레터논문 (Letter Paper)
방송공학회논문지 제27권 제2호, 2022년 3월 (JBE Vol.27, No.2, March 2022)
https://doi.org/10.5909/JBE.2022.27.2.240
ISSN 2287-9137 (Online) ISSN 1226-7953 (Print)

정확한 프로그램 결함 위치 추적을 위한 전-후처리 방법론

김 동 선[a)‡]

Pre/post-processing Operator Selection for Accurate Program Bug Localization

Dongsun Kim[a)‡]

요 약

프로그램 결함에 대한 위치 추적은 소프트웨어 유지 및 보수를 위해 필수적인 요소이다. 대부분은 버그리포트가 제출되었을 때, 결함 추적이 개발자들의 수작업으로 이루어지기 때문에 비용 소모가 많은 작업에 속한다. 현재까지 많은 연구자가 해당 작업을 자동화시키기 위하여 노력해 왔지만 보고된 결과에 따르면, 현업에서 사용되기에 아직도 부족한 성능을 보이는 추세이다. 이에, 본 연구에서는 많은 양의 버그 리포트 데이터와 관련 최신 연구들을 분석하여, 기존 연구들이 하나의 전처리 방법을 모든 버그 리포트에 일괄적으로 적용하고, 이런 방법은 위치 추적에 악영향을 준다는 것을 파악하였다. 본 논문에서는 이와 같은 문제점을 해결하기 위해 선택적인 전-후처리 방법론을 제안한다.

Abstract

Tracking the location of program defects is an essential task for software maintenance and repair. When a bug report is submitted, bug localization is a costly task because of the developer's manual effort. Many researchers have tried to automate the task, but according to the reported results, the performance is still insufficient in practice. Therefore, in this study, we analyzed a large amount of bug report data and the latest research and found that the existing studies used only one preprocessing without considering the characteristics of the bug report. In this paper, to solve the problems mentioned earlier, we propose a pre/post-processing operator selection approach for bug localization.

Keyword : Bug report, bug localization, processing operators

a) 경북대학교 컴퓨터학부(School of Computer Science and Engineering College of IT Engineering, Kyungpook National University)
‡ Corresponding Author : 김동선(Dongsun Kim)
E-mail: darkrsw@gmail.com
Tel: +82-53-950-7282
ORCID:https://orcid.org/0000-0003-0272-6860
※ This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. 2021R1I1A 3048013).
· Manuscript received January 17, 2022; Revised February 16, 2022; Accepted February 16, 2022.

Ⅰ 서 론

소프트웨어 개발에서 버그의 존재와 이것을 제거하는 활동은 매우 많은 비용을 초래한다. 소프트웨어의 배포 전 단계에서 수많은 테스팅 기법과 코드 리뷰 기법을 이용해 다양한 버그들을 식별할 수 있지만 소프트웨어 배포 후에도 여전히 많은 버그가 발견된다. 예를 들어, Apache Hive[1]의 경우, 버그가 12년 동안 22,000개나 발견되었다고 보고되었다. 버그 리포트들은 내부, 외부 개발자 혹은 소프트웨어의 사용자들에 의하여 작성되는데, 이 내용은 대부분 의도하지 않은 소프트웨어의 작동, 결과, 해당 시점의 로그 등에 초점이 맞추어져 있다. 버그 리포트가 제출되면 소프트웨어 내부에서 해당 버그가 존재하는 코드를 파악하는 노력이 필요한데, 소프트웨어의 크기에 따라 이 작업은 큰 부담이 되기도 한다. 따라서, 버그의 위치를 자동으로 파악하는 연구에 큰 노력이 필요하다[2].

기존 연구 중 많은 수가 정보 검색(Information Retrieval)[3] 기법을 기반으로 결함 위치 식별을 수행하였고, 추가로 소프트웨어의 변경 이력과 같은 정보를 사용하거나[4], 기계학습 알고리즘들을 사용하여 결함 위치 검출 성능(버그가 존재하는 것으로 추정되는 파일을 예측)을 높이고자 하였다. 이와 같은 노력에도 불구하고, 새로운 데이터에 기반한 평가를 다룬 최근 논문[5] 에서는 기존 연구방법들이 현업에서 사용하기에 부적합한 성능을 보였다고 보고하고 있다. 최근 연구들[6,7]에서 제시한 증거들을 바탕으로 일관된 전-후처리 방법을 모든 버그 리포트에 적용하는 것이 효율적이지 않다는 것을 파악하였다. 본 연구에서는 이와 같은 문제를 해결하기 위해 버그 리포트들에 각기 다른 전-후처리를 실시하는 방법을 제안한다.

