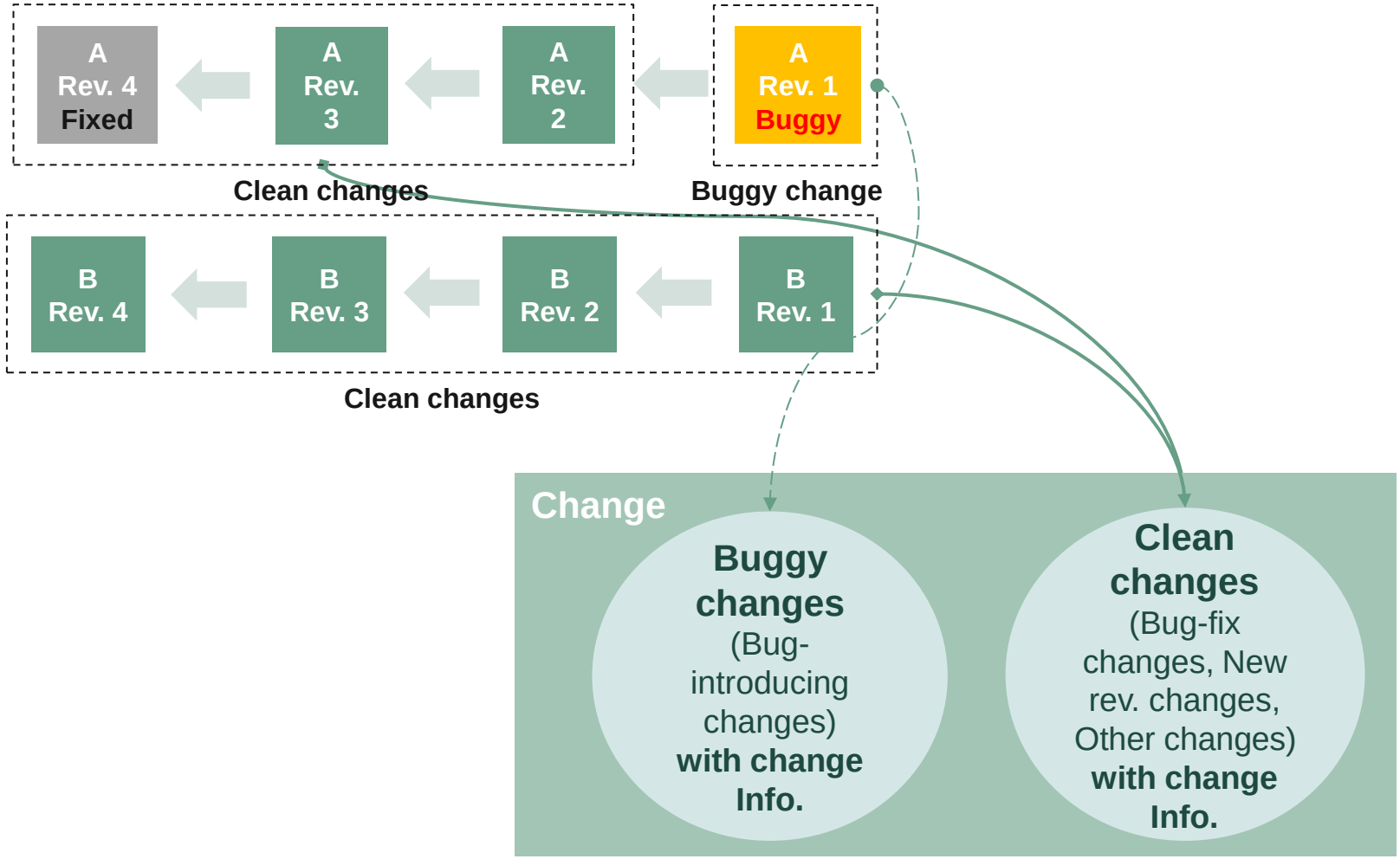
- Bug-introducing change(=A revision file)
  - It is an initial change which causes a bug.
  - It is identified by finding bug-fix changes.



