

Cost-Aware Triage Ranking Algorithms for Bug Reporting Systems

Jin-woo Park¹, Mu-Woong Lee¹, Jinhun Kim¹, Seung-won Hwang¹, Sunghun Kim²

POSTECH, 대한민국¹

HKUST, Hong Kong²

- 1. CosTriage: A Cost-Aware Triage Algorithm for Bug Reporting Systems**, Association for the Advancement of Artificial Intelligence (AAAI), 2011
- 2. CosTriage+: Cost-Aware Triage Ranking Algorithms for Bug Reporting Systems (Submitted)**, Distributed and Parallel Databases (DPD), 2012

□ Bugs!!

- More than 300 bug reports per day in Mozilla (a big software project)

□ Bug Solving

- One of the important issues in a software development process
- Bug reports are posted, discussed, and assigned to developers

□ Open sources projects

- Apache
- Eclipse
- Linux kernel
- Mozilla



Bug reporting systems

□ Bug reports

- Has Bug ID, title, description, status, and other meta data
- Assigned to developers and fixed by developers

□ Challenges

- Bug triage
- Duplicate bug detection
- ...

The screenshot shows a Bugzilla bug report for Eclipse. The bug ID is 199854. The title is "[archives][api][breaking] Improve error reporting for archive handlers". The status is "RESOLVED FIXED". The report was made on 2007-08-14 and modified on 2008-05-07. The bug is assigned to Martin Oberhuber. The description states that the bug was initially created as a clone of Bug #199065 and describes the issue with the ArchiveHandlerManager and ISystemArchiveHandler APIs.

Annotations on the left side of the screenshot:

- bug ID: points to "Bug 199854"
- status: points to "Status: RESOLVED FIXED"
- other data: points to the metadata section (Product, Component, Version, Platform, Importance, Target Milestone, Assigned To, QA Contact, URL, Whiteboard, Keywords, Depends on, Blocks)
- description: points to the main text area containing the bug description

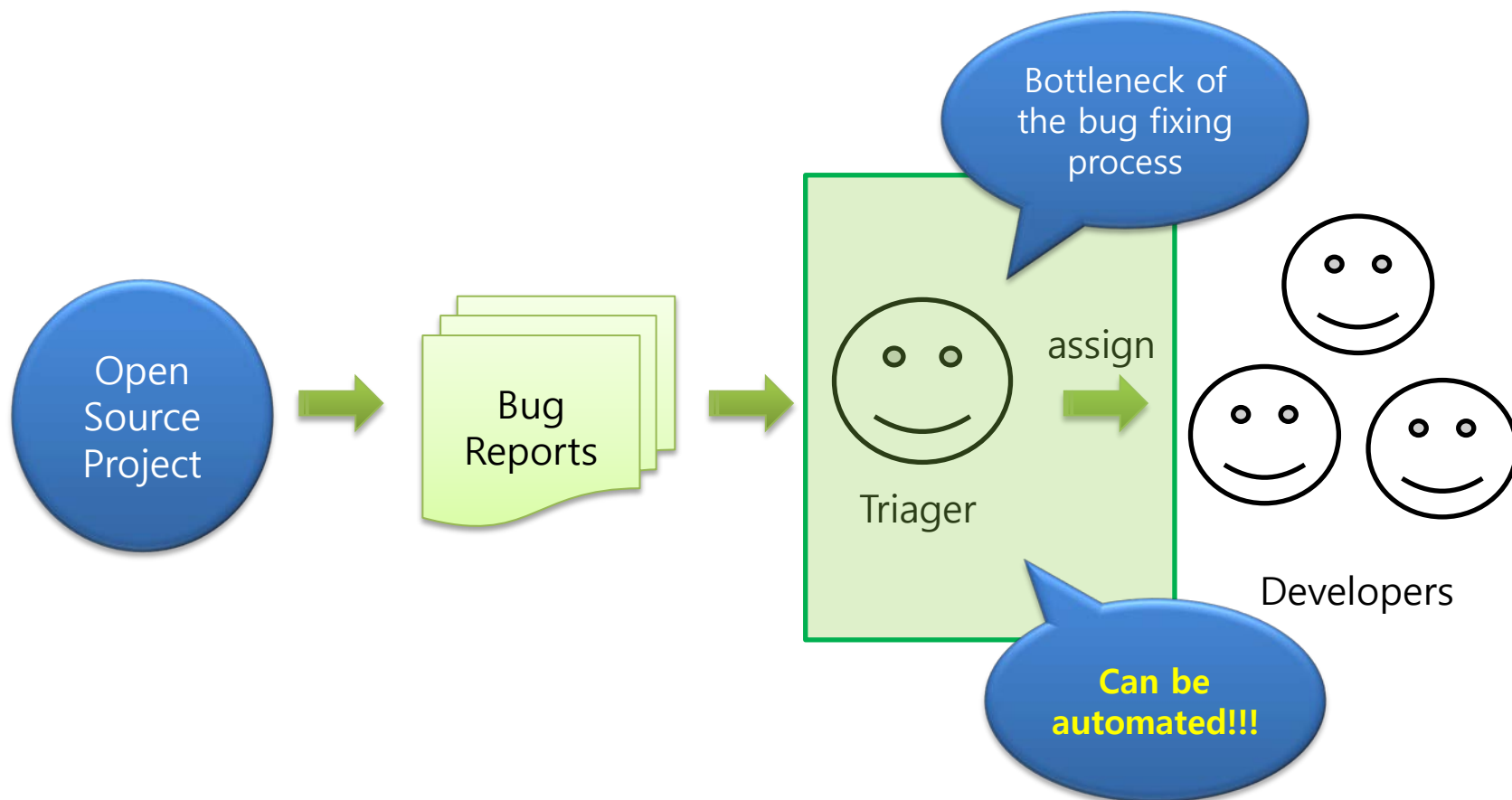
Annotations on the right side of the screenshot:

- title (summary): points to the bug title
- bug fix history (time): points to the "Reported" and "Modified" dates

Bug reporting systems

□ Bug Triage

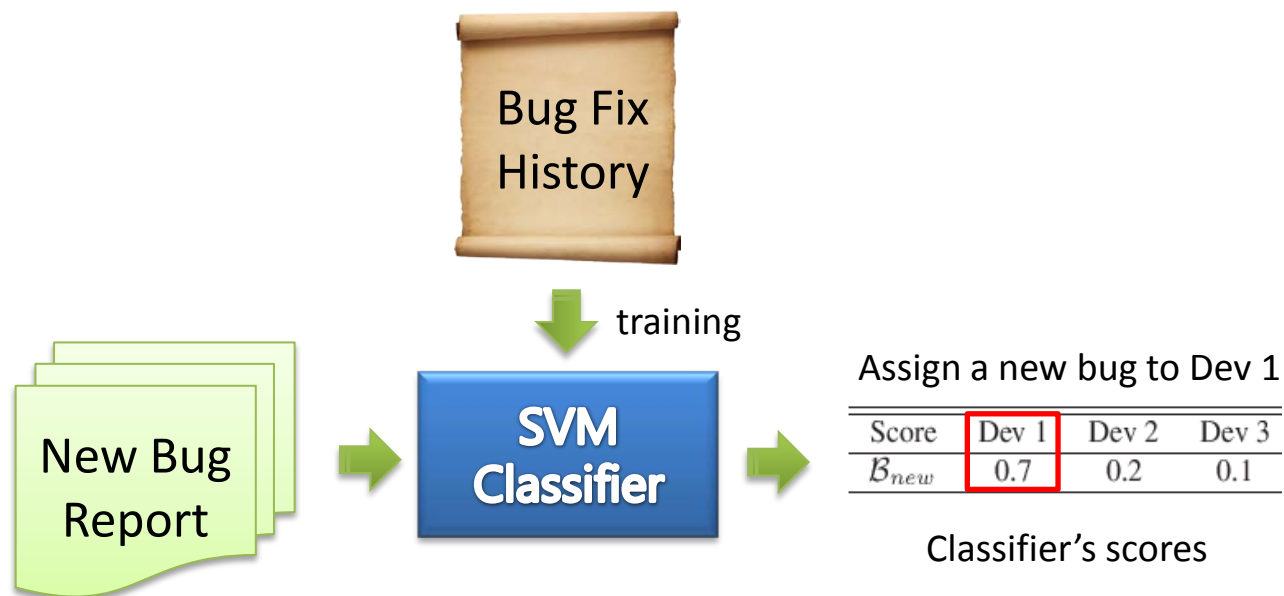
- Assigning a new bug report to a suitable developer
- Bottleneck of bug fixing process
 - Labor intensive
 - Miss-assignment can lead to slow bug fix



Preliminary (Bug triage)

□ PureCBR [Anvik06]

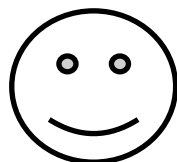
- Construct multi-class classifier using a SVM classifier
- Bug reports B are converted into pair $\langle \text{Word vector}, \text{Dev} \rangle$ for training
 - Bug Report History \rightarrow Input Data
 - Developers \rightarrow Classes



Preliminary (Bug triage)

□ PureCBR [Anvik06]

- Good performances
- Problem
 - This approach only considers accuracy
 - Are developer happy?
 - We consider **developer's cost** (e.g., interests, bug fix time, and expertise)
 - We assume that faster bug fixing time has higher developer cost
 - Each developer has different preference for a bug



50 days



2 days

Preliminary (recommendation)

□ Recommender algorithms

- Content-based recommendation (CBR)
 - Predicting user's interests based on **item features**
 - Machine learning methods
 - Over-specialization problem

- Collaborative filtering recommendation (CF)
 - Predicting user's interests based on **affinity's interests** for items
 - User neighborhood
 - Sparsity problem

- Hybrid recommendation
 - Content-boosted collaborative filtering (CBCF)
 - **Combining** an existing **CBR** with a **CF**
 - Better performance than either approach alone

Preliminary (recommendation)

□ Recommender algorithms

■ Hybrid recommendation

- Content-boosted collaborative filtering (CBCF)
- Combining an existing CBR with a CF
- Better performance than either approach alone

Two Phases

- CBR phase
- CF Phase



Feature	word ₁	word ₂	word ₃
Count	3	2	4

	Bug 1	Bug 2	Bug 3	Bug 4	Bug 5
Dev 1	10	?	?	?	?
Dev 2	?	8	3	?	?
Dev 3	?	?	?	7	?

Preliminary (recommendation)

□ Recommender algorithms

■ Hybrid recommendation

- Content-boosted collaborative filtering (CBCF)
- Combining an existing CBR with a CF
- Better performance than either approach alone

CBR phase



	Bug 1	Bug 2	Bug 3	Bug 4	Bug 5
Dev 1	10	10	10	10	10
Dev 2	3	8	3	8	3
Dev 3	7	7	7	7	7

Preliminary (recommendation)

□ Recommender algorithms

■ Hybrid recommendation

- Content-boosted collaborative filtering (CBCF)
- Combining an existing CBR with a CF
- Better performance than either approach alone

CF phase



	Bug 1	Bug 2	Bug 3	Bug 4	Bug 5
Dev 1	10	9	7	9	8
Dev 2	5	8	3	8	5
Dev 3	7	8	5	7	6

Existing recommendation approaches are not suitable!