Ⅱ. 제안기법

본 연구는 특정 한 종류의 전-후처리 기법이 버그 리포트에 맞지 않을 것이라 가정한다. 대부분의 기존 연구들은 고정된 전-후처리 기법을 일괄적으로 적용한다. 본 연구에서는 각기 다른 프로젝트의 버그 리포트와 소스코드는 서로 다른 특징들을 가지고 있을 수 있고, 따라서 이를 분석하여 해당 프로젝트들에 맞는 전-후처리 기법들이 적용되면 성능 향상에 기여할 수 있을 것으로 추측한다.

1. 결함 위치 추적을 위한 전-후처리 기법

정보 검색을 기반으로 하는 결함 위치 추적 기법들은 버그 리포트와 소스 코드를 입력으로 가진다. 이 입력들은 불필요한 정보의 제거, 정형화와 같은 전처리를 통해 성능 향상에 영향을 줄 수 있는데, 따라서 결함 위치 추적 연구들에서도 관련 있는 전처리를 해주어야 한다. 이와 비슷하게, 결과로 도출 된 파일들 중 불필요한 요소(e.g., 테스트 파일)를 제거하고, 특정한 정보를 바탕으로 하는 결과의 재 정렬과 같은 후처리도 사용 될 수 있다. 본 논문에서는 표 1과 같이 기존 논문들에서 사용 된 전-후처리 기법들에 집중한다. 기법으로 나열된 Basic은 모든 기존 연구에서 사용한 토큰화를 위한 방법이고, SWR(Stop Word Removal)과 STM(Stemming)은 각각 불용어 제거와 형태소 분석을 위해 쓰인 기법들이다. SPC(Dropping out Special Characters)와 CMC(Camel Case Split)는 소스코드의 텍스트를 분석할 때 전형적으로 사용된다. 마지막으로 후처리 기법에 쓰인 CE(Code Entity)는 소스 코드 파일 내의 토큰에 대해 우선순위를 높여주는 방법으로 사용되었다.

표 1. 프로그램 결함 위치 추적을 위한 전-후처리 기법
Table 1. Pre/post-processing operators for program bug localization

Pre-processing Operators	
Name	Description
Basic	Tokenization by space, tab, and line feed
SWR	Stopword removal
STM	Stemming
SPC	Special character removal
CMC	Camel Case tokenization
Post-processing Operator	
CE	Extraction of Source code tokens from bug reports

Ⅲ. 실 험

각기 다른 전-후처리 기법들이 각각의 프로젝트에 어떤

영향을 미치는지 파악하기 위해, 각 전-후처리의 조합을 각기 다른 프로젝트로 부터 추출된 버그 리포트와 그에 해당하는 소스 코드 쌍에 적용함으로써 성능을 비교해볼 수 있다. 또한, 본 논문에서 제안하는 방법을 일반적인 정보 검색 알고리즘인 VSM(Vector Space Model) 에 적용한 후 기존 연구들과 비교하여 제안하는 방법의 효과를 증명한다.

1. 데이터

검증을 위한 데이터 세트으로 표 2와 같이 이미 많이 알려지고, 저명한 기관들로부터 유지되고 있는 Apache MATH, Spring LDAP, Spring SHDP, 그리고 Spring SWS 프로젝트 데이터를 사용했다. 데이터는 시점에 따라 개발자나 사용자에 의해 추가, 삭제 될 수 있으므로 버그 리포트가 제출된 직후부터의 추가적인 정보와 첨부 파일들은 사용하지 않았다.

