KIPS Tr. Software and Data Eng.
Vol.3, No.12 pp.511~522 pISSN: 2287-5905

http://dx.doi.org/10.3745/KTSDE.2014.3.12.511

A Technique to Recommend Appropriate Developers for Reported Bugs Based on Term Similarity and Bug Resolution History

Park Seong Hun† · Kim Jung Il†† · Lee Eun Joo†††

ABSTRACT

During the development of the software, a variety of bugs are reported. Several bug tracking systems, such as, Bugzilla, MantisBT, Trac, JIRA, are used to deal with reported bug information in many open source development projects. Bug reports in bug tracking system would be triaged to manage bugs and determine developer who is responsible for resolving the bug report. As the size of the software is increasingly growing and bug reports tend to be duplicated, bug triage becomes more and more complex and difficult. In this paper, we present an approach to assign bug reports to appropriate developers, which is a main part of bug triage task. At first, words which have been included the resolved bug reports are classified according to each developer. Second, words in newly bug reports are selected. After first and second steps, vectors whose items are the selected words are generated. At the third step, TF-IDF(Term frequency - Inverse document frequency) of the each selected words are computed, which is the weight value of each vector item. Finally, the developers are recommended based on the similarity between the developer's word vector and the vector of new bug report. We conducted an experiment on Eclipse JDT and CDT project to show the applicability of the proposed approach. We also compared the proposed approach with an existing study which is based on machine learning. The experimental results show that the proposed approach is superior to existing method.

Keywords : Bug Tracking System, Software Repository Mining, Bug Management, Bug Classification, Natural Language Processing

개발자 별 버그 해결 유형을 고려한 자동적 개발자 추천 접근법

박 성 훈† · 김 정 일†† · 이 은 주†††

요 약

소프트웨어 개발 및 유지보수 과정에서 여러 종류의 버그가 발생된다. 버그는 소프트웨어의 개발 및 유지 보수 시간을 증가시키는 주요 원인으로 소프트웨어의 품질 저하를 초래한다. 버그의 발생을 사전에 완벽하게 방지하는 것은 불가능하다. 대신 버그 질라(Bugzilla), 멘티스BT(MantisGBT), 트랙 (Trac), 질라 (JIRA)와 같은 버그 트래킹 시스템을 이용하여 버그를 효과적으로 관리하는 것이 가능하다. 개발자 또는 사용자가 발생된 버그를 버그 트래킹 시스템에 보고하면, 프로젝트 매니저에 의해서 보고된 버그는 버그 해결에 적합한 개발자에게 전달되어 해결될 때까지 버그 트래킹 시스템에 의해서 추적된다. 여기서 프로젝트 매니저가 버그 해결에 적합한 개발자를 선별하는 것을 버그 분류 작업 (Bug triaging)이라고 하며, 대량으로 발생되는 버그 리포트들을 수동으로 분류하는 것은 프로젝트 매니저에게 있어서 매우 어려운 문제가 된다. 본 논문에서는 버그 트래킹 시스템에 저장된 과거에 해결된 버그 리포트에서 개발자 별 버그 해결 유형을 추출하고, 이를 활용한 버그 분류 작업, 즉 개발자 추천 방법을 제안한다. 먼저 버그 트래킹 시스템에서 각 개발자가 해결한 버그 리포트들을 분류한 후, 자연 언어 처리 알고리즘과 TF-IDF (Term frequency - Inverse document frequency)를 활용하여 각 개발자 별 단어 리스트를 생성한다. 그 후, 새로운 버그가 발생되었을 때 코사인 유사도를 통해서 생성된 개발자 별 단어 리스트와 새로운 버그 리포트의 단어 리스트를 비교하여 가장 유사한 단어 리스트를 가지는 개발자를 추천하는 방법이다. 두 오픈 소스 프로젝트인 이클립스 JDT.UI와 CDT.CORE를 대상으로 수행한 개발자 추천 실험에서 기계 학습 모델 기반의 추천 방법보다 제안하는 방법이 더 우수한 결과를 얻은 것을 확인하였다.

키워드 : 버그 트래킹 시스템, 소프트웨어 저장소 마이닝, 버그 관리, 버그 분류, 자연 언어 처리

1. 서 론

버그는 소프트웨어 개발 및 유지보수 시간을 증가시키기 때문에 소프트웨어 시스템의 품질을 저해하는 주요 원인이 된다. 소프트웨어 개발 및 유지보수 과정에서

※ 이 논문은 2014년도 정부(미래창조과학부)의 재원으로 한국연구재단의 지원을 받아 수행된 기초연구사업임(2012R1A1A3011005).
† 준 회 원 : 경북대학교 컴퓨터학부 석사
†† 비 회 원 : 경북대학교 컴퓨터학부 박사과정
††† 종신회원 : 경북대학교 컴퓨터학부 부교수
Manuscript Received : September 15, 2014
Accepted : October 17, 2014
* Corresponding Author : Lee Eun Joo(ejlee@knu.ac.kr)

발생되는 버그를 사전에 완전히 방지하는 것은 불가능하다. 따라서, 체계적인 버그 관리를 통해서 버그 해결 프로세스를 유연하게 만드는 방법이 대안으로 선택될 수 있다. Bugzilla, MantisBT, Trac, JIRA와 같은 버그 트래킹 시스템(Bug Tracking System: BTS)[1]은 소프트웨어 개발 및 유지보수 도중 발생되는 버그들을 체계적으로 관리하기 위해서 제안되었다. 프로젝트 매니저 및 개발자는 버그 트래킹 시스템을 통해서 발생된 버그가 완전히 해결될 때까지 버그를 추적하면서, 발생 원인, 해결 방법 등 버그에 대한 정보를 세밀하게 기록할 수 있다 [1, 2, 3]. 또한 프로젝트 매니저는 버그 트래킹 시스템에 보고된 버그를 검토하여 프로젝트에 관련된 개발자들 중에서 해당 버그를 가장 잘 해결할 수 있을 것 같은 개발자를 선정하여 버그에 대한 내용을 알리고, 버그 해결을 요청한다. 즉, 프로젝트 매니저는 보고된 버그가 어떠한 유형인지, 어떤 프로젝트 컴포넌트에서 발생되었는지, 과거에 유사한 증상의 버그가 있었는지 등 여러 정보를 버그 리포트로부터 종합하여 버그를 해결하는 데 가장 이상적인 개발자를 찾는데, 이러한 프로세스를 버그 분류 작업(Bug triage)이라고 한다 [4, 5]. 하지만, 소프트웨어 프로젝트 개발의 실제 관계자뿐만 아니라 소프트웨어의 일반 사용자 역시 버그 트래킹 시스템을 통해서 자신이 발견한 버그를 보고하는 것이 가능하기 때문에 대량의 버그 리포트가 버그 트래킹 시스템에 보고된다 (평균적으로 하루에 300개 이상의 버그 [2]). 또한 일반 사용자는 소프트웨어 프로젝트에 대한 깊은 지식을 가지고 있지 않기 때문에 실제 버그가 아닌 것을 버그로 착각하여 잘못된 버그 보고를 제출하는 경우가 발생할 수 있다. 이러한 문제들은 프로젝트 매니저가 수동적으로 버그 분류 작업을 수행할 경우 많은 시간과 비용을 소모하게 하며, 나아가 소프트웨어 프로젝트의 전체 개발 비용이 증가되는 원인이 된다 [2]. 따라서, 이를 지원하기 위한 자동적인 버그 분류 작업이 필요하다.