<Identify bug-introducing changes from bug-fix changes>

Identifying bug-introducing changes (2/2)

- Two categories of the change



Feature extraction (1/2)

- Feature (Property of the change)
  - Every term in the source code, change delta, and change log text is used as features.
  - Features are generated from all file changes.
  - 7 Feature groups

Feature group	Description
Added delta (A)	Terms in the added delta source code
Deleted delta (D)	Terms in the deleted delta source code
Directory/File name (F)	Terms in the directory/file names
Change log (L)	Terms in the change log
New revision source code (N)	Terms in the new revision source code file
Metadata (M)	Change metadata e.g. time and author
Complexity metrics (C)	S/W complexity metrics of each source code

Feature extraction (2/2)

- Feature extraction method

Feature	Method
Metadata	Author, Commit hour (0~12), Commit day (sun~mon), Cumulative change count, Length of change log, Changed LOC (added delta LOC + deleted delta LOC), New revision source code
Complexity metric	61 Complexity metrics for each file including LOC, lines of comments, max nesting…
Log message	BOW
Source code	BOW+, Modified version of BOW which takes special char.
File name	BOW++, Modified version of BOW which extracts words in the directory* and filenames**.

*Directory : src/mail/core/columba → “src”, “mail”, “core”, “columba”

**Filename : ReceiveOptionPanel.java → “receive”, “option”, “panel”, “java”

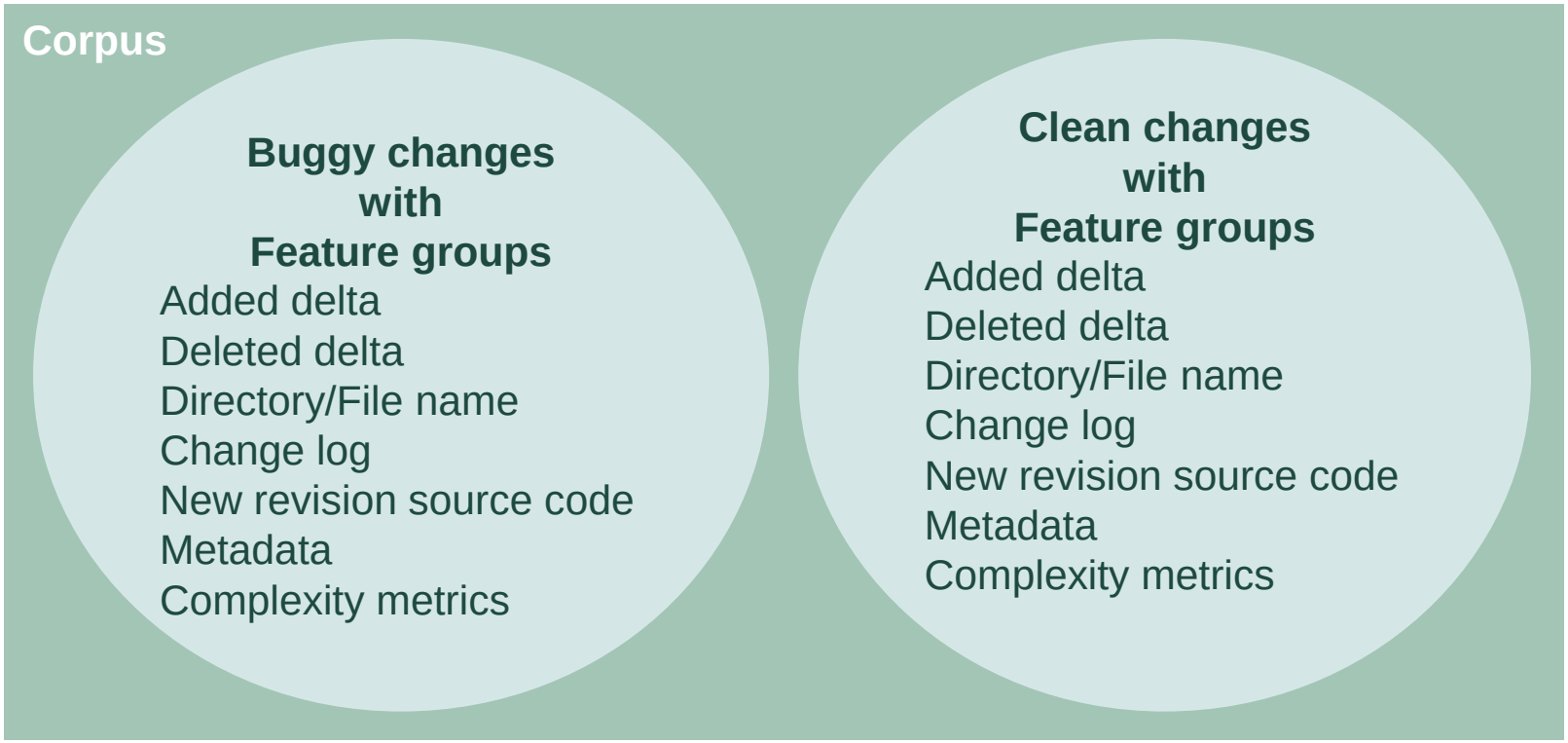
Feature extraction (2/2)