표 2. 실험에 사용된 프로그램
Table 2. Programs used in the experiment

Group	Project	# of source code files	# of versions	# of bug reports
Apache	MATH	1,617	15	245
Spring	LDAP	566	5	53
	SHDP	1,102	9	45
	SWS	925	25	174

2. 실험 결과

실험 결과는 전체적으로 각기 다른 조합의 전-후처리 기법들의 결과가 상이하게 나타날 수 있다는 것을 입증한다. 먼저, 본 연구에서는 이미 존재하는 연구인 BugLocator[6] 의 전처리 기법에 모든 조합을 사용해 보았고, 38.2% 의 MAP(Mean Average Precision) 가 39.9%로 상승, 50.6%의 MRR(Mean Reciprocal Rank)이 52.5% 로 상승하는 것을 발견 했다. 이는 전처리 기법의 변경만으로도 제안하는 방법이 결함 위치 식별 성능을 향상할 수 있다는 것을 보여준다.

추가적인 조합 실험에서 32(=2^5) 개의 조합을 모두 적용해 보았고, 그림 1에서 처럼 MATH 프로젝트의 경우, MRR 이 최하 38%(조합: SWR) 에서 최고 59%(조합: SPC, SWR, STM, CE)까지 상승하는 것을 볼 수 있었다. 가장 극단적인 사례는 SHDP 프로젝트로 33.3%p(32.9% ->

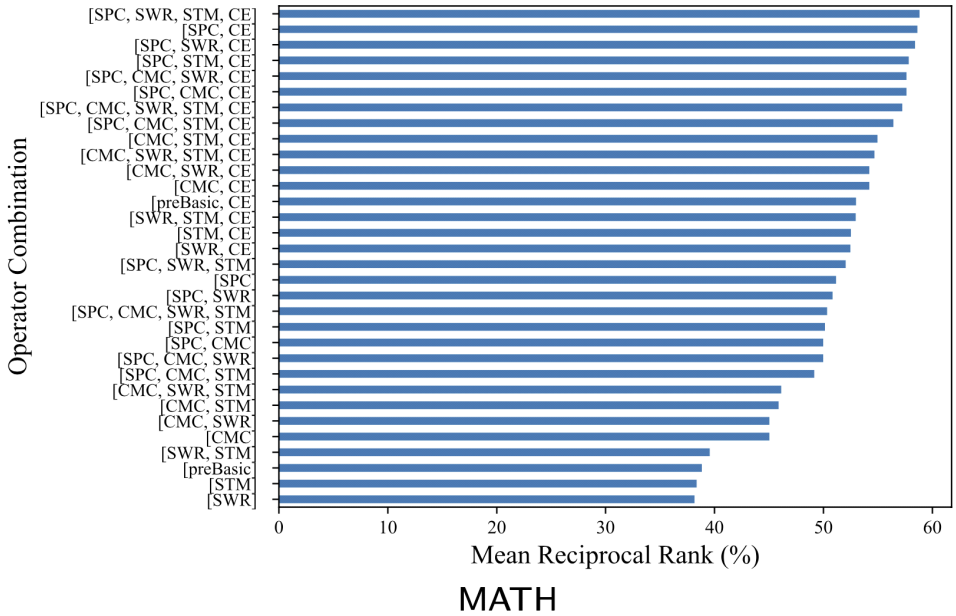

그림 1. Apache Math 프로그램에 제안된 기법을 적용한 결과
Fig. 1. Results of applying the proposed approach to Apache Math

표 3. 제안된 방법과 기존 연구 방법들의 결함 위치 추적 결과 비교
Table 3. Comparison of bug localization results between the proposed method and existing techniques

Project	BugLocator		BRTracer		BLUiR		BLIA		Locus		Proposed-Method	
	MAP	MRR	MAP	MRR	MAP	MRR	MAP	MRR	MAP	MRR	MAP	MRR
MATH	0.1563	0.2173	0.1586	0.2274	0.1952	0.2413	0.1765	0.2394	0.1895	0.2251	0.4376	0.5882
SHDP	0.4433	0.6279	0.4652	0.6734	0.3899	0.5184	0.4654	0.6222	0.4633	0.5826	0.4835	0.6617
LDAP	0.4401	0.6344	0.4875	0.7197	0.4681	0.6251	0.4824	0.6665	0.3857	0.5058	0.5328	0.7286
SWS	0.4002	0.5400	0.4211	0.5872	0.3811	0.4886	0.3969	0.5456	0.4177	0.5680	0.4317	0.5898
Average	0.3600	0.5049	0.3831	0.5519	0.3586	0.4684	0.3803	0.5184	0.3641	0.4704	0.4692	0.6421