Goal

□ Goal

- Find efficient bug-developer matching
 - Optimizing not only **accuracy** but also **cost**
- Use modified CBCF approach
 - Constructing **developer profiles** for **cost**

□ Challenge

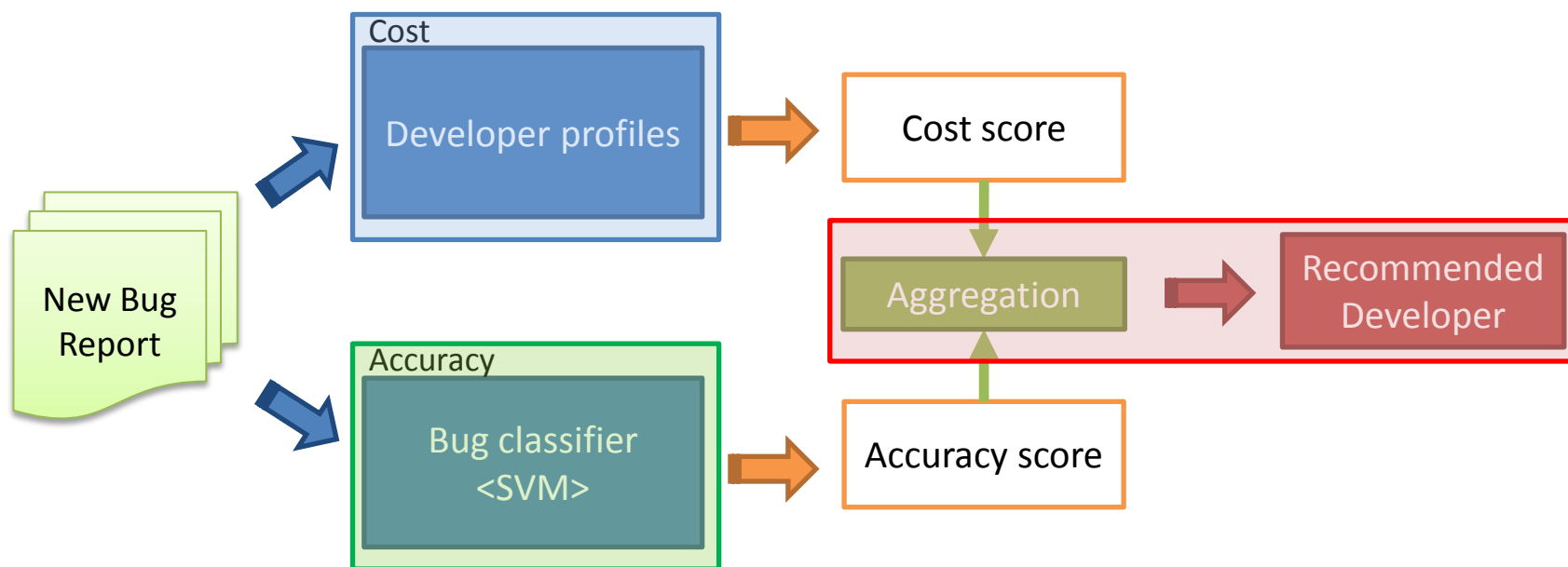
- Enhancing CBCF approach for sparse data
- **Extreme sparseness** of the past bug fix history data
 - A bug fixed by a developer
 - Need to reduce sparseness for enhancing quality of CBCF

Dev	Bug						
	\mathcal{B}_1	\mathcal{B}_2	\mathcal{B}_3	\mathcal{B}_4	\mathcal{B}_5	\mathcal{B}_6	\mathcal{B}_7
\mathcal{D}_1	2.3	3.7	-	-	10.5	-	-
\mathcal{D}_2	-	-	-	12.7	-	-	-
\mathcal{D}_3	-	-	30.1	-	-	2.9	-
\mathcal{D}_4	-	-	-	-	-	-	7.1

Bug fix time from bug fix history

Overview

- **Merging classifier's scores and developer's cost scores.**
 - **The accuracy scores** are obtained using **PureCBR** [Anvik06]
 - **The developer cost scores** are obtained from “de-sparsified” bug fix history.
 - Two scores are then merged for prediction



CosTriage (Cost Estimation)

- CosTriage: A **Cost**-aware **Triage** Algorithm for bug reporting system
- Challenge to estimate the developer cost?
 - How to reduce the sparseness problem?
 - Using a Topic Modeling

Dev	Bug						
	B_1	B_2	B_3	B_4	B_5	B_6	B_7
D_1	2.3	3.7	-	-	10.5	-	-
D_2	-	-	-	12.7	-	-	-
D_3	-	-	30.1	-	-	2.9	-
D_4	-	-	-	-	-	-	7.1

Bug fix time from bug fix history



Dev	Bug Types	
	1 (B_1, B_3, B_4)	2 (B_2, B_5, B_6, B_7)
D_1	2.3	7.1
D_2	12.7	-
D_3	30.1	2.9
D_4	-	7.1

Categorization bugs to reduce the sparseness

CosTriage

□ Categorizing bugs

■ Topic Modeling

- Latent Dirichlet Allocation (LDA) [BleiNg03]
- Each topic is represented as a bug type
- The topic distribution of reports determine bug types

■ We adopt the divergence measure proposed in [Arun, R. PAKDD '10]

- Finding the natural number of topics (# bug types)

$$\mathcal{T} = \underset{t}{\operatorname{argmin}} KL_Divergence(t)$$

t is the natural number of bug types

Table 1: An LDA model of bug reports for Mozilla. $\mathcal{T} = 7$. Top-10 representative words with the highest probabilities.