- Feature groups and applied method

Feature group	Method	Example
Added delta (A)	BOW+	if, while, for, ==
Deleted delta (D)	BOW+	true, 0, <, ++, int
Directory/File name (F)	BOW++	src, module, java
Change log (L)	BOW	fix, added, new
New revision source code (N)	BOW+	if, ||, !=, do, while, string, false
Metadata (M)	Direct	Author : Kim, Commit hour : 12
Complexity metrics (C)	61 Metrics	LOC : 34, Line of comments : 9

Creating corpus

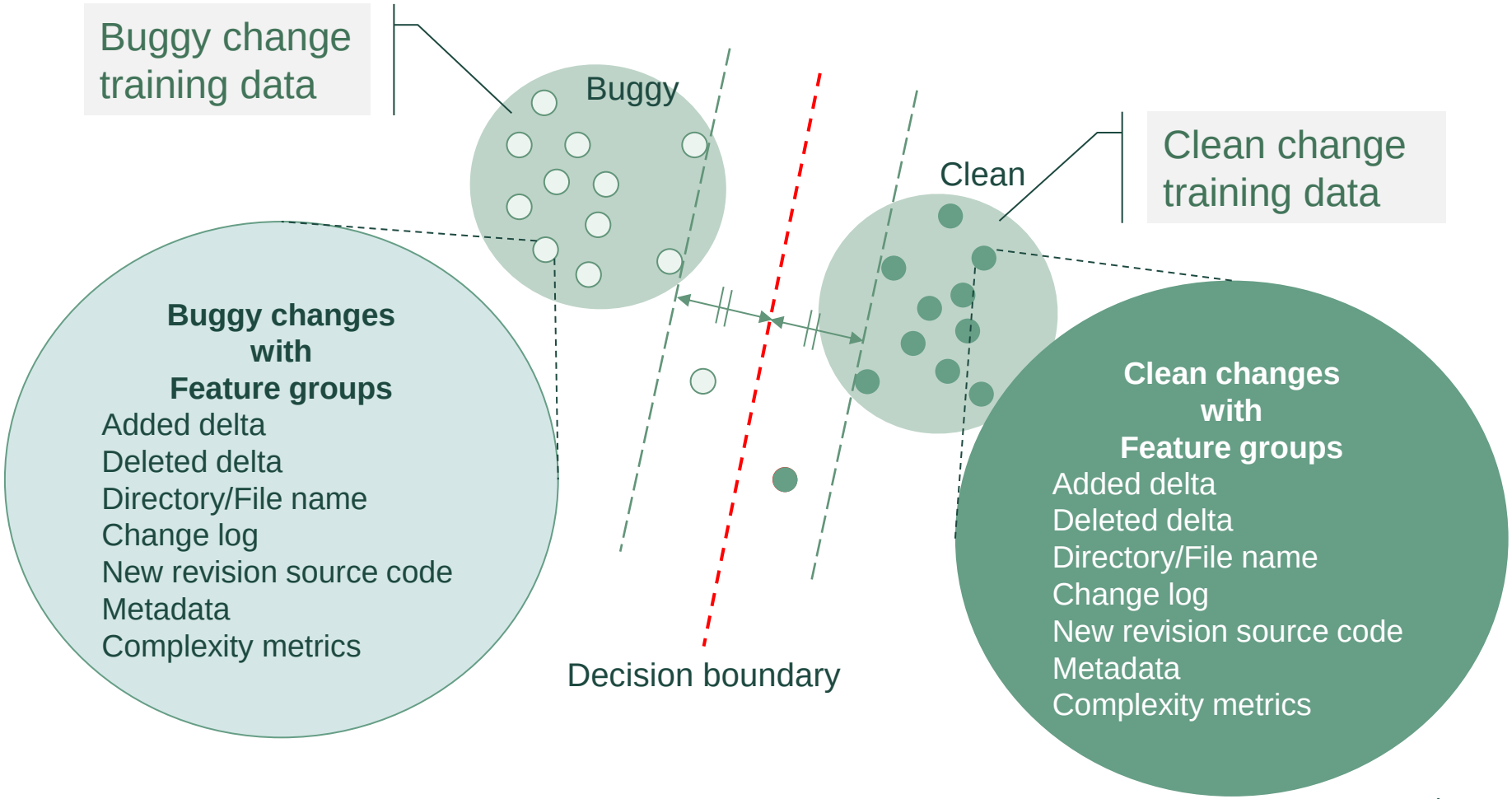
- Created corpus



<Overview of corpus>

Change classification

- Using the corpus, a classification model is trained.