버그 분류 자동화를 위해서 자연 언어 처리(Natural Language Processing)와 기계 학습 모델을 사용하는 것을 고려할 수 있다 [2]. 버그 트래킹 시스템에서는 과거에 해결된 버그 리포트는 해당 버그에 대한 상세한 설명(Description)과 이를 해결한 개발자를 표시한다. (Fig. 1)과 (Fig. 2)는 이클립스 프로젝트의 버그 트래킹 시스템의 버그 리포트의 한 예를 나타내고 있으며, 버그에 대한 상세 설명은 자연어로 구성되어 있는 것을 (Fig. 2)에서 확인할 수 있다. 그러므로, 자연 언어 처리를 통해서 자연어로 이루어진 상세 설명 텍스트를 좀 더 의미 있는 단어 집합으로 변환한 후, 변환된 정보들을 이용하여 SVM, Naïve Bayes와 같은 기계 학습 모델로 학습시킨다. 학습이 완료된 기계 학습 모델은 새로운 버그 리포트가 발생되었을 때 해당 버그 리포트의 상세 설명을 기준으로 과거에 가장 유사한 버그 리포트를 해결해 본 적이 있는 개발자를 추측한다. 앞선 연구에서는 이와 같은 자연 언어 처리와 기계 학습 모델을 기반으로 한 자동화된 버그 분류 작업 방법을 제안했다 [2]. 하지만, 새로운 버그 리포트의 상세 설명에 포함된 단어들과 유사한 과거 버그 리포트들이 여러 개 존재할 경우, 잘못된 추측 결과가 나타날 확률이 높아진다. 이는 버그를 설명하는 단어의 종류가 생각보다 다양하지 않아서 버그 리포트들의 상세 설명에 포함되는 단어 종류가 자주 겹치게 되는 것이 주 원인이다.

본 논문에서는 새로운 버그 리포트가 주어졌을 때 더욱 정확하게 버그 해결에 적합한 개발자를 추천하는 방법을 고려해본다. 일반적으로 생각해보자면 새로운 버그가 발생되었을 때 발생된 버그에 대해서 가장 잘 이해하고 있는 개발자에게 버그를 전달할 경우 버그 해결이 빠르게 성공할 확률이 높다. 즉, 각 개발자 별로 과거에 해결한 버그 리포트들을 수집한 후 버그 리포트의 요약 및 상세 설명에 포함된 단어들을 추출하여 개발자 별로 자주 해결한 버그 유형을 나타내는 단어 리스트 (단어 벡터)를 생성한다. 그 후, 새로운 버그 리포트가 제출되었을 때 제출된 버그 리포트의 단어 벡터와 각 개발자의 단어 벡터들과 비교해서 제출된 버그 리포트의 단어 벡터와 가장 유사한 단어 벡터를 가지는 개발자를 추천하는 것이다. 이러한 방법은 버그를 해결하는 데 적합한 개발자를 찾을 때 버그 리포트가 아닌 개발자들의 관점에 더욱 초점을 두는 것을 의미한다.

제안하는 접근법을 세부적으로 설명하자면, 제외어 제거, 어간 추출 등과 같은 자연 언어 처리를 버그 리포트에 적용 후, 각 개발자의 단어 리스트를 추출한다. 이때, 각 개발자 별로 생성된 단어 리스트 안의 각 요소는 단순히 단어의 발생 빈도수를 의미하기 때문에 이것을 단어에 대한 중요도 값으로 변화해 줄 필요가 있다. 본 논문에서는 여러 문서에서 각 단어의 중요도를 계산하는 데 사용되는 TF-IDF 계산식 [6]을 이용하여 각 단어의 중요도를 결정한다. TF-IDF를 통해서 단어 리스트의 각 단어에 대한 중요도 값이 결정된다면, 단어 리스트들 사이의 유사성을 비교할 준비가 완료된다. 본 논문에서는 코사인 유사도 계산 식 [7]을 이용하여 두 단어 리스트의 유사성을 결정한다. 코사인 유사도는 두 특성 벡터 (Feature vector) 사이

1) http://en.wikipedia.org/wiki/Bug_tracking_system

의 유사성을 계산하기 위해 사용될 수 있는 방법으로 단어 리스트 역시 특성 벡터로 간주할 수 있기 때문에 코사인 유사도를 이용한 유사성 결정은 적합하다. 본 논문에서는 제안하는 접근법의 유용성을 알아보기 위해서 두 오픈 소스 프로젝트들 (이클립스 JDT UI와 CDT CORE)을 대상으로 수행한 개발자 추천 실험에서 각 실험 프로젝트들에 대해 제안하는 접근법의 실험 결과가 기존 방법 대비 평균 회수율 56%, 47%, 그리고 평균 정확도 48%, 32%가 각각 향상되는 것을 관찰할 수 있다.

본 논문의 구성은 다음과 같다. 2장에서 본 논문의 관련 연구들을 설명하고, 3장에서는 제안하는 개발자 추천 기법에 대해서 기술한다.

이어지는 4장에서 실험 및 평가 방법과 실험 결과에 대해서 고찰해보고, 마지막 5장에서 결론과 향후 연구에 대해서 설명한다.

2. 관련 연구

본 장에서는 본 논문과 관련된 연구들에 대해서 설명한다. 먼저 버그 트래킹 시스템에 관해서 설명한 후 버그 리포트 마이닝을 활용한 여러 관련 연구들에 대해서 설명한다.

Fig. 1. Predefined bug information in a bug report

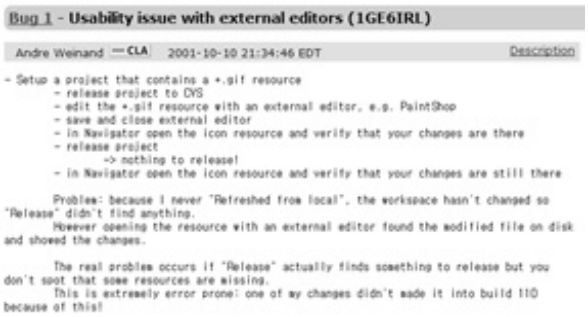

Fig. 2. The title and description of bug

2.1 버그 트래킹 시스템

버그 트래킹 시스템은 소프트웨어 개발 과정에서 생길 수 있는 버그들을 좀 더 체계적으로 관리하기 위해 사용되는 시스템이다. Bugzilla, MantisBT, Trac, JIRA와 같은 다양한 종류의 버그 트래킹 시스템이 있으며, 이러한 버그 트래킹 시스템들은 모두 거의 동일한 버그 관리 프로세스를 사용하여 버그를 관리한다. 따라서, 본 절에서는 Bugzilla를 기준으로 버그 트래킹 시스템의 특징에 대해서 설명한다.

소프트웨어 사용 및 유지보수 중에 버그가 발생했을 때 버그 발견자는 누구든지 버그 트래킹 시스템에 버그에 대한 리포트 (버그 리포트)를 제출할 수 있다. 버그 리포트는 버그를 구별하기 위한 여러 가지 정보들을 포함하고 있는데 그 중에서 자연어 텍스트로 이루어진 버그에 대한 상세 설명은 버그 리포트 마이닝을 위해서 주로 고려되는 중요한 정보이다.

