CMSuggester: Method Change Suggestion to Complement Multi-Entity Edits

Presented By

Sheikh Shadab Towqir

CS 6704

Introduction

- Developers spend a lot of time and resources for the maintenance of software.
- 70% of time and resources is spent in maintenance to fix bugs, add features, or refactor code [1].
- Bug fixes:
  - Around 80% of real bugs are fixed by editing multiple program locations together (multi-entity edits) [2]
  - Over half the maintenance issues are related to bug fixes [3]
- A **multi-entity edit** is a program commit that simultaneously changes multiple entities.

Problem Statement

- It is challenging to always apply multi-entity edits consistently and completely.

- Developers sometimes fail to identify all the edit locations relevant to a bug [4].

- Limitations with existing tools:
  - **Version history to mine rules**: Syntactic or semantic relationships between co-changed entities are not considered and have demonstrated low accuracies of suggestion [5, 6].
  - **Textual diff to infer structural differences**: Only systematic additions and deletions are considered (not changes) [7].
  - **Program transformation from similar edits:** Does not help if distinct edits should be co-applied to dissimilar methods [8].

Proposed Tool

- CMSuggester has been developed to suggest complementary changes for multi-entity edits.

- Focuses purely on suggestion for ***CM → AF** edits.
  - CM: Changed Methods
  - AF: Added Fields

- ***CM → AF** edits are when:
  - A field experiences an AF change
  - One or more methods accessing the field experience CM changes.

Proposed Tool

- CMSuggester works by identifying **peer fields** (common fields) from the added field $\boldsymbol{f_n}$.
  - Example: Fields that are accessed by the changed methods

- Uses several filtering steps to identify and generate this list of peer fields:
  - Location
  - Name Similarity
  - Access Type

- These peer fields are then used to suggest edits to unchanged methods that also access these fields.

Motivating Example

- The field **_clobValue** was added.
- The method **getLength()** along with 11 other methods were modified to include the new field.
- However, developers forgot to modify the **restoreToNull()** method.
- Developers often forget or fail to identify all locations for a multi-entity edit.
- This missing change remained in the code for two years.

```
1  public class SQLChar extends
2      DataType implements
3      StringDataValue, StreamStorable{
4    ...
5  +  protected Clob _clobValue;
6    public int getLength() throws
7        StandardException{
8  +     if ( _clobValue != null ) {
9  +       return getClobLength(); }
10     if (rawLength != -1)
11       return rawLength;
12     if (stream != null) {
13     ...
14   }
15   public void restoreToNull() {
16     value = null;
17     stream = null;
18     rawLength = -1;
19     cKey = null;
20   }}
```

Motivating Example

- Could CMSuggester have solved this issue?
  - Identify added field
  - Identify changed methods
  - Identify peer fields
  - Suggest methods to change

```
1  public class SQLChar extends
2      DataType implements
3      StringDataValue, StreamStorable{
4    ...
5  + protected Clob _clobValue;
6    public int getLength() throws
7        StandardException{
8  +   if ( _clobValue != null ) {
9  +     return getClobLength(); }
10     if (rawLength != -1)
11       return rawLength;
12     if (stream != null) {
13     ...
14   }
15   public void restoreToNull() {
16     value = null;
17     stream = null;
18     rawLength = -1;
19     cKey = null;
20   }}
```

Empirical Findings

- In a prior study [9], 2,854 bug fixes from four popular open source projects to explore multi-entity edits.
- Study reveals that *CM→AF is one of the most popular patterns.
- Five such commits in each project were sampled to manually analyze the co-changed methods for any newly added field.

Table 1: Commonality inspection of 20 ***CM→AF** multi-entity edits

Project	Commits	Added Field	# of Changed Methods	Commonality
Aries	3d072a4	monitor	2	Field access
	50ca3da	properties	2	Field access
	5d334d7	BEAN	2	Method invocation
	95766a2	NS_AUTHZ	2	None
	9586d78	enlisted	3	Field access
Cassandra	0792766	validBufferBytes	3	Field access
	0963469	isStopped	2	Field access
	0d1d3bc	componentIndex	3	Field access
	1c9c47d	nextFlags	2	Field access
	266e94f	STREAMING_SUBDIR	2	Method invocation
Derby	f578f070	stateHoldability	2	Field access
	6eb5042	outputPrecision	2	Field access
	2f41733	MAX_OVERFLOW_ONLY_REC_SIZE	3	None
	099e28f	XML_NAME	3	Field access
	81b9853	activation	5	Field access
Mahout	0be2ea4	LOG	2	Field access
	0fe6a49	FLAG_SPARSE_ROW	2	Field access
	22d7d31	namedVector	2	Field access
	29af4d7	normalizer	2	Field access
	2f7f0dc	NUM_GROUPS_DEFAULT	2	None