66.2%)의 향상이 있었다. 본 연구에서는 각 프로젝트(i.e., MATH, SHDP, LDAP, SWS) 별로 최고의 조합은 [SPC, SWR, STM, CE], [SPC, CMC, CE], [SPC, SWR, STM, CE], [SPC, CE] 인 것을 찾아냈다. 조합 [SPC, SWR, CE]가 프로젝트 대부분에서 앞섰지만, SHDP에서는 9.5%p나 낮은 성능을 보였다. 이는 본 연구에서 가정한 "특정 한 종류의 전-후처리 기법이 모든 프로젝트와 데이터 세트에 맞지 않을 것"을 증명한다.

표 3은 같은 데이터를 사용한 다른 연구들과의 비교 결과를 보여준다. SHDP 프로젝트의 MRR 값을 제외한 모든 경우에서 본 연구가 제안하는 방법이 우세한 것을 알 수 있고, 평균적으로 11.0%p MAP 와 17.4%p MRR 의 성능향상을 보였다.

Ⅳ. 결 론

결함 위치 추적은 소프트웨어가 커질수록 비용 소모가 크고, 어려운 작업이다. 효과적인 자동 결함 위치 추적 방법론이 중요하지만, 아직 현업에서 사용하기에는 부족한 상황이다. 본 논문에서는 결함 위치 추적에서 사용되는 전-후처리 방법론들의 조합이 프로젝트마다 어떤 성능을 보여주는지 연구하였으며, 같은 데이터로 서로 다른 연구들을 비교하였다. 실험 결과를 통해서 각 프로젝트마다 최적의 성능을 보여주는 전-후처리 방법조합이 다르다는 것을 파악할 수 있었다. 또한, 최적의 조합을 사용하였을 때, 기존의 결함 위치 추적 방법들에 비해서, 본 연구에서 제안한 방법이 더 좋은 성능을 보여준다. 실험 결과를 통해 모든 버그 리포트에 일관된 전-후처리 방법을 적용하는 것이 아니라, 버그 리포트마다 특성에 맞게 적절한 전-후처리 방법을 적용할 필요가 있음을 보여준다. 향후 연구로는, 더 많은 전-후처리 방법들을 조합하는 방법, 다른 정보검색 기반의 결함 위치 추적 방법론들에 본 연구의 아이디어를 적용하는 것이 포함된다.

참 고 문 헌 (References)

[1] Hive-issues, https://issues.apache.org/jira/projects/HIVE/issues/ (Last Accessed: Jan. 2022).

[2] X. Huo, F. Thung, M. Li, D. Lo, S.-T. Shi, "Deep transfer bug localization", IEEE Transactions on Software Engineering, Vol. 47, No. 7, July 2019. doi: https://doi.org/10.1109/TSE.2019.2920771

[3] G. Gay, S. Haiduc, A. Marcus, T. Menzies, "On the use of relevance feedback in ir-based concept location", Proceedings of the IEEE International Conference on Software Maintenance (ICSM 2009), pp. 351 – 360, 2009. doi: https://doi.org/10.1109/ICSM.2009.5306315

[4] K. C. Youm, J. Ahn, E. Lee, "Improved bug localization based on code change histories and bug reports", Information and Software Technology, Vol. 82, pp. 177 – 192, 2017. doi: https://doi.org/10.1016/j. infsof.2016.11.002

[5] J. Lee, D. Kim, T. F. Bissyandé, W. Jung, Y. LeTraon, "Bench4BL: Reproducibility study on the performance of IR-based bug localization", Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis, pp. 61 – 72, 2018. doi: https://doi.org/10.1145/3213846.3213856

[6] J. Zhou, H. Zhang, and D. Lo, "Where should the bugs be fixed? - more accurate information retrieval-based bug localization based on bug reports," Proceedings of the 2012 International Conference on Software Engineering, Piscataway, NJ, USA, 2012, pp. 14 – 24. doi: https://doi.org/10.1109/ICSE.2012.6227210