Advanced Topics in Software Engineering: Software Repository Mining

Giovanni Magoga
MatrNr: 21855119
Supervisor: M.Sc. Alexander Trautsch

January 29, 2019

Contents

1 Introduction

In the realm of software engineering, bug-fixing commits are often missing references to the related issues in the respective bug-tracking systems. Hence, valuable software quality metrics and production insights are lost, due to the very same negligence of software developers which is being assessed. The purpose of this report is illustrating the various techniques aimed at artificially constructing links between VCS and bug-trackers, with a particular regard to their varying motivations and conclusions from a chronological perspective (Section 2). Each consecutive section will compare the approaches in a standardized format, illustrating the main steps prosecuted by each author, the feature set analyzed, their final results and considerations. The report is concluded with a visual comparison of the increasing performances reported by each study (Section 9).

2 Timeline

2005 — The described problem was first acknowledged by Sliwerski et al. [6], in the scope of a research on when changes to a code base would require themselves to be fixed again in the future. The authors approach relied on basic metrics employing regular expressions to extract identifiers such as bug numbers, issue flags and author ids. Nevertheless, their research provided valuable insights on which days of the week were the least productive, as they introduced a sensible amount of fix-inducing changes, requiring additional work. With such a promising outcome, their contribution set the first milestone for a long series of studies to come.

2011 — With the creation of git and the birth of codebase hosting services like GitHub, open source communities shifted towards more heterogeneous environments, where good development practices are harder to enforce. With an estimated average of 54% of bugs with missing links, Wu et al. [9] are motivated to research a new scalable approach to link recovery, named ReLink. After comparing their results against the ones obtained through the previous set of heuristics, they confirm the importance of the relationship between the percentage of true links and the reliability of quality metrics.

2012 — ReLink fell short when trying to link bugs to commits whose descriptions weren't similar. To overcome the issue, Nguyen et al. [4] built MLink upon the previous studies, which also evaluates similarities across the source code, reporting substantially higher both precision and accuracy against the very same datasets from Wu et al.

2014 — Aknowledging the general direction and high industry value of the previous studies, Prechelt et al. [5] provide a highly formal framework to build upon, demonstrating the vast applicability and reproducibility of their method on both commercial (Infopark AG) and communal datasets (GitHub projects)

2015 Thanks to Changescribe [3], released in the same year, it becomes possible to enrich commit messages with additional contextual information. Le et al. [2] propose a more sophisticated approach to make the most out of the wealth of newly available commit metadata, employing a random forest algorithm as classifier. Their results, obtained against a set of Apache commons libraries, are substantially better than those obtained through MLink, whose binary file was made available for comparision.

2016 Sun et al. [7] take the research one step further by tuning the data collection phase. This research goal is achieved by including non-source documents in the starting dataset and by filtering out irrelevant source documents to minimize data noise. Coupled with a relatively less sophisticated classifier, they obtain higher scores than the previous state of the art, underlying the importance of starting from a clean and representative pool of candidates.

3 When do changes induce fixes?

3.1 General approach

The authors employ the Mozilla and Eclipse codebases as test subjects, and thus foresee some level of consistency from the developers. With this premise, links are retrieved by expecting valid bug-change references and appropriate tags to relevant bug-tracker entries. Upon link validation, transactions are unpacked and their changes interlinked with past commits, building a fix-inductive tree for each commit.

3.2 Features

For a candidate link to be deemed as valid, it would need to satisfy a condition accounting for semantic and syntactic relevance. These two metrics are obtained by validating the following characteristics, assigning a *syn* score of maximum 1 in case any syntactic condition is met, whereas *sem* would represent the number of valid semantic ones. Afterwards, all the links with $syn \neq 1$ would be discarded, unless their $sem > 0$.

- **Syntactic features**
  - The commit log contains a bug number
  - The log message contains a keyword or only plain/bug numbers
- **Semantic features**
  - The associated bug has been resolved as fixed at least once
  - The short description of the bug report is contained in the log message of the transaction.
  - The author of the transaction has been assigned to the bug.
  - One or more of the files affected by the transaction have been attached to the bug.