Case study

1. Evaluate the performance of change classification
2. Evaluate the accuracy of each feature group

Evaluate change classification (1/3)

- Goal
  - To evaluate the performance of SVM models
- Target
  - SVM models for 12 projects using all features
- Method
  - 1. Assessing of SVM models with metrics
    - Accuracy, Recall, Precision
  - 2. Comparing SVM models with dummy predictor

Evaluate change classification (2/3)

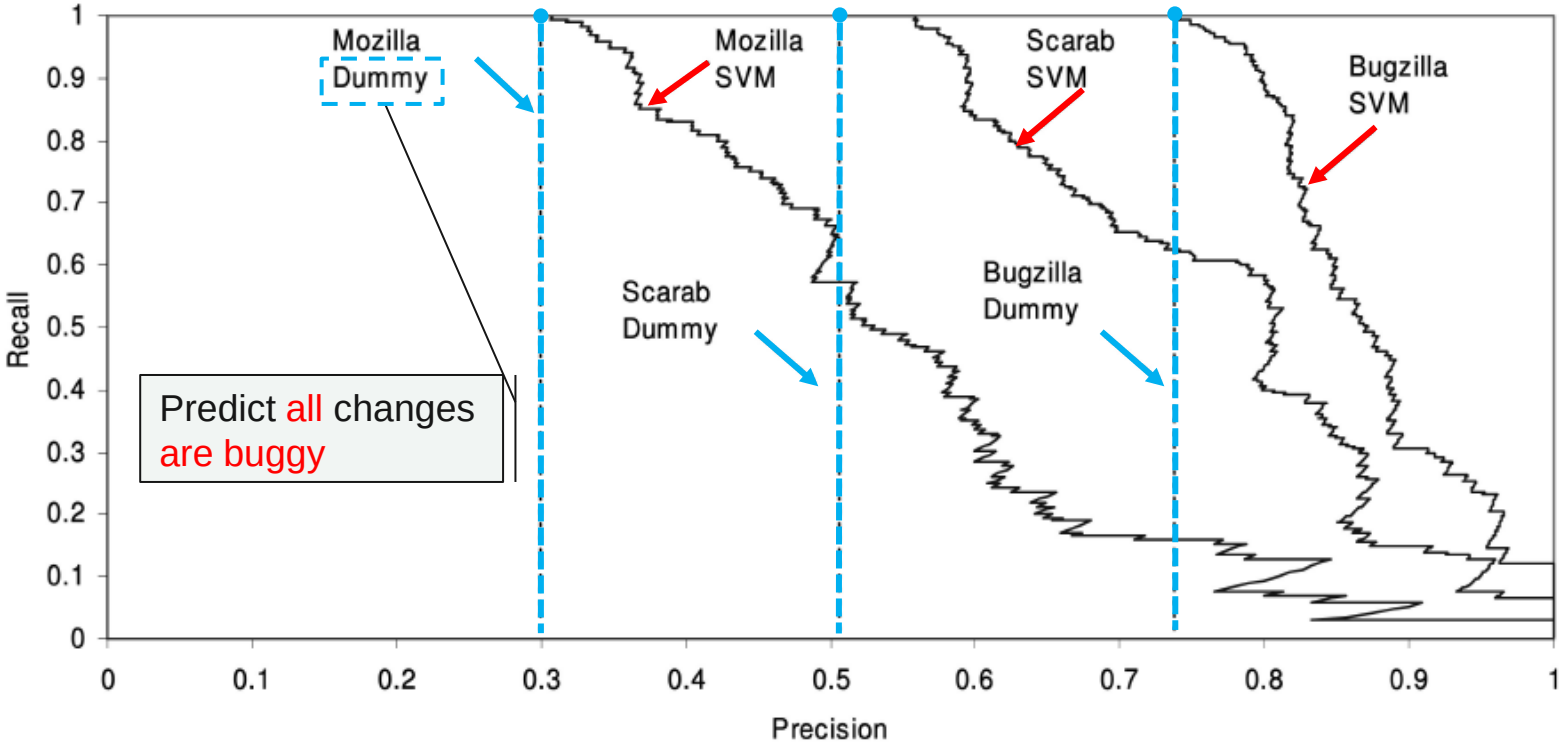
- Assessment of SVM models with metrics
  - Accuracy, recall, precision

