Cost-Aware Triage Ranking Algorithms for Bug Reporting Systems

Jin-woo Park[1], Mu-Woong Lee[1], Jinhan Kim[1], Seung-won Hwang[1], Sunghun Kim[2]
[1]Pohang University of Science and Technology (POSTECH), Republic of Korea
[2]Hong kong University of Science and Technology (HKUST), Hong Kong
[1]{jwpark85, sigliel, ,deepbrother swhwang}@postech.ac.kr
[2]hunkim@cse.ust.hk

Introduction

Fig. 1 : A bug report of Eclipse bug reporting system

- **Bug**
  - A common term for **an error, flaw, mistake, failure, or fault** in a computer program or system
  - Occurs **unexpected results**

- **Bug Reporting System**
  - To post, discuss, and assign bug reports to developers
  - More than 300 reports per day in Mozilla
  - Open source projects have their bug reporting system (e.g., Apache, Eclipse, Mozilla, …)

- **Bug Triage**
  - **Assigning the bugs** to a suitable developer
  - Bottleneck of bug fixing process
    - Labor intensive
    - Miss-assignment makes to slow bug fix process

Bug Triage Techniques

- PureCBR [Anvik 06]
  - Using multi-class SVM classifier
    - Feature: keyword vector of the bug reports
  - Focusing on **accuracy**

- CosTriage [Park 11]
  - Considering both **accuracy and cost**
    - **PureCBR** is adopted for **accuracy**
    - **Recommender algorithm** is used for **cost**
  - Key challenge
    - Since **the bug fix history** is **extremely sparse**, **recommender algorithm cannot be adopted directly.**
  - CosTriage solved the challenge
    - Categorizing the bug types using topic modeling approach LDA

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

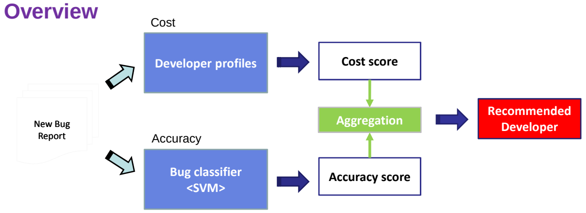
➡

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

    - Using **Collaborative filtering (CF)** for the **remained missing values**
    - **Ranking the developers** by the aggregated scores (acc + cost)

Approach: CosTriage+

- **Overview**



Bug	Dev 1	Dev 2	Dev 3
B_{new}	0.7	0.2	0.1

Accuracy scores

Bug	Dev 1	Dev 2	Dev 3
B_{new}	0.3	0.4	0.3

Cost scores

=

Bug	Dev 1	Dev 2	Dev 3
B_{new}	1.0	0.6	0.4

Hybrid scores

- **The accuracy scores** are obtained using **PureCBR** [Anvik06]
- **The developer cost scores** are obtained using the enhanced **CBCF**
- **Two scores are then merged** for **ranking developers**

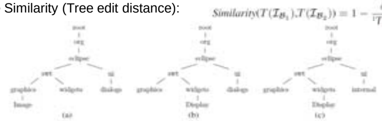
- **Improvements**

  - **Bug type prediction** using **code information**
    - To determine the types of undetermined bug the reports (4.04% in Mozilla)
    - **The code similarity** using the **import paths** in the codes of the bugs

B_1:
```
org.eclipse.swt.graphics.Image,
org.eclipse.swt.widgets,
org.eclipse.ui.dialogs
```

B_2:
```
org.eclipse.swt.graphics,
org.eclipse.swt.widgets.Display,
org.eclipse.ui.internal
```

1. Set Similarity (Jaccard coefficient): $Similarity(S(\mathcal{I}_{B_1}), S(\mathcal{I}_{B_2})) = \frac{|S(\mathcal{I}_{B_1}) \cap S(\mathcal{I}_{B_2})|}{|S(\mathcal{I}_{B_1}) \cup S(\mathcal{I}_{B_2})|}$,
2. Tree Similarity (Tree edit distance): $Similarity(T(\mathcal{I}_{B_1}), T(\mathcal{I}_{B_2})) = 1 - \frac{\delta(T(\mathcal{I}_{B_1}), T(\mathcal{I}_{B_2}))}{|T(\mathcal{I}_{B_1})| + |T(\mathcal{I}_{B_2})|}$,



    - **Determining bug type** from **the Top-k most similar bugs**

$$\operatorname*{argmax}_i \; score[i] = \sum_{\forall B'_j \in T_k(B)} Similarity(\mathcal{I}_B, \mathcal{I}_{B'_j}),$$

  - Modeling developer profile changes over time
    - **To reduce the weight of history** with a rate proportional to **a period time**
    - Quantifying developer's cost for *i* th-type bugs as the weight of bug history **using exponential decay**:

$$p_{u[i]} = \frac{1}{\Sigma_{\forall B_j \in B_{u[i]}} N(t_{B_j})} \Sigma_{\forall B_j \in B_{u[i]}} (N(t_{B_j}) \cdot tf_{B_j}) \qquad N(t) = N_0 e^{-\lambda t}$$

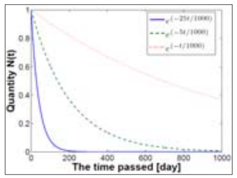


Fig. 2: Example of weight of bug fix history undergoing exponential decay for 1,000 days.

Experiments

- **Dataset**
  - We used the bug reports from 4 bug reporting system

Table 1: Subject systems

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

- **Experimental Results**
  - Precision of prediction for bug types

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 (k = 10)	130/558 (23.30%)
Set similarity (k = 15)	**135/558 (24.19%)**
Set similarity (k = 20)	131/558 (23.48%)
Tree similarity (k = 10)	**138/558 (24.73%)**
Tree similarity (k = 15)	132/558 (23.66%)
Tree similarity (k = 20)	128/558 (22.94%)

  - Absolute 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

	PureCBR		Reducing ratio: CBCF		CosTriage		CosTriage+	
Project	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%

Conclusion

- **We proposed a new bug triaging technique**
  - Optimize not only accuracy but also cost
  - Solve data sparseness problem by using topic modeling
- **We solved the limitations of COSTRIAGE**
  - Enlarging coverage of bug types
  - Modeling developer profiles changes over time
- **Experiments using four real bug report corpora**
  - We conducted the experiments with bug reports from real bug corpora

[Anvik 06] J. Anvik, L. Hiew, G. C. Murphy, Who should fix this bug?, ICSE, 2006.

[Park 11] Park, M.-W. Lee, J. Kim, S. Hwang, S. Kim, Costriage: A cost-aware triage algorithm for bug reporting systems, AAAI, 2011.