3.3 Results

The results display a somewhat even distribution around the central class ($syn = 1, sem = 2$).

syn / sem	0	1	2	3	4	total
0	270 (1%)	1,287 (5%)	2,057 (8%)	1,439 (6%)	2 (0%)	5,055 (20%)
1	324 (1%)	4,152 (16%)	9,265 (37%)	1,581 (6%)	5 (0%)	15,327 (61%)
2	110 (0%)	1,922 (8%)	2,421 (10%)	482 (2%)	0 (0%)	4,935 (19%)
total	704 (3%)	7,361 (29%)	13,743 (54%)	3,502 (14%)	7 (0%)	25,317 (100%)

Table 1: Distribution of links by (syn, sem) class in Eclipse

syn / sem	0	1	2	3	4	total
0	560 (1%)	2,899 (5%)	4,281 (8%)	639 (1%)	8 (0%)	8,387 (16%)
1	1,211 (2%)	9,059 (17%)	16,336 (30%)	2,241 (4%)	22 (0%)	28,669 (54%)
2	478 (1%)	5,250 (10%)	9,133 (17%)	1,645 (3%)	12 (0%)	16,518 (31%)
total	2,249 (4%)	17,208 (32%)	29,750 (55%)	4,525 (8%)	42 (0%)	53,574 (100%)

Table 2: Distribution of links by (syn, sem) class in Mozilla

3.4 Considerations

In the scope of bug-linking, the authors acknowledge the work from Cubranic and Murphy [8], as a pursuable path for further improvement. In their analysis, links are also researched in the opposite direction, from bug database to version control, hypothesizing bug reports to be more descriptive than commits. While the authors' main goal is not relevant to the purpose of this report, it should be noted that they identified days of the week with the highest likelihood for fixes to be buggy: Friday for Eclipse and Saturday for Mozilla, with proportions of 24% and 61.1% respectively. This could have been a good estimation for the relatively homogeneous environments of the pre-git era.

4 ReLink

4.1 General approach

In this report, ZXing, OpenIntents and Apache are analyzed. From their dataset analysis, the authors identify some characterizing features, whose threshold values are iteratively computed by trying to maximize F-scores. The candidate links' space necessary to compute the measure, is obtained by either removing existing links or manually linking reports through the LINKSTER tool from Bird et al. [1] The results of the classification are then combined with links extracted traditionally, and their union is then employed in assigning scores for maintainability and defect prediction. Additionally, the authors note that ReLink's feature set could be adopted as a discriminator for their own initial, traditionally acquired, dataset, reducing the overall number of false positives. In this case, higher precision is achieved at the cost of lower recall; an unfavorable trade-off when striving for scalability.

4.2 Features

- **Time intervals** between bug submission and commit are required to be positive and up to a maximum of one day. The authors later increased the accuracy of the metric by measuring intervals between related comments as well.

- **Bug owner and change committer** should refer to the same author identity. The detection of identities was deemed non-trivial as string identitifiers were found to be variable among the different systems adopted by each project.

- **Text similarity** between bug-reports and change-logs was obtained by computing cosine similarity between TF-IDF vectors of their descriptions. To reduce the dimensionality, and thus the noise, of the words' vectors space, stop words are removed while synonyms only increase the score of one common term.

4.3 Results

Project	**Period**	**Revisions**	**#Fixed Bugs**	**Approach**	**% Linked Bugs**	**Recovery Links**		
						Precision	**Recall**	**F-measur**
ZXing	11/2007-12/2010	1-1694	135	Traditional	42.2%	1.0 69/69	0.482 (69/143)	0.651
				ReLink	70.4%	0.90 (107/118)	0.748 (107/143)	0.820
OpenIntents	12/2007-12/2010	1-2890	101	Traditional	69.3%	1.0 (87/87)	0.674 (87/129)	0.805
				ReLink	73.3%	1.0 (95/95)	0.731 (95/129)	0.847
Apache	11/2004-4/2008	76294-899841	686	Traditional	77.1%	0.746 (791/1060)	0.764 (791/1035)	0.755
				ReLink	89.8%	0.747 (904/1210)	0.873 (904/1035)	0.805
Apache Simulation	11/2004-4/2008	76294-899841	686	Traditional	38.2%	0.741	0.375	0.498
				ReLink	53.0%	0.682	0.523	0.592
Eclipse MAT Simulation	4/2008-2/2011	10-1070	108	Traditional	49.1%	1	0.418	0.582
				ReLink	96.3%	0.858	0.623	0.718

Table 3: Performance comparison between regex-based and ReLink

To justify the need of pursuing further research towards bug-linking techniques, the authors also report the substantial improvement in the precision of code quality metrics when applied with alongside their retrieval method.