Project	Accuracy	Buggy change		Clean change	
		Recall	Precision	Recall	Precision
Apache HTTP 1.3	0.85	0.5	0.56	0.92	0.9
Bugzilla	0.78	0.86	0.85	0.56	0.6
Columba	0.76	0.58	0.59	0.83	0.83
Gaim	0.71	0.63	0.62	0.76	0.77
GForge	0.64	0.6	0.6	0.67	0.67
JEdit	0.65	0.53	0.54	0.73	0.72
Mozilla	0.77	0.57	0.63	0.86	0.83
Eclipse	0.92	0.61	0.61	0.96	0.96
Plone	0.8	0.48	0.49	0.89	0.87
PostgreSQL	0.73	0.43	0.44	0.82	0.82
Scarab	0.79	0.78	0.8	0.8	0.78
Subversion	0.9	0.59	0.6	0.94	0.94
Average	**0.78**	**0.60**	0.61	0.81	0.81

78% average accuracy and 60% average buggy change recall are comparable to the best recent work on bug prediction with small granularity.

Evaluate change classification (3/3)

- Comparison of SVM models and dummy predictor

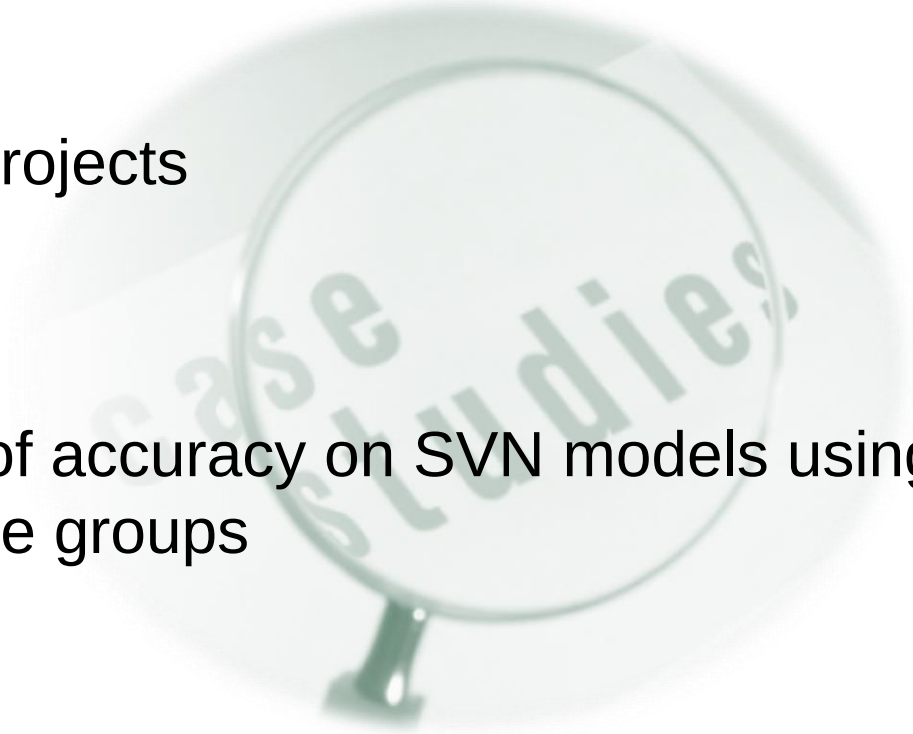


<Buggy change recall-precision curves of the three selected projects>

➡ Dummy classifier is stuck at a certain point.
However SVM can improve the buggy change precision by 20% to 35%.

Evaluate feature groups (1/2)

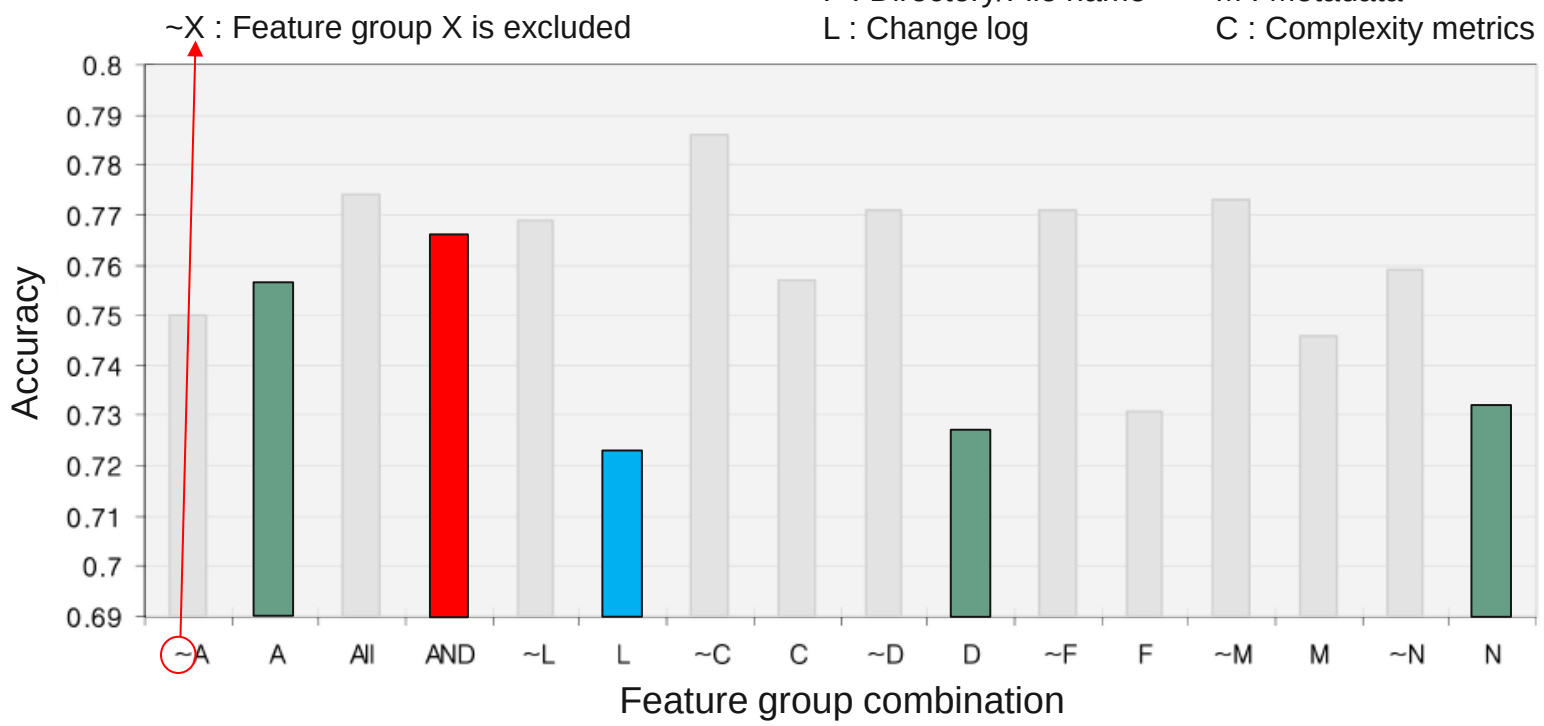
- Goal
  - To evaluate the accuracy of different feature groups
- Target
  - SVM models for 12 projects
- Method
  - Computing average of accuracy on SVN models using combination of feature groups