	Topic 1	Topic 2	Topic 3	Topic 4	Topic 5	Topic 6	Topic 7						
c	0.058	mozilla	0.047	js	0.028	window	0.023	html	0.026	windows	0.042	bug	0.036
line	0.058	firefox	0.024	error	0.018	page	0.019	text	0.018	mozilla	0.042	bugzilla	0.012
cpp	0.040	add	0.014	mozilla	0.018	click	0.016	table	0.013	gecko	0.029	cgi	0.009
int	0.032	version	0.010	file	0.014	menu	0.014	document	0.011	rv	0.028	code	0.008
mozilla	0.032	update	0.010	test	0.011	button	0.013	image	0.010	results	0.025	id	0.008
src	0.026	thunderbird	0.010	function	0.011	open	0.013	style	0.010	user	0.023	bugs	0.008
const	0.023	file	0.008	ns	0.010	text	0.012	content	0.009	build	0.020	time	0.007
unsigned	0.020	files	0.008	c	0.010	dialog	0.010	page	0.009	message	0.019	patch	0.007
bytes	0.019	added	0.007	chrome	0.009	select	0.009	type	0.009	nt	0.019	set	0.005
builds	0.018	install	0.007	content	0.009	search	0.009	id	0.009	firefox	0.018	fix	0.005

CosTriage

□ Developer profiles modeling

■ Developer Profiles

□ N-dimensional feature vector

□ The element of developer profiles, $P_u[i]$, denotes the developer cost for i th-type bugs

$$\mathcal{P}_u = \langle p_{u[1]}, p_{u[2]}, \dots, p_{u[\mathcal{T}]} \rangle$$

■ Developer Cost

□ The average time to fix i^{th} type bugs

Dev	Bug types						
	1	2	3	4	5	6	7
\mathcal{D}_1	7.73	1.71	8.59	14.28	7.54	5.44	8.45
\mathcal{D}_2	8.18	-	3.50	1.75	4.00	12.90	13.18
\mathcal{D}_3	11.56	-	60.50	23.50	-	2.67	19.20
\mathcal{D}_4	-	-	-	-	-	22.40	20.75


CosTriage

□ Predicting missing values in profiles

■ Using CF for developer profiles

□ Similarity measure:

$$S(\mathcal{P}_u, \mathcal{P}_v) = \frac{\mathcal{P}_u \cdot \mathcal{P}_v}{\|\mathcal{P}_u\| \|\mathcal{P}_v\|} \times \omega(\mathcal{P}_u, \mathcal{P}_v)$$

k=1 

Dev	Bug types						
	1	2	3	4	5	6	7
\mathcal{D}_1	7.73	1.71	8.59	14.28	7.54	5.44	8.45
\mathcal{D}_2	8.18	-	3.50	1.75	4.00	12.90	13.18
\mathcal{D}_3	11.56	-	60.50	23.50	-	2.67	19.20
\mathcal{D}_4	-	-	-	-	-	22.40	20.75



$$p_{u[i]} = F(\mathcal{P}_u) \times \frac{\sum_{\forall \mathcal{P}_v \in N_u} S(\mathcal{P}_u, \mathcal{P}_v) \cdot \left(\frac{p_{v[i]}}{F(\mathcal{P}_v)}\right)}{\sum_{\forall \mathcal{P}_v \in N_u} S(\mathcal{P}_u, \mathcal{P}_v)}$$

Dev	Bug types						
	1	2	3	4	5	6	7
\mathcal{D}_1	7.73	1.71	8.59	14.28	7.54	5.44	8.45
\mathcal{D}_2	8.18	6.40	3.50	1.75	4.00	12.90	13.18
\mathcal{D}_3	11.56	27.22	60.50	23.50	40.52	2.67	19.20
\mathcal{D}_4	13.18	12.99	10.97	11.41	15.14	22.40	20.75

□ Obtaining developer's cost for a new bug report

Bug type = 1



Bug	Keyword			D
	w_1	w_2	w_3	
B_{new}	5	0	1	?



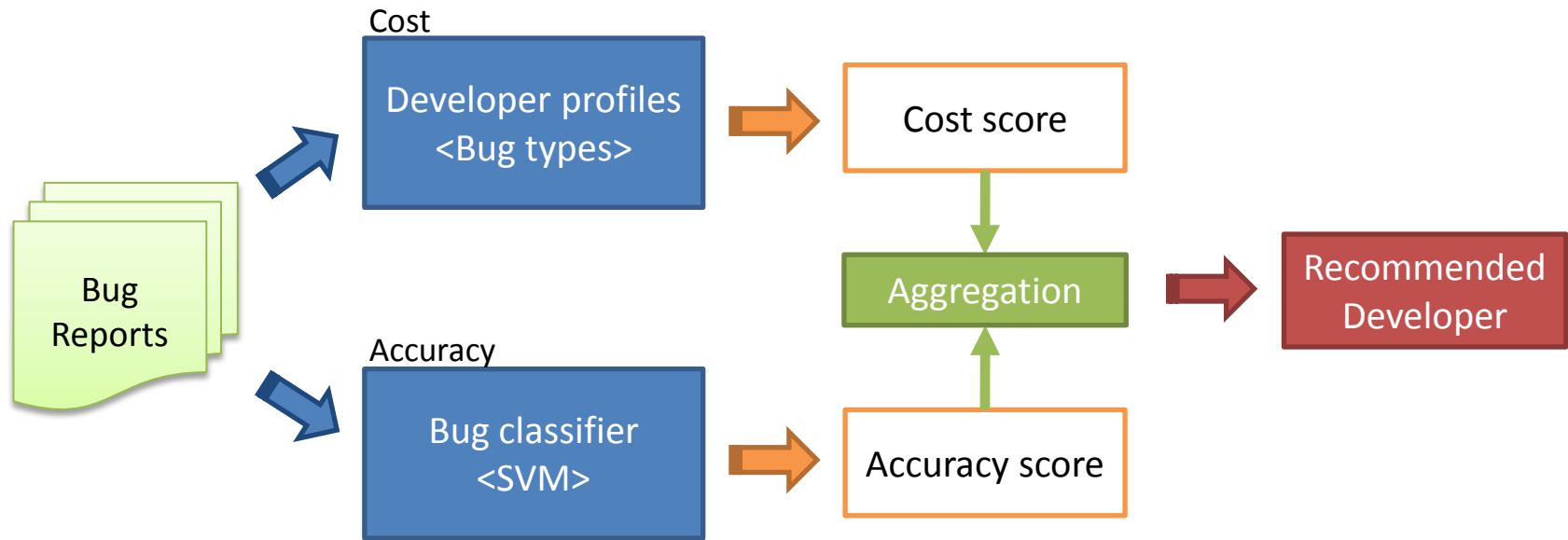
	Topic 1	Topic 2	Topic 3	Topic 4	Topic 5				
c	0.058	mozilla	0.047	js	0.028	window	0.023	html	0.026
line	0.058	firefox	0.024	error	0.018	page	0.019	text	0.018
cpp	0.040	add	0.014	mozilla	0.018	click	0.016	table	0.013
int	0.032	version	0.010	file	0.014	menu	0.014	document	0.011
mozilla	0.032	update	0.010	test	0.011	button	0.013	image	0.010
src	0.026	thunderbird	0.010	function	0.011	open	0.013	style	0.010
const	0.023	file	0.008	ns	0.010	text	0.012	content	0.009
unsigned	0.020	files	0.008	c	0.010	dialog	0.010	page	0.009
bytes	0.019	added	0.007	chrome	0.009	select	0.009	type	0.009
builds	0.018	install	0.007	content	0.009	search	0.009	id	0.009