Empirical Findings

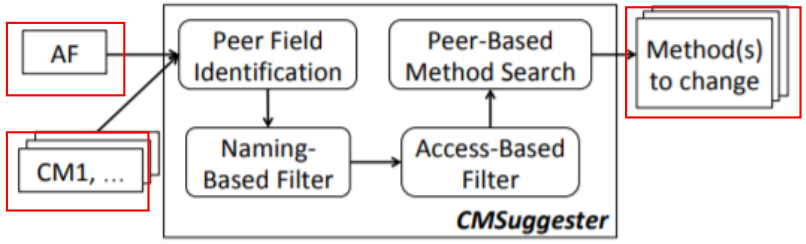
- In 15 of the 20 examined revisions, the co-changed methods commonly access existing field(s).
- For each added field, there are 2 to 5 methods co-changed to access the field
- Findings show that when one or more methods in a cluster are changed to access a new field, the other methods from the same cluster are likely to be co-changed for the new field access

Table 1: Commonality inspection of 20 ***CM→AF** multi-entity edits

Project	Commits	Added Field	# of Changed Methods	Commonality
Aries	3d072a4	monitor	2	Field access
	50ca3da	properties	2	Field access
	5d334d7	BEAN	2	Method invocation
	95766a2	NS_AUTHZ	2	None
	9586d78	enlisted	3	Field access
Cassandra	0792766	validBufferBytes	3	Field access
	0963469	isStopped	2	Field access
	0d1d3bc	componentIndex	3	Field access
	1c9c47d	nextFlags	2	Field access
	266e94f	STREAMING_SUBDIR	2	Method invocation
Derby	f578f070	stateHoldability	2	Field access
	6eb5042	outputPrecision	2	Field access
	2f41733	MAX_OVERFLOW_ONLY_REC_SIZE	3	None
	099e28f	XML_NAME	3	Field access
	81b9853	activation	5	Field access
Mahout	0be2ea4	LOG	2	Field access
	0fe6a49	FLAG_SPARSE_ROW	2	Field access
	22d7d31	namedVector	2	Field access
	29af4d7	normalizer	2	Field access
	2f7f0dc	NUM_GROUPS_DEFAULT	2	None

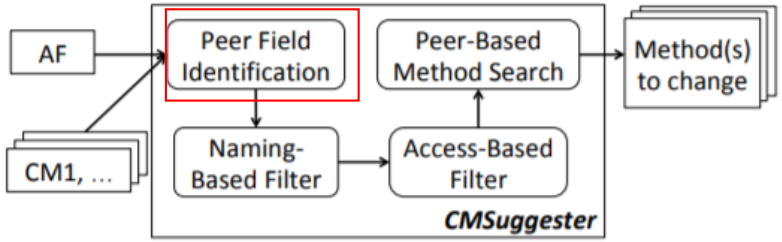
Approach

- Works in four steps:
  1. Peer Field identification
  2. Naming-Based Filter
  3. Access-Based Filter
  4. Peer-Based Method Search



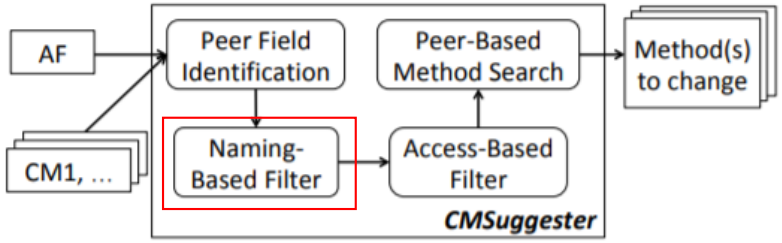
Approach: Peer Field Identification

- Newly added field: f_n
- **Peer fields** are used to denote the existing fields that satisfy the following conditions:
  - Exists in the same class as f_n
  - accessed by one or more changed methods that also access f_n
- A peer fields set **P = {p1, p2, ….}** is created.