Evaluate feature groups (2/2)

- Average of accuracy

A : Added delta
D : Deleted delta
F : Directory/File name
L : Change log
N : New revision source code
M : Metadata
C : Complexity metrics



<Average feature group combination accuracy across the 12 projects using SVM>

- Feature combination of source code **>** One feature group of source code
- Change log feature leads to the worst accuracy.

Related work

	Brun and Ernst (ICSE 2004)	Gyimothy et al. (TSE 2005)	This work (TSE 2008)
Goal	Find hidden code errors	Predict faults	Predict latent bugs
Approach	SVM Decision tree	Decision tree Neural networks	SVM
Granularity	Entire source code	Class level (entire files)	A single file
Feature	Invariant information	Object-oriented metrics	Source code Complexity metrics Change metadata Change log
Performance	Accuracy 10.6% Precision 21.6%	Recall 70% Precision 70%	Accuracy 78% Recall 60% Precision 61%

Conclusion

- Contribution
  - Introducing new bug prediction technique
    (File level granularity)
  - Applying new technique for feature extraction
    (BOW+, BOW++)
  - Evaluating change classification's performance
  - Evaluating the features' performance

Discussion

- Pros
  - The prediction granularity is small.
  - It does not require semantic information on source code.
  - It works for various type of projects and languages.
  - It uses a broader set of features.

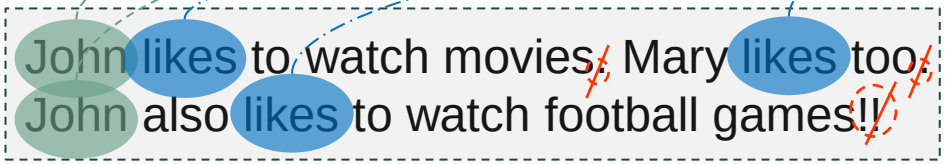
- Cons
  - Target projects are all open source.
  - The bug-introducing data is incomplete.
  - It requires initial change data to train SVM.

Thank You .

KAIST SE LAB
KAIST Software Engineering Laboratory

Appendix

- BOW (Bag-of-words)
  - BOW converts a text into vector model with frequency.
  - It extracts all words except for special characters.



<Sample text>

Feature	Frequency
john	2
likes	3
to	2
watch	2
movie	1
also	1
football	1
games	1
mary	1
too	1

<Vector model>

Appendix

- Machine learning classifier assessment metrics
  - Accuracy
    - What percent of prediction were correct?

$$\frac{n_{b\to b} + n_{c\to c}}{n_{b\to b} + n_{b\to c} + n_{c\to c} + n_{c\to b}}$$

$$\frac{a+d}{a+b+c+d}$$

		Result	
		B	C
Actual	B	a	b
	C	c	d

  - Recall
    - What percent of positive cases were caught?

$$\frac{n_{b\to b}}{n_{b\to b} + n_{b\to c}}$$

$$\frac{a}{a+b}$$

		Result	
		B	C
Actual	B	a	b
	C	c	d

Appendix

- Machine learning classifier assessment metrics (Cont'd)
  - Precision
    - What percent of positive predictions were correct?

$$\frac{n_{b \to b}}{n_{b \to b} + n_{c \to b}}$$

$$\frac{a}{a + c}$$

		Result	
		B	C
Actual	B	a	b
	C	c	d