버그 리포트는 버그의 현재 상태를 나타내는 상태 필드 (status field)를 가지는데 이것은 해당 버그의 진행 상황에 따라 (Fig. 3)에 나타나 있는 상태 중 하나를 가지게 된다. 버그 리포트가 최초에 보고 되면 상태 필드는 UNCONFIRMED가 되고 이는 현재 보고된 버그가 진짜 버그인지 아닌지 확정되지 않은 상태를 의미한다. 버그 리포트가 UNCONFIRMED 상태에 있을 때, 버그를 확인하는 권한을 가진 사용자들은(주로 매니저나 중요 개발자) 버그인지 아닌지를 확인하게 되고, 확실하게 버그라고 판단된 경우에 버그 리포트의 상태 필드는 NEW로 변경된다. 버그 리포트의 상태 필드가 NEW 상태일 때 버그 리포트를 해결하는데 적절한 개발자에게 할당되는 것을 기다린다. 만약 버그가 즉시 해결될 경우 상태 필드는 곧바로 RESOLVED 상태로 변경된다. 반면 버그가 즉시 해결되지 않고, 특정 개발자에게 할당되면, 상태 필드는 ASSIGNED로 변경되며 개발자에 의해서 버그 해결 작업이 시작되는 것을 의미한다. 만약에 버그 리포트를 할당 받은 개발자가 버그를 해결하지 못하는 경우에는 상태 필드는 다시 NEW로 변경되며 다른 개발자로의 재할당을 기다린다. 개발자가 버그 해결을 완료하

면 상태 필드는 RESOLVED로 변경된다. RESOLVED 상태에서는 QA (Quality Assurance)의 검증을 기다리게 되고, QA의 검증이 완료되면 상태 필드는 VERIFIED로 변경된다. 이때 버그가 성공적으로 완전히 해결되었다고 판단되는 경우에는 상태 필드는 최종적으로 CLOSED로 변경되고 이것은 버그 리포트가 닫히는 것으로 버그가 무사히 해결되었다는 것을 의미한다. 한번 닫힌 버그가 다시 발생되는 경우 해당 버그의 상태 필드가 REOPEN으로 변경되며 버그 해결 프로세스가다시 수행된다. 버그 리포트에는 상태 필드 이외에도 해결 상태 필드(resolution)가 존재하는데, 상태 필드는 버그 리포트의 전체 생명 주기에서 버그가 현재 어느 위치에 있는지 구분하기 위해 사용되는 필드인 반면, 해결 상태 필드는 버그 리포트의 해결 결과를 나타내는 필드이다. 해결 상태 필드는 버그 리포트의 상태가 RESOLVED일 때, <Table 1>의 값 중 하나로 결정된다. <Table 1>에 나열된 리스트 가운데 FIXED를 제외한 다른 필드 값들(DUPLICATE, WORKSFORME, INVALID, WONTFIX)은 모두 버그가 제대로 해결되지 않았다는 것을 의미하는데, 해당 버그가 복사된 버그, 재현이 불가능한 버그, 실제 버그가 아닌 경우 그리고 버그 해결이 불가능한 버그를 각각 의미한다.

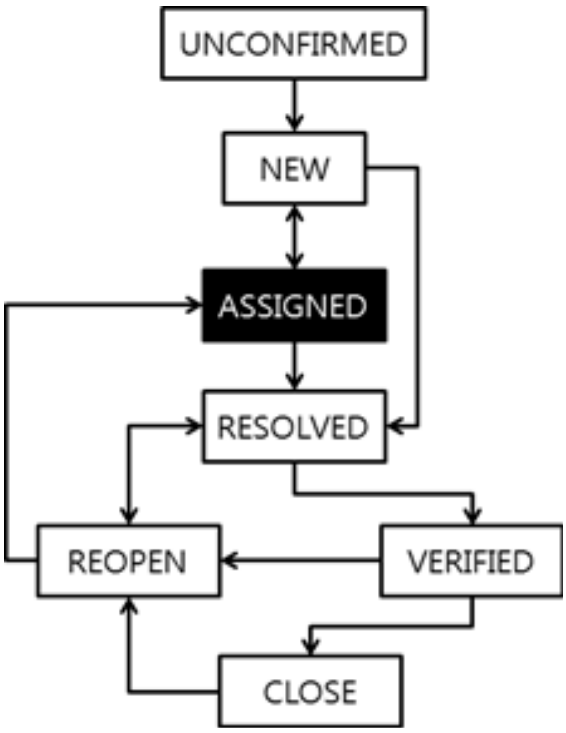

Fig. 3. The life cycle of a bug report

2.2 버그 리포트 마이닝

이 절에서는 버그 리포트 마이닝에 관련된 연구들에 대해서 소개한다. 버그 리포트 마이닝은 버그 트래킹 시스템에 저장된 버그 리포트 데이터를 활용하는 분석 방법으로 소스 코드 저장소 마이닝 [8]과 유사하다.

Table 1. The resolution status of bug report

Resolution Status	Description
FIXED	Fixed bug report
DUPLICATE	Duplicated bug report
WORKSFORME	Impossible to reproduce the bug report
INVALID	Not a bug
WONTFIX	Can't be fixed bug report

버그 리포트의 상세 설명은 해당 버그의 내용을 파악하는데 중요한 정보가 되므로, 버그 리포트 마이닝에 관련된 연구들에서 주로 사용되었다 [9]. 상세 설명은 자연어로 이루어진 텍스트 데이터이므로 분석을 위해서는 의미 있는 데이터로 변환할 필요가 있다. 이때 자연 언어 처리를 위한 알고리즘들이 주로 사용된다 [10].

버그 리포트에 대한 설명은 사람에 의해서 자연어로 작성되기 때문에 버그 리포트들의 유사성을 비교하는 것은 어려운 문제다 [11, 12]. 그러므로 Runeson 등은 복사된 버그 리포트들 검출하기 위해서 자연 언어 처리 테크닉을 적용하는 것을 고려하는 방법을 제안했고, 성공적으로 복사된 버그를 검출할 수 있다는 것을 실험에서 보였다. 반면 Wang 등은 동일한 문제를 해결하는 데 있어서 자연 언어 처리 테크닉과 함께 버그 리포트의 실행 정보 (Execution Information)를 함께 고려하는 것으로 더욱 정확하게 복사된 버그 리포트들을 검출할 수 있다는 것을 보였다 [13].

Nicholas 등은 분류기 (Classifier)를 활용한 복사된 버그 리포트 검출 방법을 고려하였다. 버그 리포트에서 단어 정보들을 추출하고, 이를 통해서 분류기의 학습을 수행한 후 학습된 분류기를 통해서 버그 리포트의 복사 유무를 결정하는 방법이다 [14]. [14]는 단어 단위의 비교 방법을 고려한 반면, Ashish 등은 버그 리포트의 유사성을 보다 세밀하게 평가할 필요성을 느끼고 하나의 단어를 세부적으로 N개의 조각으로 나누어서 비교하는 것을 고려했다 [15].

버그 리포트 마이닝은 새로운 버그 리포트의 필드 값들을 미리 예측하기 위한 목적으로도 사용되었다 [16]. 한번 해결된 버그는 여러 가지 원인으로 인해 또다시 발생될 수 있다. 이러한 버그를 재발생된 버그 (Reopen bug)라고 한다. Emad 등은 재발생될 수 있는 버그를 예측하기 위해서 버그 리포트 마이닝을 고려했다. 즉, 과거에 해결된 버그 리포트들을 재발생된 버그 리포트 그룹과 재발생되지 않은 버그 리포트 그룹으로 나눈 다음, 각 그룹에 포함된 버그 리포트들이 가지는 필드 값들을 기반으로 버그 리포트의 재발생

유무를 결정하는 방법이다 [17]. Garcia 등 또한 버그 리포트의 해결 우선 순위와 다른 버그의 해결을 지연 시키는 심각한 버그를 예측하기 위해서 Emad 등이 수행한 접근법과 유사한 방법을 고려했다. 과거 버그 리포트들에서 심각한 버그와 그렇지 않은 버그 그룹을 나누고, 각 그룹의 심각도 (severity)와 우선 순위 (priority) 필드의 값을 비교하는 방법이다 [18]. 본 논문의 목적은 새로운 버그에 대한 적합한 개발자를 찾는 것이다. 따라서, 복사된 버그 리포트를 검출하는 연구들과 버그 리포트의 특정 필드 속성 값을 예측하는 연구는 본 연구의 목적과 차이가 있다.