Dev	Bug types						
	1	2	3	4	5	6	7
D_1	7.73	1.71	8.59	14.28	7.54	5.44	8.45
D_2	8.18	6.40	3.50	1.75	4.00	12.90	13.18
D_3	11.56	27.22	60.50	23.50	40.52	2.67	19.20
D_4	13.18	12.99	10.97	11.41	15.14	22.40	20.75

Developer cost for a new bug

CosTriage

□ Merging classifier's scores and developer's cost scores.



Accuracy scores [Anvik06]	+	Cost scores (CosTriage)	=	Hybrid scores																								
<table border="1" style="border-collapse: collapse; text-align: center;"> <tr><td>Bug</td><td>Dev 1</td><td>Dev 2</td><td>Dev 3</td></tr> <tr><td>\mathcal{B}_{new}</td><td>0.7</td><td>0.2</td><td>0.1</td></tr> </table>	Bug	Dev 1	Dev 2	Dev 3	\mathcal{B}_{new}	0.7	0.2	0.1		<table border="1" style="border-collapse: collapse; text-align: center;"> <tr><td>Bug</td><td>Dev 1</td><td>Dev 2</td><td>Dev 3</td></tr> <tr><td>\mathcal{B}_{new}</td><td>0.3</td><td>0.4</td><td>0.3</td></tr> </table>	Bug	Dev 1	Dev 2	Dev 3	\mathcal{B}_{new}	0.3	0.4	0.3		<table border="1" style="border-collapse: collapse; text-align: center;"> <tr><td>Bug</td><td>Dev 1</td><td>Dev 2</td><td>Dev 3</td></tr> <tr><td>\mathcal{B}_{new}</td><td style="border: 2px solid blue;">1.0</td><td>0.6</td><td>0.4</td></tr> </table>	Bug	Dev 1	Dev 2	Dev 3	\mathcal{B}_{new}	1.0	0.6	0.4
Bug	Dev 1	Dev 2	Dev 3																									
\mathcal{B}_{new}	0.7	0.2	0.1																									
Bug	Dev 1	Dev 2	Dev 3																									
\mathcal{B}_{new}	0.3	0.4	0.3																									
Bug	Dev 1	Dev 2	Dev 3																									
\mathcal{B}_{new}	1.0	0.6	0.4																									

□ Limitations of CosTriage

- CosTriage cannot determine the bug types of some bug reports
 - The report does not include any topic word identified by LDA model
 - 4.04% bug reports in Mozilla projects
 - CosTriage determines the bug types for the reports randomly.

- CosTriage does not consider temporal characteristic on modeling developer profiles
 - Several studies show that a user's interests do change over time
 - The recent bug fix history is more important than the older history

Overcoming the limitations of CosTriage

■ Code-information-based type prediction approach

- To determine the types of undetermined bug the reports
- Using a set of classes of libraries imported in the code

■ Modeling developer profile changes over time

- To reduce the weight of bug fix history with a rate proportional to a period time
- Using exponential decay model

□ Code-information-based type prediction approach

\mathcal{B}_1 :

```
org.eclipse.swt.graphics.Image,  
org.eclipse.swt.widgets,  
org.eclipse.ui.dialogs
```

\mathcal{B}_2 :

```
org.eclipse.swt.graphics,  
org.eclipse.swt.widgets.Display,  
org.eclipse.ui.internal
```

■ Using a set of classes or libraries imported in the code

□ Set similarity

$$\text{Similarity}(S(\mathcal{I}_{\mathcal{B}_1}), S(\mathcal{I}_{\mathcal{B}_2})) = \frac{|S(\mathcal{I}_{\mathcal{B}_1}) \cap S(\mathcal{I}_{\mathcal{B}_2})|}{|S(\mathcal{I}_{\mathcal{B}_1}) \cup S(\mathcal{I}_{\mathcal{B}_2})|},$$