Project	Linking approaches	%Bug-fixing changes	%Buggy files	Mean Time to Fix(days
ZXing	Golden	8.1% (138/1694)	29.6% (118/399)	7.5
	Traditional	4.0% (67/1694)	14.8% (59/399)	10.2
	ReLink	6.3% (107/1694)	20.8% (83/399)	7.3
OpenIntents	Golden	4.2% 121/2890	4.9% (36/742)	25.1
	Traditional	2.9% (83/2890)	2.6% (19/742)	21.2
	ReLink	3.3% (94/2890)	4.0% (30/742)	25.1
Apache	Golden	2.3% (976/43167)	50.5% (98/194)	159.7
	Traditional	1.7% (753/43167)	47.9% (93/194)	178.9
	ReLink	2.0% (866/43167)	52.1% (101/194)	153.9

Subject	Version	# of files	# of metrics	Linking approaches	% of buggy
ZXing	1.6	399	41	Traditional	14.8%
				ReLink	20.8%
				Golden	29.6%
Open-Intents	Revision 1088~2073	56	41	Traditional	10.7%
				ReLink	28.6%
				Golden	39.3%
Apache	2.0	194	60	Traditional	57.2%
				ReLink	46.9%
				Golden	50.5%

Table 4: Software quality metrics after ReLink

4.4 Improvements

One major downside of trying to maximize F-scores comes from extracting the starting "golden set", which has to be done manually and is thus hardly scalable. Furthermore, should similarity between bug and commit descriptions not be characteristic, F-scores would drop significantly. Hence, the authors note how further work should be aimed at reducing this kind of variance and focus on commercial projects as well.

5 Mlink

5.1 General approach

The authors follow the same general protocol delineated by Wu et al, employing the same dataset and "golden set". Links are obtained by assembling their new filters (reported below) with the previously researched ones into a sequential pipeline, ordered by descending specificity. The results are reported in the case where recall is maximized, enabling a manual inspector to detect and correct false positives with ease.

5.2 Features

Since the major contribution of MLink can be attributed to the inspection of source files, the actual feature set should be classified as a variation of the text similarity metric. Nevertheless, the list of features as reported by the authors is illustrated below for sake of completeness:

- **Code features** are the main focus of this study. Terms contained in the source code are associated with their descriptive counterparts, reconstructing links even where there's low similarity between their descriptions.

- **Different terms in both sides** of the bug-commit relationships are unified to compute similarity with stemming and lemmatization techniques to reduce the dimensionality of the vector space.

- **Notes in bug comments** could contain terms linkable by similarity as well, especially in controversial issues responsible for high degrees of interaction.

5.3 Results

For the purpose of comparison, the authors rely on the publicly available ReLink source code and measure an improvement of 10% in F-score. On the footsteps of Wu et al., software metrics are also computed, although with a comparatively less sensitive impact.

	ZX			OI			AP		
	Golden	Re-Link	M-Link	Golden	Re-Link	M-Link	Golden	Re-Link	M-Link
Changes	1,694	1,694	1,694	2,890	2,890	2,890	43,167	43,167	43,167
Fixes	138	107	120	121	94	103	976	866	921
%Fixes	8.1%	6.3%	7.1%	4.2%	3.3%	3.6%	2.3%	2.0%	2.1%
Files	399	399	399	742	742	742	194	194	194
Buggy	118	83	102	36	30	30	98	101	100
%Buggy	29.6%	20.8%	25.6%	4.9%	4.0%	4.0%	50.5%	52.1%	51.5%

Table 5: Software quality metrics after MLink

Sys.		T	base	base+ patch	base+ patch+ name	base+ patch+ name+ text	base+ patch+ name+ text+ assoc (MLink)	Re-Link	base+ Bug-Scout
ZX	Det	69	73	78	93	130	131	118	130
	Corr	69	73	78	92	125	126	107	102
	Incor	0	0	0	1	5	5	11	28
	Miss	74	70	65	51	18	17	36	41
	Prec	100	100	100	99	96	96	91	79
	Rec	48	51	55	64	87	88	75	71
	Fs	65	68	71	78	92	93	82	75
OI	Det	87	100	100	103	105	113	95	112
	Corr	87	100	100	103	105	110	95	105
	Incor	0	0	0	0	0	3	0	7
	Miss	42	29	29	26	24	19	34	24
	Prec	100	100	100	100	100	97	100	94
	Rec	67	78	78	80	81	85	74	81
	Fs	81	87	87	89	90	91	85	87
AP	Det	1,060	1,096	1,112	1,129	1,184	1,193	1,261	1,142
	Corr	791	909	920	932	973	978	934	921
	Incor	269	187	192	197	211	215	327	221
	Miss	299	181	170	158	117	112	156	169
	Prec	75	83	83	83	82	82	74	81
	Rec	73	83	84	86	89	90	86	84
	Fs	74	83	84	85	86	87	79	83

Table 6: MLink performance vs ReLink

5.4 Considerations

The optimistically high recall value is disputed by the following authors for the same reasons underlined in the discussion of ReLink.