버그 리포트 분류 (Bug report triage)는 새로운 버그가 발생되었을 때, 발생된 버그를 가장 적절하게 해결할 수 있는 개발자에게 전달하는 작업을 의미한다 [19]. 하지만, 하루에 평균 300개 이상의 버그 리포트가 버그 트래킹 시스템에 제출되기 때문에 자동화된 버그 리포트 분류 작업이 필요하다 [2]. 그래서 Cubranic 등은 자동화된 버그 리포트 분류를 위해서 기계 학습 모델에 기반한 분류기 (Classifier)를 이용하는 것을 고려했다. 이때 버그 리포트의 상세 설명에 대한 텍스트 데이터는 분류기 학습을 위한 주 정보로 이용하고, Naïve Bayes 분류기를 이용하여 새로운 버그 리포트에 적합한 개발자를 분류하는 것을 보였다. Anvik 등은 [19]와 동일한 방법으로 여러 종류의 분류기 (결정 트리 [20], Naïve Bayes [21] 그리고 SVM [22])들을 이용한 버그 리포트 분류 실험을 수행하였고, 실험 결과를 비교하여 SVM이 버그 리포트 분류 작업에 있어서 가장 우수한 결과를 가지는 것을 보였다 [2]. 버그 리포트 분류에 관련된 연구는 새로운 버그 리포트를 해결하는 데 적합한 개발자를 찾기 위한 목적으로 이용될 수 있으며, 이것은 본 논문의 목적과 유사하다. 하지만, 본 논문에서 제안하는 방법에서는 기계 학습 머신을 사용하지 않고, 버그 리포트보다는 개발자 관점에 좀 더 초점을 둔 방법이기 때문에 Anvik 등이 제시한 방법과는 차이가 있다.

다음 장에는 본 논문에서 제안하는 방법에 대해 자세한 설명을 기술하고, 이어지는 4장에서 [2]의 방법과 비교를 위해 수행한 실험에 대해서 설명한다.

3. 개발자 별 버그 해결 유형 기반의 개발자 추천 프로세스

제안하는 개발자 추천 접근법은 각 개발자가 가지는 버그 해결 유형 전문성을 과거에 해결한 버그 리포트에서 단어 벡터 형태로 추출하고, 이를 기준으로 새로운 버그 리포트가 제출되었을 때, 제출된 버그 리포트의 유형에 가장 적합한 개발자를 찾는 방식이다. 즉, 먼저 버그 트래킹 시스템에서 과거에 해결된 버그 리포트들을 각 개발자 별로 분류하고, 자연 언어 처리 프로세스를 통해서 각 개발자 별로 분류된 버그 리포트와 새로운 버그 리포트에서 각 개발자 별 단어 리스트와 새로운 버그 리포트의 단어 리스트를 각각 생성한다. 그리고, 코사인 유사도 계산을 통해서 새로운 버그 리포트 단어 리스트와 가장 유사한 단어 리스트를 가지는 개발자를 추천한다 (Fig. 4). 아래에 이어지는 부분 절에서 각 방법에 대한 자세한 설명을 기술한다.

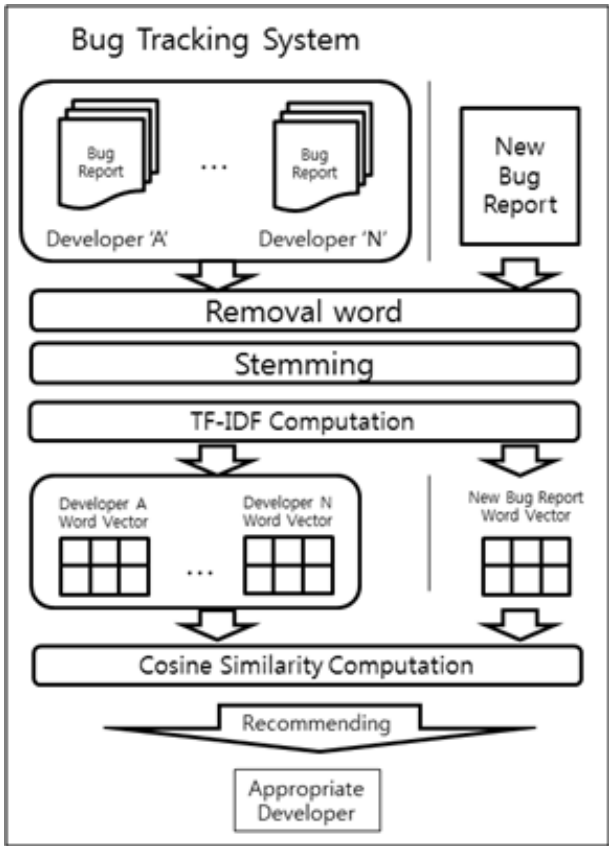

Fig. 4. The process of proposed approach

Table 2. The result of removing stop word

Original words	current line in debugger should start at center of source editor you have probably had this pr reported before but just in case
Removed stop-words	current line debugger start center source editor probably pr reported before case

Table 3. The result of stemming algorithm

Remove stop-words	current line debugger start center source editor probably pr reported before case
Words after stemming	current line debugg start center sourc editor probabl pr report befor case

Table 4. An example of BugWordSim among the bug report (398383) and developers

Developer	Similarity
snjeza_peco	0.943
samrat_dhillon	0.124
rgransberger	0.093
tobias_widmer	0.090
manju_mathew	0.087

3.1 개발자 별 버그 해결 유형 추출

1) 개발자 별 버그 리포트 분류

버그 리포트에서 STATUS 필드는 버그 리포트의 현재 해결 진행 상태를, ASSIGNED_TO 필드는 버그 리포트가 현재 누구에게 할당되었는가를, 그리고 RESOLUTION 필드는 버그 리포트의 해결 상태를 각각 나타낸다. 프로젝트 진행 중에 발생된 버그는 버그 트래킹 시스템에서 버그 리포트 형태로 관리되며, 일반적으로 버그는 개발자에 의해서 정상적으로 해결되지만, 실제로 버그가 아닌 경우, 기존에 보고된 버그와 중복된 경우, 완전히 해결하지 못한 경우 등의 이유로 인해서 해결되지 못할 수 있다. 이 때 버그 리포트의 RESOULTION 필드 값은 해결되지 못한 원인에 따라 DUPLICATE, WORKSFORME, INVALID, WONTFIX 중 하나의 값을 가지게 되는데 이러한 버그 리포트들은 개발자 별 버그 해결 유형을 추출하는 데 있어서 불확실한 정보를 제공할 수 있다. 따라서, 본 논문에서는 정확한 데이터 추출을 위해서 STATUS 필드와 RESOLUTION 필드를 고려하여 STATUS 필드가 CLOSED 값을 가지면서, RESOLUTION 필드가 FIXED 값을 가지는 버그 리포트들만을 구분하여 ASSIGNED 필드 값을 기준으로 개발자별로 버그 리포트들을 분류하는 방법을 적용한다.