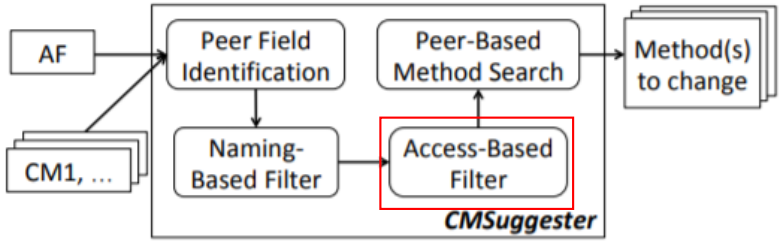
Approach: Name-Based Filter

- Considers two naming patterns:
  - Constant fields (Usually capitalize all letters)
    Example:MAX_OVERFLOW_SIZE
  - Variable fields (Usually have a combination of lowercase and uppercase letters)
    Example:outputPrecision
- Fields are classified to be either constants or variables.
- If f_n is a variable, constants are removed from **P (**the list of peer fields), and vice versa.



Approach: Access-Based Filter

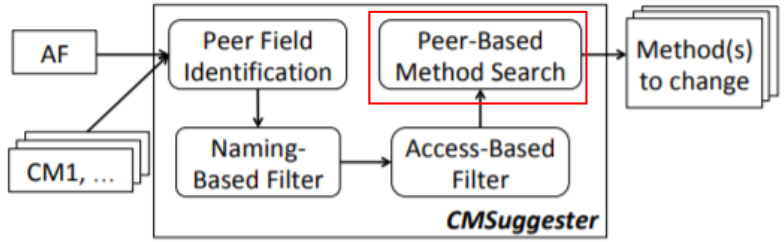
- For each methods, accessed fields are classified into the three following modes:
  - Pure Read
  - Pure Write
  - Read and Write



- When a field's access mode is distinct from that of f_n, CMSuggester removes the field from **P**.

Approach: Peer-Based Method Search

- For the refined list of peer fields in **P**:
  - Search for unchanged methods that access at least two of the refined fields.
  - For efficiency, leverage access modifiers to reduce search space.

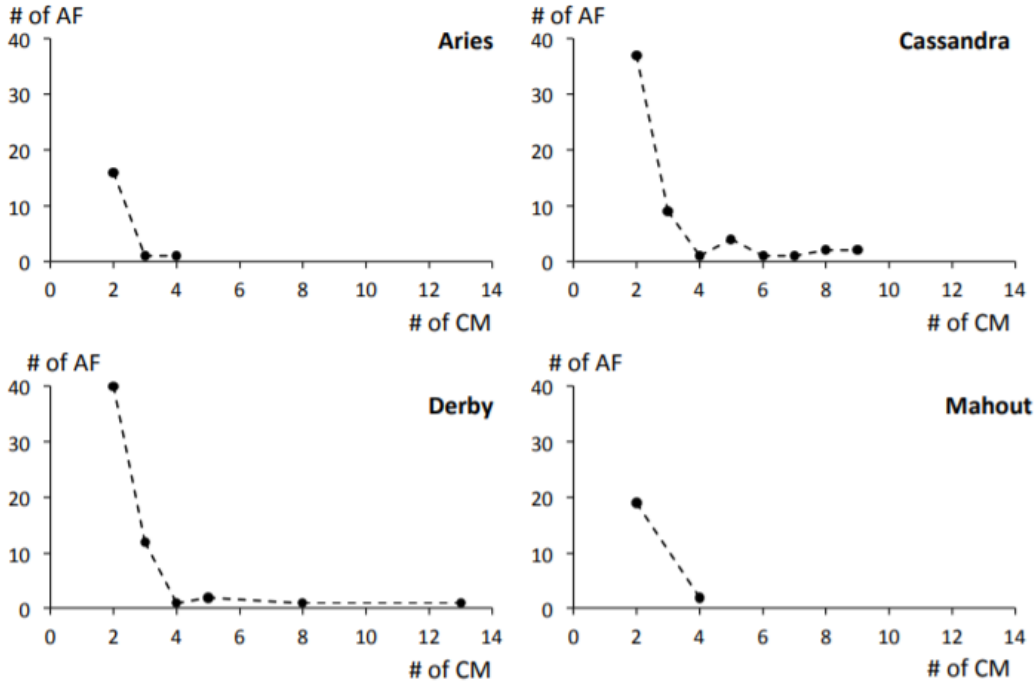


Evaluation: Data Set

- For four open source projects (discussed before), search for *CM → AF edits based on the following criteria:
  - Contains at least two methods co-changed for an added field
  - Has each changed method accessing at least two existing fields

- A total of 106 commits are retrieved for the four projects.