□ Tree similarity (tree edit distance)

$$\text{Similarity}(T(\mathcal{I}_{\mathcal{B}_1}), T(\mathcal{I}_{\mathcal{B}_2})) = 1 - \frac{\delta(T(\mathcal{I}_{\mathcal{B}_1}), T(\mathcal{I}_{\mathcal{B}_2}))}{|T(\mathcal{I}_{\mathcal{B}_1})| + |T(\mathcal{I}_{\mathcal{B}_2})|},$$

CosTriage+

□ Code-information-based type prediction approach

\mathcal{B}_1 :

`org.eclipse.swt.graphics.Image,`
`org.eclipse.swt.widgets,`
`org.eclipse.ui.dialogs`

\mathcal{B}_2 :

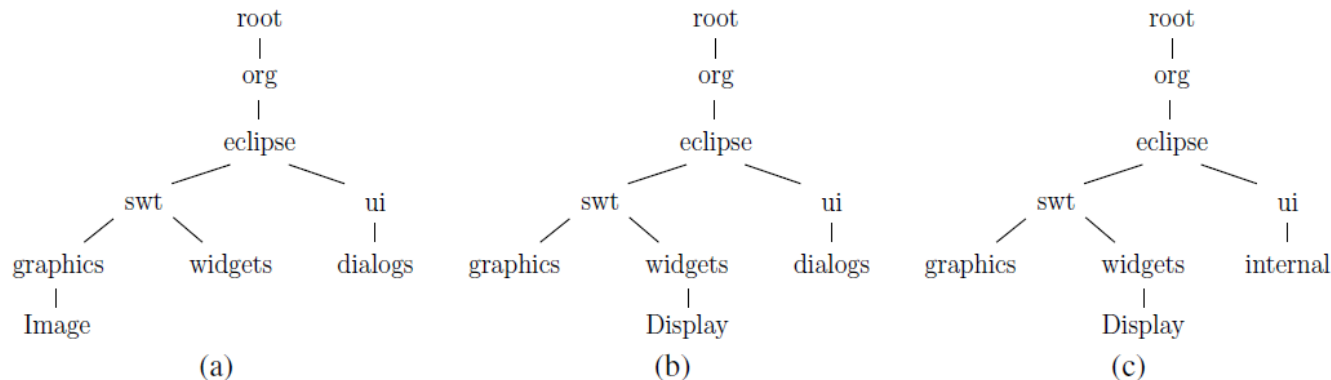
`org.eclipse.swt.graphics,`
`org.eclipse.swt.widgets.Display,`
`org.eclipse.ui.internal`

■ Using a set of classes or libraries imported in the code

□ Set similarity

$$\text{Similarity}(S(\mathcal{I}_{\mathcal{B}_1}), S(\mathcal{I}_{\mathcal{B}_2})) = \frac{|S(\mathcal{I}_{\mathcal{B}_1}) \cap S(\mathcal{I}_{\mathcal{B}_2})|}{|S(\mathcal{I}_{\mathcal{B}_1}) \cup S(\mathcal{I}_{\mathcal{B}_2})|}$$

□ Tree similarity (tree edit distance)



CosTriage+

□ Code-information-based type prediction approach

\mathcal{B}_1 :

```
org.eclipse.swt.graphics.Image,
org.eclipse.swt.widgets,
org.eclipse.ui.dialogs
```

\mathcal{B}_2 :

```
org.eclipse.swt.graphics,
org.eclipse.swt.widgets.Display,
org.eclipse.ui.internal
```

■ Using a set of classes or libraries imported in the code

□ Set similarity

$$\text{Similarity}(S(\mathcal{I}_{\mathcal{B}_1}), S(\mathcal{I}_{\mathcal{B}_2})) = \frac{|S(\mathcal{I}_{\mathcal{B}_1}) \cap S(\mathcal{I}_{\mathcal{B}_2})|}{|S(\mathcal{I}_{\mathcal{B}_1}) \cup S(\mathcal{I}_{\mathcal{B}_2})|},$$

□ Tree similarity (tree edit distance)

$$\text{Similarity}(T(\mathcal{I}_{\mathcal{B}_1}), T(\mathcal{I}_{\mathcal{B}_2})) = 1 - \frac{\delta(T(\mathcal{I}_{\mathcal{B}_1}), T(\mathcal{I}_{\mathcal{B}_2}))}{|T(\mathcal{I}_{\mathcal{B}_1})| + |T(\mathcal{I}_{\mathcal{B}_2})|},$$

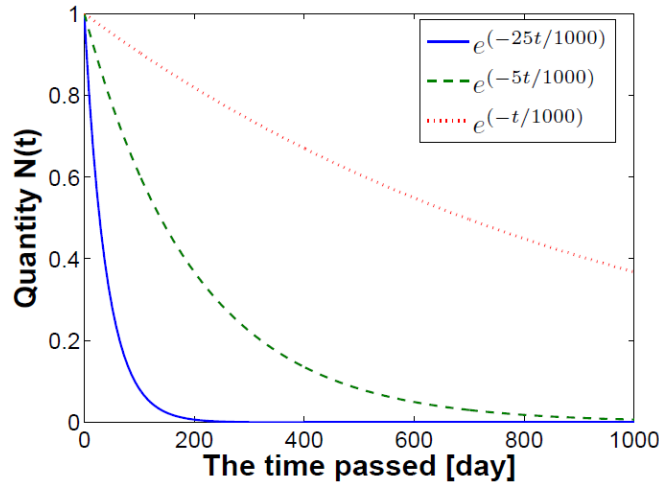
$$\text{score}[i] = \sum_{\forall \mathcal{B}_j^i \in T_k(\mathcal{B})} \text{Similarity}(\mathcal{I}_{\mathcal{B}}, \mathcal{I}_{\mathcal{B}_j^i}),$$