2) 단어 추출

각 개발자 별로 버그 리포트를 분류한 후, 분류된 버그 리포트들의 상세 설명 텍스트에서 단어 빈도수를 추출한다. 버그 리포트의 상세 설명 텍스트는 자연어로 이루어져 있기 때문에 의미 있는 단어들만을 고려하여 추출할 필요가 있다. 본 논문에서는 [2]와 동일하게 제외어 제거, 어간 추출 자연어 처리 알고리즘을 사용한다. 먼저 버그 리포트의 상세 설명 텍스트에서 추출된 단어들에서 모든 대문자를 소문자로 변환한다. 그 후 아무런 의미를 나타내지 않는 제외어를 제거하는 작업을 수행한다. 제외어 제거를 수행한 결과 예시는 <Table 2>에서 볼 수 있다. 제외어 제거가 완료된다면 그 다음은 어간 추출을 수행한다. 본 논문에서는 어간 추출을 위해서 Martin F. Porter가 개발한 어간 추출 알고리즘을 사용한다[2] [23]. 어간 추출 알고리즘은 여러 형태를 가질 수 있는 단어들을 원형으로 변경시키기 위해서 수행되며 어간 추출 결과의 예시는 <Table 3>에서 볼 수 있다. 이러한 작업은 보다 의미 있는 개발자 별 버그 해결 유형을 나타내는 단어 리스트를 생성하기 위해서 다음에 수행되는 단어 중요도 계산 전에 반드시 거쳐야 되는 작업이다. (문장을 간소화하기 위해서 이후 내용에서는 개발자 별 버그 해결 유형을 나타내는 단어 리스트를 개발자 별 단어 리스트로 표기한다.)

3) 단어 중요도 계산

각 개발자 별로 생성된 단어 리스트는 N개의 요소를 가지는 벡터가 된다. 벡터의 각 요소는 버그와 관련된 단어를 의미하며, 요소의 값은 개발자가 해결한 버그 리포트들의 전체 텍스트에서 등장된 빈도 수가 된다. 단어의 등장 빈도수는 단순히 해당 단어가 전체 버그 리포트에서 얼마나 많이 등장했는가를 나타낼 뿐이므로 해당 단어의 중요성을 직접적으로 의미하지 않는다. 따라서, 본 논문에서는 버그 리포트에서 각 단어가 가지는 중요성을 고려하기 위해서 TF-IDF (Term Frequency-Inverse Document Frequency) 계산을 활용한다 [13]. 아래 식 (1)은 단어 빈도수를 중요도 값으로 변환하는 TF-IDF 식을 나타낸다.

단어의 중요도 :

$$TF-IDF(t,d) = TF(t,d) \times IDF(t,B) \quad (1)$$

여기서,
t: 단어
d: 개발자 단어 리스트
B: 전체 버그 리포트 집합
TF(t,d): 개발자 단어 리스트 d에서 단어 t의 총 빈도 수 값
IDF(t,B): 전체 버그 리포트 B에서 단어 t의 중요도 가중치

TF(t,d)는 개발자 단어 리스트에서 단어 t의 등장 빈도수를 나타낸다. 각 단어의 중요도는 전체 버그 리포트에서 각 단어가 차지하는 중요성을 의미하며 이는 아래 식 (2)를 통해서 결정한다.

$$IDF(t,B) = \log \frac{|B|}{|\{b \in B : t \in b\}|} \quad (2)$$

2) http://snowball.tartarus.org/

여기서,
b : 버그 리포트
|{b∈B:t∈b}| : 단어 t를 포함하는 버그 리포트 수

식 (2)에서 전체 버그 리포트 집합에서 빈번하게 등장하는 단어의 경우, 해당 단어의 IDF 값은 작아지며, 그렇지 않은 단어의 IDF 값은 커진다. 즉 IDF값이 큰 단어는 특정 버그 리포트에만 나타나는 특수한 단어를 의미한다. <Table 4>는 Eclipse JDT 프로젝트에서 개발자 Snjeza Peco가 가지는 버그 해결 유형 단어 리스트의 예를 보여준다.

3.2 개발자 추천 프로세스

3.1절의 전체 프로세스를 통해서 얻은 각 개발자 별 단어 리스트는 새로운 버그 리포트에 대한 가장 적절한 개발자를 추천하기 위해서 비교된다. 새로운 버그 리포트가 주어졌을 때 주어진 버그 리포트의 단어 리스트를 생성하는 작업은 각 개발자 별 단어 리스트를 생성하는 것과 동일한 프로세스를 통해 수행된다. 즉, 주어진 버그 리포트 텍스트에 대해서 단어를 추출하고, 각 단어의 중요도를 계산한다. 새로운 버그 리포트에 대한 단어 리스트가 생성된 후, 각 개발자의 단어 리스트와 비교해서 주어진 버그 리포트와 가장 유사한 단어 리스트를 가지는 개발자가 해당 버그 리포트를 해결하는 데 가장 적합한 개발자가 될 수 있다. 단어 리스트는 단어에 대한 특성 벡터 (Feature vector)와 유사하다. 따라서, 본 논문에서는 개발자 단어 리스트와 버그 리포트의 단어 리스트의 유사도 값을 코사인 유사도 측정 방법 [7]을 기반으로 다음 식 (3)과 같이 정의한다.

정의 1 : 개발자 단어 리스트와 버그 리포트의 유사도

$$BugWordSim(d,b) = \frac{\sum_{i=1}^{n} d[i] \times b[i]}{\sqrt{\sum_{i=1}^{n} (d[i])^2} \times \sqrt{\sum_{i=1}^{n} (b[i])^2}} \quad (3)$$

여기서,
d : 개발자의 단어 리스트
b : 새 버그 리포트의 단어 리스트
n: 전체 단어의 수
d[i]: 개발자 단어 리스트의 i번째 단어의 TF-IDF 값
b[i]: 새 버그 리포트 단어 리스트의 i번째 단어의 TF-IDF 값

위 식 (3)은 새로운 버그 리포트의 단어 리스트와 특정 개발자의 단어 리스트에 대한 유사도 값을 계산한다. 예를 들어, 이클립스 프로젝트의 398383번 식별 번호를 가진 버그 리포트를 새로운 버그 리포트라고 가정하고, 이에 대한 단어 리스트를 추출했을 때의 결과는 <Table 5>와 같다. 그리고 이클립스 개발자들의 단어 리스트를 추출한 뒤 식 (3)을 이용하여 398383번 버그 리포트의 단어 리스트와 비교한 결과는 <Table 6>과 같다. 이 결과에서 Snjeza Peco가 가장 높은 유사도 값을 가지는 것을 확인할 수 있다. 그러므로, 이 경우 398383번 버그 리포트의 할당 우선 순위는 Snjeza Peco가 가장 높다.