Evaluation: Data Set



Evaluation: Experimental Setup

- For the experiments, the tasks are divided into three categories:
  - one-AF-one-CM (1A1C)
  - one-AF-two-CM (1A2C)
  - one-AF-three-CM (1A3C)

	Aries	Cassandra	Derby	Mahout	Total #
# of program commits	10	45	42	9	106
# of 1A1C suggestion tasks	39	172	151	46	408
# of 1A2C suggestion tasks	9	237	168	12	426
# of 1A3C suggestion tasks	4	379	366	8	757

Evaluation: Metrics

- Coverage
- Precision
- Recall
- F-Score
- Weighted Average

Evaluation: Metrics

- Coverage (later evaluations are limited based on coverage)

$$C = \frac{\#\text{ of tasks with a tool's suggestion}}{\text{Total }\#\text{ of tasks}} * 100\%$$

Evaluation: Metrics

- Precision

$$P = \frac{\#\text{ of correct suggestions}}{\text{Total }\#\text{ of suggestions by a tool}} * 100\%$$

Evaluation: Metrics

- Recall

$$R = \frac{\#\text{ of correct suggestions by a tool}}{\text{Total }\#\text{ of expected suggestions}} * 100\%$$

Evaluation: Metrics

- F-score

$$F = \frac{2 * P * R}{P + R} * 100\%$$

Evaluation: Metrics

- Weighted Average

$$\Gamma_{overall} = \frac{\sum_{i=1}^{4} \Gamma_i * n_i}{\sum_{i=1}^{4} n_i}.$$

Evaluation: Comparison

- CMSuggester is compared against state-of-the-art co-change suggestion tool: ROSE [10].
- ROSE mines the association rules between co-changed entities from software version histories.
- Example rule mined by ROSE:

$$\{(\_Qdmodule.c, func, GrafObj\_getattr())\} \Rightarrow$$
$$\{(qdsupport.py, func, outputGetattrHook()).\}$$

Evaluation: Result (1A1C Tasks)

- CMSuggester outperforms ROSE for all the evaluation metrics.

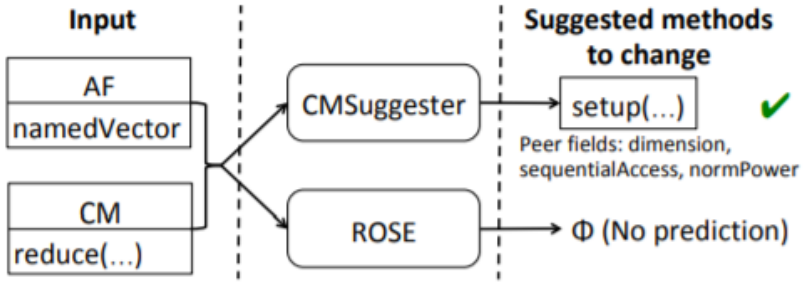
- Two major reasons:
  - Since ROSE depends on version history (which may be incomplete or some entities may have never been co-changed before)

  - ROSE does not leverage any syntactic or semantic relation between the co-changed entities and can infer incorrect rules.