6 Bflinks

6.1 General approach

For the purpose of establishing thresholds for those filters, the authors note the infeasibility of manually generating a "golden set" from which to compute absolute recall scores, proposing, in turn, an evaluation model which leverages relative measures of recall. Beside this criticism, the authors recognize the importance of the previous works, adopting a similar filtering pipeline model.

6.2 Features

Most of the features employed in the study have already been investigated by previous authors, yet Prechelt et al. formalize their roles to achieve a high degree of modularity, with a distinct separation between "generators" used for candidate selection and actual filters. Most filtering thresholds are calculated empirically, while some of them are established on the base of epistemic engineering contributions where possible.

- **Generators**
  - IDs of bug-tracker entries mentioned in commits

  - IDs of commits mentioned in bug-tracker entries

- **Filters**
  - Commit-bug ID references shouldn't be (too) many-to-one
  - Bug-commit ID references shouldn't be (too) many-to-one
  - Bugzilla IDs must have an acceptable format
  - Delay between bug and commit should be positive
  - Delay between bug and commit should be relatively small
  - Bug-fix link references should be bidirectional

6.3 Results

The measurements reported in table 7 are calculated against a theoretical "golden set" with 100% coverage by only manually classifying links after filtering. Still, with a relative loss of 7.1% in the best case scenario, the final recall of at least 65% can be estimated with high confidence, justifying the soundness of the model over an hypothetical random sampling method with 50% recall.

	This one filter solo				Filters up to here together			
	fp	*st*	*rl*	*op*	*fp*	*st*	*rl*	*op*
TT(C)	43	0.3	0.2	73	43	0.3	0.2	73
TT(B)	99	0.9	0	74	86	1.2	0.2	74
FC	96	3.7	0.2	77	93	4.9	0.4	78
LU	62	5.6	2.1	77	74	8.3	2.7	79
SB	81	6.9	1.3	81	67	15	4.0	90
FB	50	11	5.6	77	60	23	9.3	93
UD[a]	29	59	42	99	63	19	7.1	93

Table 7: Filtering precision, strength, results loss and overall precision for each filter

6.4 Considerations

The authors stress the importance of picking the right filter for each dataset one wishes to analyze, with examples regarding the enforcement of bidirectional references in particular. Said heuristic is found to be decisive when analyzing commit data generated in its regard, notably in the case of GitHub, where such practice is enforced by the system. Since many young OSS projects rely on said service, the bidirectional filter asserts an even greater importance since broader alternatives could yield suboptimal results. Given the success of such an aggressive filter in the right environments, the authors suggest any future effort to focus on similarly discriminating features.

7 RClinker

7.1 General approach

The authors overcome the issue of generating a "golden set" for their F-score calculation, by breaking existing links and performing 10-fold cross-validation on the starting set. Doing so provides a disproportionate majority of false links, which, if fed to their random forest classifier, would deal wrong performance measurements by referring to the over represented class. Thus, the candidate links' pool dataset is undersampled by extracting a N number of neighbors textually close to the true links.

7.2 Features

Thanks to Changescribe, the authors can compile a set of 20 meta-features taking into account text similarity, time, frequency and ownership values for each commit. After assigning Fischer scores to each feature, the authors rank each them by their significance. It can be observed that text similarity metrics are dominant once again (Ti).

	CLI	Collections	CSV	IO		Math
1	T_3	T_3	T_3	T_3	T_3	T_3
2	M_5	T_2	T_2	T_1	T_1	T_2
3	T_1	T_1	T_1	T_2	T_2	T_1
4	T_2	T_5	T_5	T_7	M_5	T_5
5	T_5	T_8	T_7	T_8	T_7	T_7
6	T_7	T_7	T_8	T_6	T_5	T_4
7	T_4	T_4	M_5	M_1	T_9	T_8
8	T_8	M_5	M_8	T_5	T_4	M_5
9	T_9	M_{10}	T_4	T_4	T_8	T_6
10	M_9	T_9	T_6	T_9	M_1	M_1

Table 8: Top-10 relevant features by project

7.3 Results

The algorithm outperforms MLink by 138.66% as reported by the authors. The tests are run on 7 libraries from the Apache commons collection.