Table 5. The TF-IDF value of Snjeza Peco

Term	TF-IDF	Term	TF-IDF	Term	TF-IDF
newwiza	4.91998	issu	3.80666	packag	1.36827
attach	1.46040	java	0.53408	navig	4.21950
reprodu	1.36827	call	2.54944		
time	2.66258	view	1.77495		
editor	2.92928	explor	3.09104		
happen	2.92928	more	2.54944		
menu	3.09104	class	0.38136		
patch	2.78809	context	2.92928		
project	1.77495	leak	4.21950		
step	1.50851	get	3.51154		

Table 6. The TF-IDF value of bug report (398383)

Term	TF-IDF	Term	TF-IDF	Term	TF-IDF
newwiza	4.91998	issu	3.80666	packag	1.36827
attach	2.92080	java	0.53408	navig	4.21950
reprodu	1.36827	call	5.09889		
time	2.66258	view	3.54990		
editor	5.85857	explor	6.18208		
happen	2.92928	more	2.54944		
menu	6.18208	class	0.38136		
patch	5.57618	context	5.85857		
project	1.77495	leak	4.21950		
step	1.50851	get	3.51154		

4. 실험 및 평가

4.1 실험 데이터

제안하는 접근법의 유용성을 알아보기 위해서는 새로운 버그 리포트가 주어졌을 때 주어진 버그 리포트를 해결하는 데 적합한 개발자를 추천하는 실험을 수행할 필요가 있다. 그러므로, 본 논문에서는 두 개의 오픈 소스 프로젝트 이클립스 JDT.UI[3]와 이클립스 CDT.CORE[4]를 대상으로 개

3) http://projects.eclipse.org/projects/eclipse.jdt.ui

발자 추천 실험을 수행한다. 이클립스 JDT와 CDT는 Bugzilla를 이용하여 프로젝트의 버그를 관리하고 있으며, 버그 리포트 마이닝에 대한 앞선 연구들에서 실험 데이터로 자주 사용되어 왔다[2, 18, 24]. 본 논문에서는 각 프로젝트에 포함된 버그 리포트들 가운데 STATUS와 RESOLUTION 필드 값이 CLOSED이면서 FIXED인 버그 리포트들만을 실험을 위한 데이터로 고려하였다. 이는 3.1.1절에서 언급한 데이터 불확실성 문제를 해결하기 위해서이다. 결과적으로, 실험을 위해서 JDT.UI와 CDT.CORE 프로젝트에서 추출된 버그 리포트들의 수는 각각 8758개, 3408개로 <Table 7>과 같다.

Table 7. Number of bug reports in JDT.UI & CDT.CORE by year

Year	#. bug reports (JDT UI)	#. bug reports (CDT Core)
2002	2141	17
2003	1275	260
2004	1098	381
2005	1205	206
2006	851	91
2007	736	530
2008	400	436
2009	317	1195
2010	278	137
2011	181	55
2012	138	39
2013	138	61
Total	8758	3408

Table 8. Number of developers in year

Year	JDT_UI	CDT_Core
2002	23	4
2003	20	17
2004	18	19
2005	26	14
2006	15	12
2007	13	21
2008	13	26
2009	18	47
2010	11	20
2011	9	13
2012	12	14
2013	19	12

4.2 실험 방법

오픈 소스 프로젝트는 개발 또는 유지보수 중에서 기존에 참여하고 있던 개발자가 물러서거나, 새로운 개발자가 참여하는 등, 개발 인력의 변경이 빈번하게 발생되는데 이것은 오픈 소스 프로젝트 개발 참여가 자유롭기 때문이다. 그렇기 때문에 오픈 소스 프로젝트의 경우 해마다 개발에 참여하는 개발자 리스트에 차이가 날 수 있다. 실제로 <Table 8>은 매년 실험 프로젝트에서 참여한 개발자의 수가 변경되는 것을 보여주고 있다. 오픈 소스 프로젝트가 가지는 이러한 특징은 개발자 추천을 위한 버그 리포트 마이닝이 잘못된 결과를 초래하는 주요 원인이 될 수 있는데, 예를 들어, 프로젝트에서 활발하게 개발에 참여하던 개발자가 갑자기 참여가 불가능해졌을 때, 이 경우에 새롭게 발생된 버그에 대해서 해당 개발자는 추천되지 않아야 한다. 그렇지 않을 경우, 참여가 불가능한 개발자를 추천하게 되는 현상이 발생할 가능성이 높아진다. 이러한 경우, 잘못된 결과의 원인이 버그 리포트 마이닝 방법 자체에 있는지, 버그 트래킹 시스템 히스토리에 있는지 판단하기 어렵다. 그러므로, 본 실험에서는 프로젝트의 각 연도에 해당하는 버그 리포트들을 트레이닝 집합과 테스트 집합으로 구분하여 연도 단위로 실험을 나누어 평가한다. 또한, 각 실험에서 사용되는 버그 리포트의 수에 차이가 있기 때문에 각 실험에서 사용되는 트레이닝 집합과 테스트 집합의 수 역시 차이가 있다. 이것은 연도별로 얻은 실험 결과의 정확도가 실험 데이터 수에 의존적인 원인이 될 수 있다. 본 실험에서는 이러한 문제를 고려하기 위해서 주어진 데이터 셋에 대해서 10번의 트레이닝과 10번의 테스트를 수행하여 결과를 평가하는 10-Fold Cross Validation 방법을 적용하여 실험을 수행하고 결과를 비교한다 [25].

4.3 평가 방법

버그 트래킹 시스템에 있는 버그 리포트들은 개발자의 관점에서 개발자 자신이 직접 해결했거나, 다른 사람이 해결한 것으로 분류될 수 있다. 이러한 분류 결과에 대한 평가 방법은 버그 리포트 마이닝에 대한 앞선 연구들에서 수행한 방법과 동일하다 [2, 17, 25]. 따라서, 본 실험에서 앞선 연구들에서 수행한 평가 방법을 따라서 개발자 추천 결과에 대한 정확성을 평가한다. 버그 리포트 해결을 위한 개발자 추천의 결과는 <Table 9>와 같이 실제로 버그를 해결한 사람

Table 9. Confusion matrix for evaluation

	Actual Fixer	
Classified Developer	Developer	Other
Developer	$TP_{d\to d}$ (True Positive)	$FP_{o\to d}$ (False Positive)
Other	$FN_{d\to o}$ (False Negative)	$TN_{o\to o}$ (True Negative)

4) http://projects.eclipse.org/projects/tools.cdt

을 찾는 경우 (추천된 개발자와 버그를 실제로 해결한 사람이 동일한 경우 (TP)와 그 반대의 경우 (FP))가 존재하며, 버그를 해결하지 않은 사람을 찾는 경우 (추천된 개발자가 버그를 실제로 해결하지 않은 사람일 경우 (TN)와 그 반대의 경우 (FN)))가 존재한다. 그러므로, 전체 추천 결과에 대한 회수율, 적중률, 정확도 값은 아래 수식으로 결정할 수 있다.

회수율 (Recall) :

$$R(d) = \frac{TP_{d\to d}}{TP_{d\to d} + FN_{d\to o}} \quad (4)$$

적중률 (Precision) :

$$P(d) = \frac{TP_{d\to d}}{TP_{d\to d} + FP_{o\to d}} \quad (5)$$

정확도 (Accuracy) :

$$A(d) = \frac{TP_{d\to d} + TN_{o\to o}}{TP_{d\to d} + FP_{o\to d} + FN_{d\to o} + TN_{o\to o}} \quad (6)$$

여기서,

d : 버그 리포트를 해결한 개발자

o : 버그 리포트를 해결하지 않은 개발자 (other)

$TP_{d\to d}$: 개발자 d가 해결한 것을 정확하게 분류한 경우

$FP_{o\to d}$: 다른 개발자가 해결한 것을 개발자 d로 분류한 경우

$FN_{d\to o}$: 개발자 d가 해결한 것을 다른 개발자로 분류한 경우

$TN_{o\to o}$: 다른 개발자가 해결한 것을 정확하게 분류한 경우

4.4 실험 결과

제안하는 접근법의 타당성을 증명하기 위해서 앞선 연구 [19, 2]에서 제안한 기계 학습 모델 기반의 개발자 추천 접근법과 비교할 필요가 있다. [19, 2]에서는 Naïve Bayes, SVM, 결정 트리 기반의 분류기 (Classifier)를 이용하여 새로운 버그 리포트를 해결하는 데 적합한 개발자를 결정하는 방법을 보였다. 버그 리포트에 자연 언어 처리 알고리즘을 적용하여 버그 리포트별 특성 벡터 (Feature vector)를 생성한 후, 생성된 특성 벡터를 기반으로 분류기를 학습시킨다. 학습된 분류기는 프로젝트에 참여하는 개발자들 중에서 새로운 버그 리포트에 적절히 할당되어야 되는 개발자를 분류한다.