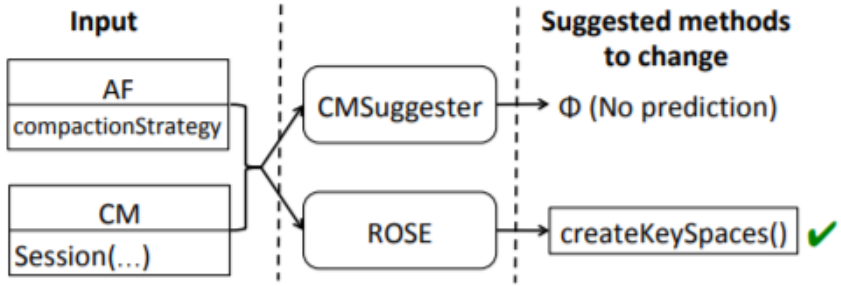
Project	CMSuggester				ROSE			
	C	P	R	F	C	P	R	F
Aries	51	68	85	76	31	35	39	37
Cassan-dra	69	81	75	78	38	53	71	61
Derby	71	71	68	69	22	25	42	31
Mahout	72	72	68	70	13	5	33	9
WA	**68**	**75**	**72**	**73**	**29**	**41**	**58**	**48**

Evaluation: Result (1A1C Tasks)

- CMSuggester outperformed ROSE in many 1A1C tasks.
- This shows that CMSuggester complements ROSE by inferring co-changes from methods' common field accesses instead of from the history.



CMSuggester outperforms ROSE



ROSE outperforms CMSuggester

Evaluation: Result (1A2C and 1A3C Tasks)

- Both CMSuggester and ROSE have obtained similar F-scores.
- However, CMSuggester has obtained a significantly higher coverage.
- Higher coverage shows that it predicted changes for the majority of tasks
- ROSE is limited by historical data.
- CMSuggester can find more peer fields from more methods provided.

Project	CMSuggester				ROSE			
	C	P	R	F	C	P	R	F
Aries	89	35	50	41	0	-	-	-
Cassandra	76	65	66	65	31	63	69	66
Derby	96	65	55	60	3	7	15	10
Mahout	100	35	39	37	0	-	-	-
WA	**85**	**63**	**60**	**61**	**8**	**59**	**66**	**62**

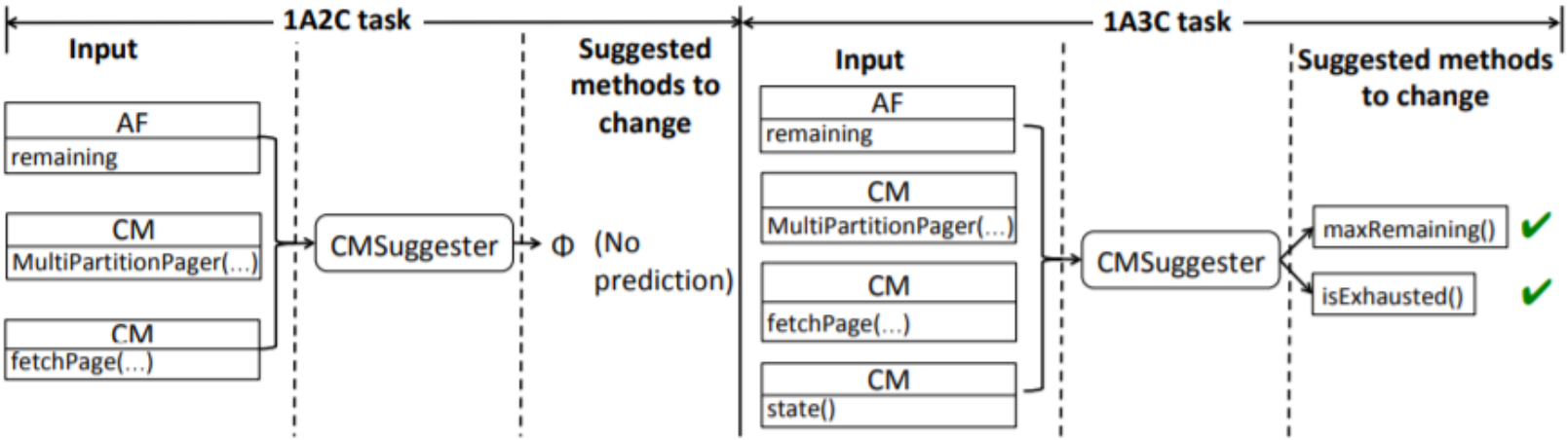
1A2C Tasks

Project	CMSuggester				ROSE			
	C	P	R	F	C	P	R	F
Aries	100	12	25	16	0	-	-	-
Cassandra	75	56	62	59	33	57	64	60
Derby	100	66	61	63	0	0	0	-
Mahout	100	21	25	23	0	-	-	-
WA	**88**	**61**	**61**	**61**	**17**	**57**	**63**	**60**

1A3C Tasks

Evaluation: Result (1A2C and 1A3C Tasks)

- CMSuggester predicts better when more CMs are provided as input.