□ Modeling developer profile changes over time

■ Exponential decay

- Reducing weight of bug fix history with a rate proportional to a period time
- The quantity N decreases according to the following law:

$$N(t) = N_0 e^{-\lambda t}$$



- Quantifying developer's cost for i th-type bugs as the weight of bug history using exponential decay

$$p_{u[i]} = \frac{1}{\sum_{\forall \mathcal{B}_j \in \mathcal{B}_{u[i]}} N(t_{\mathcal{B}_j}) \cdot \overset{\text{bug fix time}}{t_{\mathcal{B}_j}}}$$

weight

□ Subject Systems

- 97,910 valid bug reports
- 255 active developers
- From four open source projects

Projects	# Fixed bug reports	# Valid bug reports	# Total developers	# Active developers	# Bug types	# Words	Period
Apache	13,778	656	187	10	19	6,915	2001-01-22 - 2009-02-09
Eclipse	152,535	47,862	1,116	100	17	61,515	2001-10-11 - 2010-01-22
Linux kernel	5,082	968	270	28	7	11,252	2002-11-14 - 2010-01-16
Mozilla	162,839	48,424	1,165	117	7	71,878	1998-04-07 - 2010-01-26

□ Approaches

- **PureCBR**: State of the art CBR-based approach
- **CBCF**: Original CBCF
- **CosTraige**
- **CosTraige+**

□ Two research questions

Q1. How much can our approach **improve cost** (bug fix time) without sacrificing bug assignment accuracy?

Q2. What are the **trade-offs** between accuracy and cost (bug fix time)?

□ Evaluation measures

$$Accuracy = \frac{|W|}{|N|}$$

$$\text{Average bug fix time} = \frac{\sum_{\forall w_i \in W} \{\text{bug fix time of } w_i\}}{|W|}$$

W is the set of bug reports predicted correctly.

N is the number of bug reports in the test set.

- The real fix time is unknown, we only use the fix time for correctly matched bugs.

Relative errors of expected bug fix time

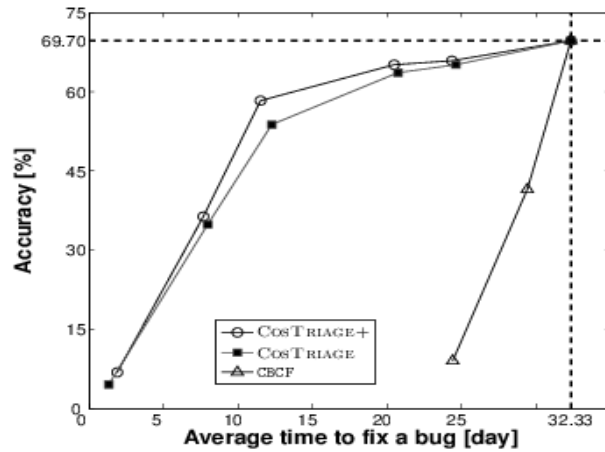
	Apache	Eclipse	Linux	Mozilla
CBCF	99.38	26.41	71.76	21.42
CosTRIAGE	57.69	25.88	46.77	20.89
CosTRIAGE+	50.45	25.61	40.67	19.73

Improvement of bug fix time (Q1)

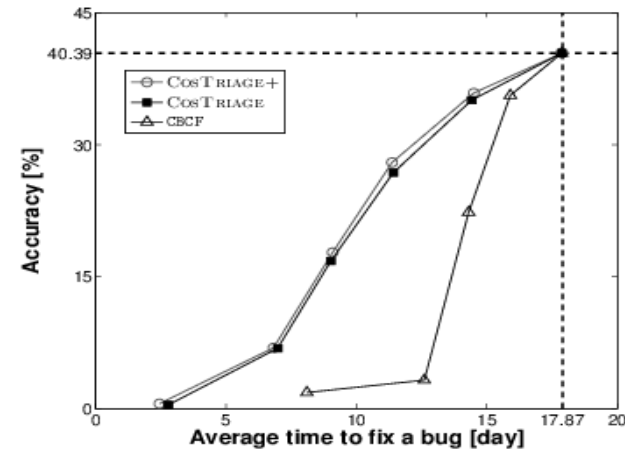
Project	PureCBR		Reducing ratio					
	Time	Acc	CBCF		CosTRIAGE		CosTRIAGE+	
	Time	Acc	Time	Acc	Time	Acc	Time	Acc
Apache	32.33	69.70	-2.28%	-5.43%	-30.88%	-5.43%	-31.49%	-5.43%
Eclipse	17.87	40.39	-5.59%	-5.04%	-10.38%	-5.04%	-10.68%	-5.04%
Linux	55.25	30.93	-24.83%	-5.00%	-28.94%	-5.00%	-31.44%	-5.00%
Mozilla	11.34	64.29	-4.79%	-5.01%	-7.21%	-5.01%	-7.74%	-5.01%

- **CosTriage+** improves the costs efficiently up to 31.49% without seriously compromising accuracy

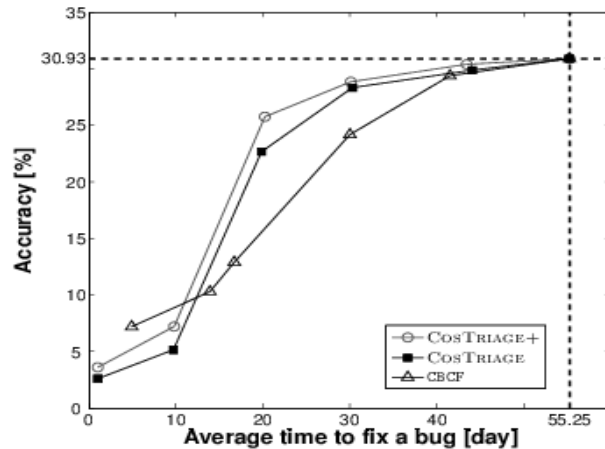
Trade-off between accuracy and bug fix time (Q2)