본 실험에서는 JDT.UI와 CDT.CORE에서 연도별 개발자 추천을 수행한다. 각 프로젝트에 대해서 기존 방법과 제안하는 방법으로 각각 개발자 추천을 수행하고, 각 연도에 대해서 얻은 회수율, 적중률, 정확도 결과를 비교한다. <Table 10>와 <Table 11>에서는 JDT.UI와 CDT.CORE에서 각 접근법으로 실험한 결과를 보여준다.

제안하는 방법이 다른 방법들 보다 높은 회수율, 적중률, 정확도 값을 가지는 것을 두 프로젝트에서 모두 확인하였다. 특히, JDT.UI에서는 제안하는 방법과 다른 방법들과의 평균 회수율, 적중률, 정확도가 각각 최소 57%, 49%, 6% 차이가

Table 10. The experiment result of JDT.UI

	SVM			Naïve Bayes			C4.5			Proposed approach		
Year	R	P	A	R	P	A	R	P	A	R	P	A
2002	0.08	0.62	0.82	0.15	0.15	0.94	0.18	0.39	0.9	0.48	0.98	0.93
2003	0.06	0.4	0.76	0.2	0.18	0.93	0.21	0.32	0.91	0.62	1	0.94
2004	0.07	0.55	0.82	0.19	0.19	0.92	0.25	0.32	0.91	0.68	0.99	0.94
2005	0.06	0.66	0.79	0.14	0.15	0.94	0.17	0.3	0.93	0.57	0.96	0.93
2006	0.12	0.39	0.78	0.13	0.16	0.9	0.13	0.24	0.85	0.66	0.98	0.94
2007	0.13	0.71	0.77	0.22	0.2	0.9	0.25	0.32	0.88	0.68	1	0.93
2008	0.07	0.23	0.51	0.12	0.21	0.8	0.1	0.17	0.82	0.69	0.93	0.94
2009	0.07	0.69	0.74	0.11	0.17	0.87	0.12	0.26	0.88	0.83	1	0.95
2010	0.09	0.39	0.74	0.15	0.27	0.85	0.1	0.22	0.76	0.82	1	0.94
2011	0.13	0.48	0.59	0.21	0.39	0.8	0.13	0.23	0.76	0.88	1	0.95
2012	0.08	0.54	0.54	0.12	0.23	0.78	0.08	0.12	0.86	0.87	1	0.93
2013	0.06	0.33	0.67	0.18	0.32	0.89	0.12	0.17	0.9	0.93	1	0.99
Average	0.085	0.50	0.71	0.16	0.22	0.88	0.15	0.255	0.86	0.73	0.99	0.94

Table 11. The experiment result of CDT.CORE

	SVM			Naïve Bayes			C4.5			Proposed approach		
Year	R	P	A	R	P	A	R	P	A	R	P	A
2002	0.21	0.19	0.47	0.32	0.4	0.67	0.35	0.5	0.71	1	1	1
2003	0.09	0.65	0.74	0.2	0.25	0.92	0.18	0.33	0.89	0.64	0.83	0.93
2004	0.06	0.82	0.73	0.19	0.18	0.9	0.14	0.23	0.9	0.54	0.77	0.93
2005	0.07	0.31	0.71	0.25	0.29	0.89	0.14	0.23	0.9	0.58	0.78	0.93
2006	0.08	0.28	0.51	0.15	0.3	0.82	0.08	0.16	0.78	0.62	0.8	0.88
2007	0.05	0.47	0.63	0.16	0.21	0.92	0.12	0.23	0.9	0.58	0.81	0.92
2008	0.04	0.66	0.65	0.15	0.28	0.92	0.14	0.21	0.94	0.63	0.86	0.93
2009	0.04	0.66	0.65	0.11	0.1	0.96	0.06	0.1	0.96	0.59	0.79	0.95
2010	0.04	0.66	0.65	0.1	0.2	0.88	0.1	0.25	0.87	0.67	0.74	0.96
2011	0.08	0.22	0.22	0.12	0.19	0.85	0.07	0.09	0.83	0.71	0.76	0.91
2012	0.07	0.2	0.5	0.03	0.03	0.79	0.07	0.1	0.86	0.5	0.55	0.79
2013	0.08	0.25	0.25	0.08	0.1	0.84	0.17	0.17	0.83	0.55	0.62	0.89
Average	0.08	0.45	0.56	0.16	0.21	0.86	0.14	0.22	0.86	0.63	0.78	0.92

나는 것을 알 수 있고, CDT.CORE에서는 47%, 33%, 6% 차이가 나는 것을 알 수 있다.

각 프로젝트의 결과에서 기존 방법으로 얻은 회수율 값이 적중률에 비해 매우 낮은 수치를 기록하였다. 기본적으로 회수율과 정확도 값 사이의 관계에서 트레이드 오프 관계가 성립하지만, 트레이드 오프 관계를 고려한다고 해도 회수율이 너무 낮은 값을 가지고 있는데, 이것은 [2]에서 언급한 원인과 동일하다. 이러한 원인은 회수율 계산 방법에 존재한다고 [2]에서 언급하였는데, 자세하게 언급하자면 각 분류기가 하나의 버그 리포트에 대해서 오직 한 명의 개발자만을 추천하고, 그것에 기반으로 회수율을 계산하기 때문이다. 본 실험에서의 회수율 계산 역시 [2]와 동일한 방법으로 수행하였기 때문에 이와 같은 결과가 나왔다는 것을 알 수 있다.

4.5 고찰

새로운 버그 리포트에 대한 적합한 개발자를 추천하는 문제에서 본 논문에서 제시하는 접근법이 기존 방법보다 더 우수한 실험 결과를 보였다. 이러한 결과는 각 개발자 별 버그 해결 유형 정보를 기반으로 하는 방법이 특정 버그 리포트에 대한 개발자를 찾는 문제를 해결하는 데 더욱 적합하기 때문이라고 생각된다.

본 논문에서 비록 오픈 소스 프로젝트인 JDT.UI와 CDT. CORE에서 얻은 실험 결과에서 제안하는 방법의 타당성을 알아볼 수 있었지만, 다른 모든 종류의 프로젝트에서 동일한 결과를 얻을 수 있을 것이라는 보장을 할 수 없다. 그러므로, 오픈 소스 프로젝트만이 아닌 상업 프로젝트에 대해서 제안하는 방법을 적용해 볼 필요가 있다.