Evaluation: Sensitivity to Filters

Project	CMSuggester				CMSuggester$_o$				CMSuggester$_n$				CMSuggester$_a$			
	C	P	R	F	C	P	R	F	C	P	R	F	C	P	R	F
Aries	51	68	85	76	77	70	83	76	72	70	86	77	56	67	86	75
Cassandra	69	81	75	78	88	78	76	77	80	81	74	77	75	79	76	77
Derby	71	71	68	69	97	63	60	61	94	66	63	64	73	67	64	65
Mahout	72	72	68	70	96	6	57	56	74	72	68	70	93	56	57	56
WA	**68**	**75**	**72**	**73**	**91**	**69**	**68**	**68**	**84**	**73**	**70**	**71**	**75**	**71**	**70**	**70**

- CMSuggester obtained the lowest overall coverage (68%), but the highest overall precision (75%), recall (72%), and F-score (73%).
- $CMSuggester_o$ achieved the highest coverage (91%) but lowest F-score (68%).
- Compared with $CMSuggester_a$, $CMSuggester_n$ obtained better coverage, precision, F-score.

Related Work

- Co-change Mining
- Change Recommendation Systems
- Automatic Program Repair (APR)

Related Work

- Co-change Mining:
  - Mine version histories for co-change patterns
  - These approaches do not analyze any syntactic or semantic relationship between co-changed modules
  - CMSuggester complements these approaches when version history is limited

Related Work

- Change Recommendation Systems:
  - Recommend changes based on either the co-occurrence of APIs or code similarity
  - In comparison, CMSuggester recommends changes based on the common field accesses between methods

Related Work

- Automatic Program Repair (APR):
  - There are tools proposed to generate candidate patches for certain bugs, and automatically check patch correctness using compilation and testing.

- Differences with CMSuggester:
  - APR focuses on single-entity changes by creating single-method updates from scratch.
  - CMSuggester focuses on multi-entity changes by suggesting method changes to complement already-applied edits
  - APR approaches generate concrete and applicable statement-level changes as a candidate fix
  - CMSuggester locates methods to change.

Discussion

- Possible additions and/or modifications of the filters for better performance.
- Extending the tool to handle other popular co-change patterns: ***CM → CM, *CM → AM, etc.**
- Thoughts about the evaluation process (possible weaknesses).
- Weaknesses of CMSuggester.

References

1. Christa, S., Madhusudhan, V., Suma, V., Rao, J.J.: Software maintenance: From the perspective of effort and cost requirement. In: Proc. ICDECT. pp. 759–768 (2017)
2. Zhong, H., Su, Z.: An empirical study on real bug fixes. In: Proc. ICSE. pp. 913–923 (2015)
3. Herzig, K., Just, S., Zeller, A.: It's not a bug, it's a feature: how misclassification impacts bug prediction. In: Proc. ICSE. pp. 392–401 (2013)
4. Park, J., Kim, M., Ray, B., Bae, D.H.: An empirical study of supplementary bug fixes. In: Proc. MSR. pp. 40–49 (2012)
5. Zimmermann, T., Weisgerber, P., Diehl, S., Zeller, A.: Mining version histories to guide software changes. In: Proc. ICSE. pp. 563–572 (2004)
6. Ying, A.T.T., Murphy, G.C., Ng, R.T., Chu-Carroll, M.: Predicting source code changes by mining change history. IEEE Trans. Software Eng. 30(9), 574–586 (2004)
7. Kim, M., Notkin, D.: Discovering and representing systematic code changes. In: Proc. ICSE. pp. 309–319 (2009)
8. . Meng, N., Kim, M., McKinley, K.: Lase: Locating and applying systematic edits. In: Proc. ICSE. pp. 502–511 (2013)

References

9. Wang, Y., Meng, N., Zhong, H.: An empirical study of multi-entity changes in real bug fixes. In: Proc. ICSME (2018)
10. Zimmermann, T., Weisgerber, P., Diehl, S., Zeller, A.: Mining version histories to guide software changes. In: Proc. ICSE. pp. 563–572 (2004)

Thank you