7.4 Considerations

The number of nearest neighbors selected for the undersampling step is found to affect both the performance and the accuracy of the method, with low values yielding exponentially lower precision scores as they end up unbalancing the data in the opposite direction. Being this the only substantially free hyperparameter, the model is deemed to be highly applicable to different environments as the random forest itself is responsible for balancing the provided features and any linear combination of them.

	Precision	Recall	F-measur
1	19.70%	92.34%	31.76%
5	50.91%	89.27%	64.32%
10	61.95%	87.04%	72.12%
15	66.23%	85.96%	74.62%
20	72.87%	85.30%	78.41%
25	73.73%	84.50%	78.68%

Table 9: F-scores relative to N

8 Frlink

8.1 General approach

Even in this case the "golden set" issue is solved by means of 10-fold cross-validation on a smaller set. Differently from RClinker, undersampling is achieved by selecting candidates committed between a timespan of 7 days from any of the temporal bug-entry attributes (i.e. comment timestamps), minimizing biases attributed to text similarity. Filtering parameters are computed with a threshold learning algorithm similar to those previously employed.

8.2 Features

Code and text features similar to MLink. Additionally, file relevance is taken into account by discarding documents with drastically different vector spaces from any candidate bug counterpart. Additionally, the authors specify that text similarity vectors are computed separately for each textual section of commits and bugs. The motivation for this choice being the varying degree of relevance between title, subtitle and description for each entry; valuable information which is evened out when computing a single vector.

8.3 Results

Comparing F-measures with the highest recall scores, RCLinker is outperformed by 40.75% on the same Apache projects. The authors note that by removing additional non-source documents, FRLink has a higher decay rate than RCLinker, suggesting the latter as a potentially superior classifier.

Approach	InputParameter	Precision	Recall	*F*
FRLink	*ITR*=0.98	48.73%	95.03%	63.72%
	ITR=0.96	59.75%	93.39%	72.51%
	ITR=0.94	65.34%	91.38%	75.71%
	ITR=0.92	72.65%	89.02%	79.86%
	ITR=0.90	76.75%	87.10%	81.44%
	ITR=0.88	80.02%	85.75%	82.66%
RCLinker	*N*=1	19.70%	92.34%	31.76%
	N=5	50.91%	89.27%	64.32%
	N=10	61.95%	87.04%	72.12%
	N=15	66.23%	85.96%	74.62%
	N=20	72.87%	85.30%	78.41%
	N=25	73.73%	84.50%	78.68%

Table 10: FRlink performance vs RCLinker

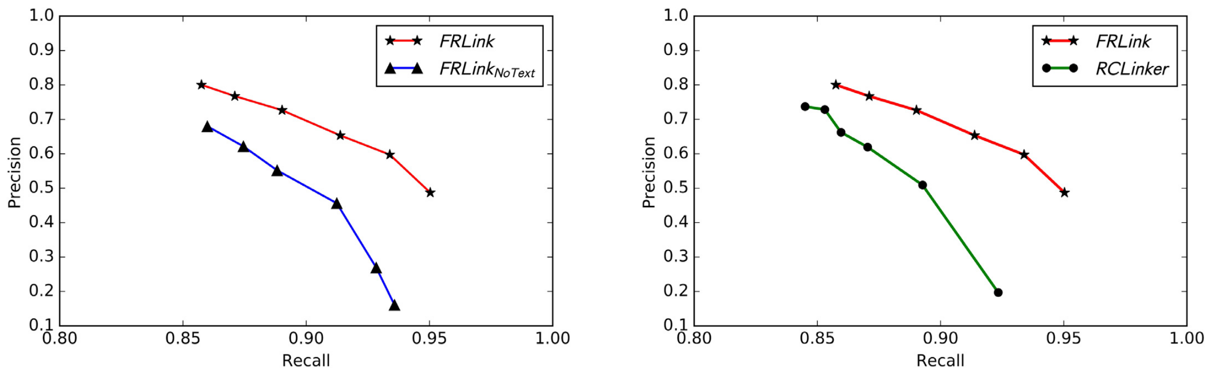

Figure 1: Precision decay similarities when neglecting non-source documents

8.4 Considerations

Subsequent research could focus its efforts towards a more sophisticated relevance selection algorithm, especially in the case of projects with a low percentage of non-source documents. Additionally, source code properties specific to the languages underlying each project could be employed to narrow the candidates space.