5. 결론 및 향후 연구

버그는 프로그램을 개발 및 유지보수 과정에서 빈번하게 발생되며, 이는 개발 및 유지보수 품질을 저해하는 주요 원인이다. 따라서, 규모가 큰 프로젝트에서는 버그를 체계적으로 관리하기 위해서 버그 질라 같은 버그 트래킹 시스템을 구축하고 이를 활용하여 발생되는 버그를 체계적으로 관리한다. 하지만, 버그 트래킹 시스템이 오히려 프로젝트의 개발 기간을 지연시키는 요소가 될 수 있는데, 그 이유는 하루에 최소 300개 이상 보고되는 버그 리포트들을 분류하는 데 있어서 대량의 작업 시간이 필요하기 때문이다. 이를 고려하기 위해서 과거 버그 리포트 데이터와 분류기를 기반으로 한 자동화된 버그 리포트 분류 방법이 제안되었다. 본 논문에서는 버그 리포트 마이닝을 적용하는 방법으로 과거에 각 개발자가 해결한 버그 리포트에서 개발자 별 버그 해결 유형 정보를 나타내는 개발자 단어 리스트를 생성하고, 새로운 버그 리포트의 단어 리스트와 각 개발자 단어 리스트의 유사성을 코사인 유사도 측정으로 계산하여 가장 유사한 개발자를 찾는 방법을 제안했다. 두 오픈 소스 프로젝트 JDT.UI와 CDT.CORE를 대상으로 수행한 개발자 추천 실험을 통해서 제안하는 방법이 기존 방법보다 평균 회수율, 적중률, 정확도가 JDT.UI에서는 57%, 49%, 6%, CDT.CORE에서는 47%, 33%, 6% 정도 각각 차이가 나는 것을 확인하였다. 향후 연구에서 좀 더 다양한 종류의 프로젝트를 대상으로 개발자 추천 실험을 수행하여 제안하는 방법의 개선점 방향을 확인하고, 활용에 대한 일반화 가능성에 대해서 알아볼 예정이다.

References

[1] E. S. Raymond, “The cathedral and the bazaar”, Transactions on Knowledge, Technology & Policy, Vol.12, pp.23-49, 1998.

[2] J. Anvik, L. Hiew, and G. C. Murphy, “Who should fix this bug?”, in Proceedings of the 28th international conference on Software engineering, pp.361-370, 2006.

[3] G. Canfora and L. Cerulo, “How software repositories can help in resolving a new change request,” in Workshop on Empirical Studies in Reverse Engineering, pp.99-101, 2005.

[4] G. Jeong, S. Kim, and T. Zimmermann, “Improving bug triage with bug tossing graphs,” in Proceedings of the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering, pp.111-120, 2009.

[5] P. Bhattacharya, I. Neamtiu, and C. R. Shelton, “Automated, highly-accurate, bug assignment using machine learning and tossing graphs,” Transaction on Systems and Software, Vol. 85, No.10, pp.2275-2292, 2012.

[6] G. Salton and M. J. McGill, “Introduction to modern information retrieval,” 1st ed., McGraw-Hill, 1983.

[7] P. N. Tan, M. Steinbach and V. Kumar, “Introduction to Data Mining,” 1st ed., Addison Wesley, 2005.

[8] T. Zimmermann, P. Weiβgerber, S. Diel, and A. Zeller, “Mining version histories to guide software changes,” IEEE Transactions on Software Engineering, Vol.31, pp.429-445, 2005.

[9] Alenezi, Mamdouh, Kenneth Magel, and Shadi Banitaan, “Efficient bug triaging using text mining,” *Journal of Software,* Vol.8, No.9, pp.2185-2190, 2013.

[10] T. Joachims, “Text categorization with support vector machines: Learning with many relevant features,” Springer Berlin Heidelberg, pp.137-142, 1998.

[11] L. Hiew, “Assisted detection of duplicate bug reports,” M.S. dissertation, University of British Columbia, Vancouver, 2006.

[12] P. Runeson, M. Alexandersson, and O. Nyholm, “Detection of duplicate defect reports using natural language processing,” in Proceedings of the 29th International Conference on Software Engineering, pp.499-510, 2007.

[13] X. Wang, L. Zhang, T. Xie, J. Anvik, and J. Sun, “An approach to detecting duplicate bug reports using natural language and execution information,” in Proceedings of the 30th International Conference on Software Engineering, pp.461-470, 2008.

[14] Jalbert, Nicholas, and Westley Weimer, “Automated duplicate detection for bug tracking systems,” in Proceedings of Dependable Systems and Networks With FTCS and DCC, pp.52-61, 2008.

[15] Sureka, Ashish, and Pankaj Jalote, “Detecting duplicate bug report using character n-gram-based features,” in Proceedings of Software Engineering Conference (APSEC), pp.366-374, 2010.

[16] A. Lamkanfi, S. Demeyer, E. Giger, and B. Goethals, “Predicting the severity of a reported bug,” in Proceedings of the Working Conference on Mining Software Repositories, pp.1-10, 2010.

[17] Shihab, Emad, et al., “Predicting re-opened bugs: A case study on the eclipse project,” in Proceedings of Reverse Engineering (WCRE), pp.249-258, 2010.

[18] Valdivia Garcia, Harold, and Emad Shihab, “Characterizing and predicting blocking bugs in open source projects,” in Proceedings of the 11th Working Conference on Mining Software Repositories, pp.72-81, 2014.

[19] D. Cubranic and G. C. Murphy, “Automatic bug triage using text categorization,” in Proceedings of the Sixteenth International Conference on Software Engineering & Knowledge Engineering, pp.92-97, 2004.

[20] R. Quinlan, “C4.5: Programs for Machine Learning”, Morgan kaufmann, Vol.1, 1993.

[21] G. H. John and P. Langley, “Estimating continuous distributions in Bayesian classifiers,” in Proceedings of the 11th Conference on Uncertainty in Artificial Intelligence, pp.338-345, 1995.

[22] S. R. Gunn, “Support Vector Machines for classification and regression,” Technical report, University of Southampton, 1998.

[23] R. A. Baeza-Yates and B. A. Ribeiro-Neto, “Modern Information Retrieval,” Addison-Wesley, Vol.463, 1999.

[24] Tian, Yuan, David Lo, and Chengnian Sun, “Drone: Predicting priority of reported bugs by multi-factor analysis,” in Proceedings of Software Maintenance (ICSM), 2013 29th IEEE International Conference on, pp. 200-209, 2013.

[25] Kim, Sunghun, E. James Whitehead, and Yi Zhang, “Classifying software changes: Clean or buggy?,” IEEE Transactions on Software Engineering: Vol.34, Issue.2, pp. 181-196, 2008.

박 성 훈

e-mail : 2012307116@knu.ac.kr

2014년 경북대학교 컴퓨터학부 석사

관심분야 : Recommendation of Bug Fixers, Mining Software Repository

김 정 일

e-mail : 2009307043@knu.ac.kr

2011년 경북대학교 전자전기컴퓨터학부 석사

2011년~현 재 경북대학교 컴퓨터학부 박사과정

관심분야 : Automated Bug Fixing, Mining Software Repository

이 은 주

e-mail : ejlee@knu.ac.kr

1997년 서울대학교 계산통계학과 학사

1999년 서울대학교 전산과학과 석사

2005년 서울대학교 전기컴퓨터공학부 박사

2005년~2005년 서울대학교 공과대학 BK 박사 후 연구원

2005년~2006년 삼성종합기술원 전문연구원

2006년~현 재 경북대학교 컴퓨터학부 부교수

관심분야 : Mining Software Repository, Change Pattern, Software Metric, Software Evolution