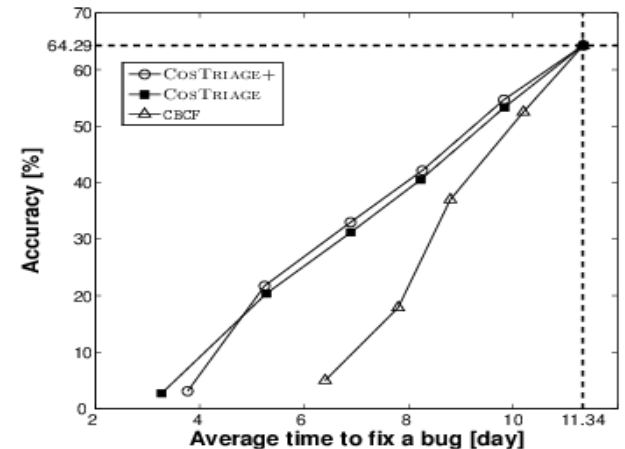
(a) Apache



(b) Eclipse



(c) Linux kernel



(d) Mozilla

Experiments

□ Using code information for type prediction

- 558 Mozilla reports (which have code)
- Models
 - Random
 - n-gram
 - E.g., “documents” (“document” in LDA model)
 Document → d, do, doc, docu, ocum, cume, umen, ment, ent, nt, t
 Documents → d, do, doc, docu, ocum, cume, umen, ment, ents, nts, ts, s
 - Code-information-based approach
 - Set similarity
 - Tree similarity

Model	Precision
Random	33/558 (5.91%)
n-gram (n = 3)	81/558 (14.52%)
n-gram (n = 4)	85/558 (15.23%)
n-gram (n = 5)	83/558 (14.87%)
Set similarity	129/558 (23.29%)
Tree similarity	125/558 (22.40%)

Conclusion

- We proposed a new bug triaging technique
 - Optimize not only accuracy but also cost
 - Solve data sparseness problem by using topic modeling

- We enhanced the approach
 - Enlarging coverage of bug types
 - Modeling developer profiles changes over time

- Experiments using four real bug report corpora
 - Improve the cost without heavy losses of accuracy

Thank you!

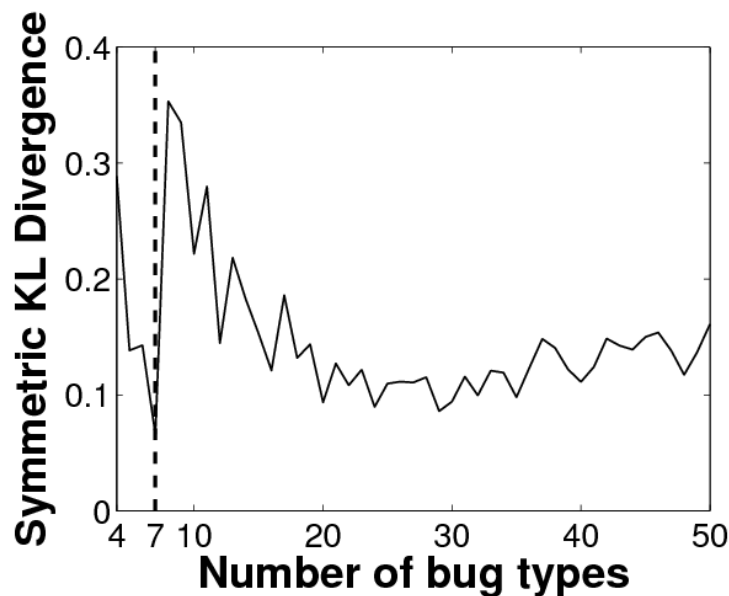
Do you have any questions?

□ We adopt the divergence measure proposed in [Arun10]

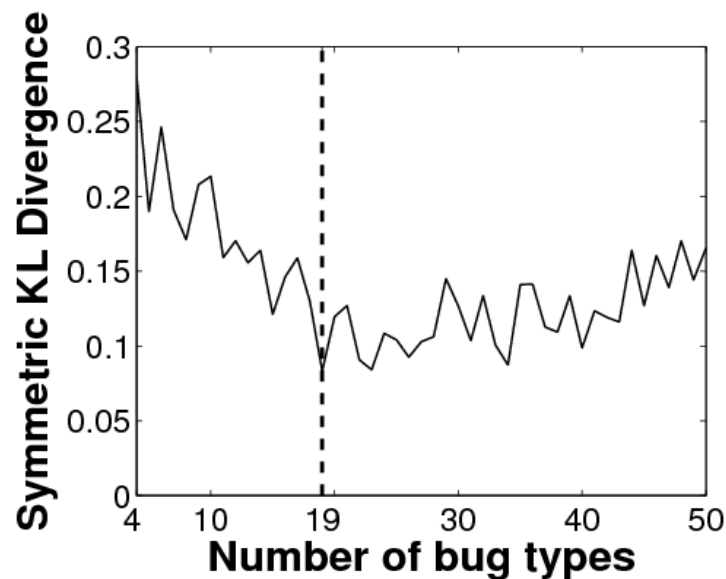
- Finding the natural number of topics (# bug types)

$$\mathcal{T} = \underset{t}{\operatorname{argmin}} KL_Divergence(t)$$

t is the natural number of bug types



Mozilla



Apache

Back up - Bug features

□ Bug features

- Keywords of title and description
- Other meta data



Title : Traditional Memory Rendering refactoring request
 Description : Request additional refactoring so we can
 ...
 Traditional Rendering.



Remove stopwords

Traditional Memory Rendering refactoring request Request
 refactoring ... Traditional Rendering



Bug	Keyword			D
	w_1	w_2	w_3	
B_{new}	5	0	1	?