9 Comparison

Although it is to be noted that it's not possible to draw an accurate comparison between each approach, they've been reported down here to visually justify their chronological order. The first two studies have been left out since they lack any information comparable to the future ones and suffer from weak means of scoring as already shown by MLink, the least recent approach.

	Dataset	Sampling	Reported by
MLink	Apache	TFIDF distance k-means	RCLink
Bflinks	InfoPark	After first run	Self
RCLink	Apache	7 day distance	FRlink
FRlink	Apache	7 day distance	Self

Table 11: Scoring inconsistencies between approaches

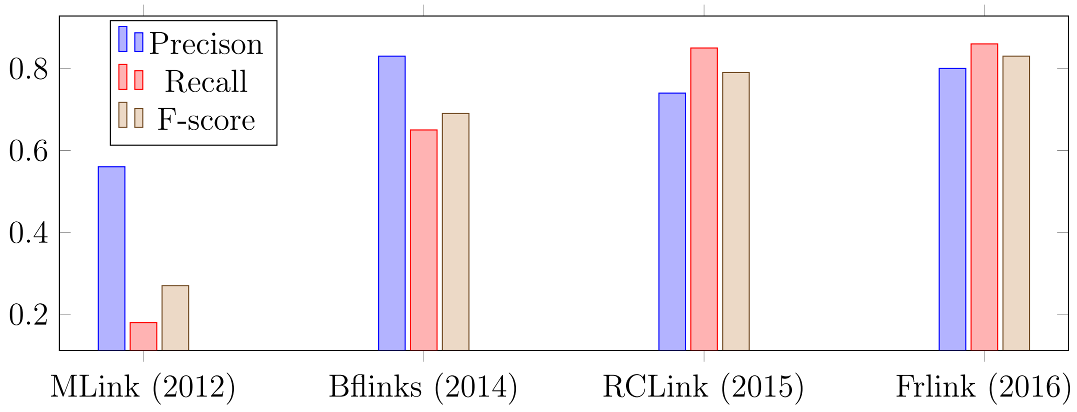

Figure 2: Chronological performance comparison

List of Figures

List of Tables

References

[1] Christian Bird, Adrian Bachmann, Foyzur Rahman, and Abraham Bernstein. Linkster: Enabling efficient manual inspection and annotation of mined data. In *Proceedings of the Eighteenth ACM SIGSOFT International Symposium on Foundations of Software Engineering*, FSE '10, pages 369–370, New York, NY, USA, 2010. ACM.

[2] Tien-Duy B. Le, Mario Linares-Vásquez, David Lo, and Denys Poshyvanyk. Rclinker: Automated linking of issue reports and commits leveraging rich contextual information. In *Proceedings of the 2015 IEEE 23rd International Conference on Program Comprehension*, ICPC '15, pages 36–47, Piscataway, NJ, USA, 2015. IEEE Press.

[3] M. Linares-Vásquez, L. F. Cortés-Coy, J. Aponte, and D. Poshyvanyk. Changescribe: A tool for automatically generating commit messages. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, volume 2, pages 709–712, May 2015.

[4] Anh Tuan Nguyen, Tung Thanh Nguyen, Hoan Anh Nguyen, and Tien N. Nguyen. Multi-layered approach for recovering links between bug reports and fixes. In *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*, FSE '12, pages 63:1–63:11, New York, NY, USA, 2012. ACM.

[5] Lutz Prechelt and Alexander Pepper. Bflinks: Reliable bugfix links via bidirectional references and tuned heuristics. In *International scholarly research notices*, 2014.

[6] Jacek Śliwerski, Thomas Zimmermann, and Andreas Zeller. When do changes induce fixes? In *Proceedings of the 2005 International Workshop on Mining Software Repositories*, MSR '05, pages 1–5, New York, NY, USA, 2005. ACM.

[7] Yan Sun, Qing Wang, and Ye Yang. Frlink: Improving the recovery of missing issue-commit links by revisiting file relevance. *Information and Software Technology*, 84, 12 2016.

[8] Davor Čubranić and Gail C. Murphy. Hipikat: Recommending pertinent software development artifacts. In *Proceedings of the 25th International Conference on Software Engineering*, ICSE '03, pages 408–418, Washington, DC, USA, 2003. IEEE Computer Society.

[9] Rongxin Wu, Hongyu Zhang, Sunghun Kim, and Shing-Chi Cheung. Relink: Recovering links between bugs and changes. In *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering*, ESEC/FSE '11, pages 15–25, New York, NY, USA, 2011. ACM.