1

INTRODUCTION

버그리포트로부터 버그 위치를 자동적으로 찾을 수 있는 방법 필요

1. 유지비용 > 개발비용
2. 버그리포트에 명시된 오류 상황을 만든 뒤 분석 → 시간 오래 걸림 (특히 큰 project)

유사연구

- 2000s: Change Impact Analysis (CIA), 소스파일 버전마다의 차이를 분석
- 2010s: Spectrum-Based Fault Localization (SBFL)
- Recently(2016): Information Retrieval, SW의 텍스트 영역을 조사해서 위치 찾고 개발자 확인, 충격 분석

IR 장점
- 상대적으로 computation cost 적음
- CIA, SBFL보다 정확함

IR 특징
- 버그 리포트를 쿼리로 취급
- 소스코드 파일들이 document collection을 이룬다

정확성을 높이기 위해 사용되는 것들
- 앞서 고쳐진 bug들과의 유사도
- 소스코드 파일을 단순 문서로 취급하는 것이 아니라 structure information을 사용
- 버전 히스토리 분석
- Stack trace 분석

해당 연구에서 제안

활용: 텍스트, stack trace, comments, structured info, 코드 변화 히스토리

1. IR(rVSM[19])으로 버그리포트와 소스파일의 텍스트 유사성 비교, 소스파일의 Structured information을 통합 ([20]에서 효율성 입증)
2. 앞서 고쳐진 버그리포트들의 유사성 분석
3. 버그 리포트가 stack trace 포함하면 의심 가는 파일 골라 내기 위해 분석
4. 소스코드파일 히스토리 분석-파일과 method
5. 위의 4개 통합해서 버그리포트마다 의심 가는 파일 localize. File level로 output
6. 랭크된 의심가는 파일에서 파일과 버그리포트의 method 유사성 분석
7. 의심가는 method가 랭크됨. Method level로 output

Contribution

1. content, stack traces, 버그리포트의 comment, 소스파일의 structured info, 소스파일의 히스토리 활용. File level 뿐만 아니라 Method Level
2. 분석한 정보들의 influence rate 변수의 최적화 범위 찾음
3. Localization 정확성 높이기 위해 comments, fixed method, 각각 버그 리포트의 fixed commits 를 추가함.

목차

BACKGROUND

1. 버그/issue 해결되는 과정
2. 버그리포트
3. IR기반 localization의 일반적인 process와 테크닉
4. SCM - Code change history

2.1. 버그와 issue management process

1. 유저가 버그 발견하면 알린다.
2. 오류 위치 찾으려는 개발자에게 배정
3. 개발자는 버그가 생기는 상황을 다시 만들려고 함
4. 원인과 장소를 찾으면 수정해서 테스트 해봄
5. 테스트에서 오류 없으면 (개발자가)resolve로 상태 변환시킴
6. 다른 teste가 오류 상황 체크하고 오류가 없으면 verify한다.(상태 바꿈)
7. Reporter에게 verify한 알림이 감

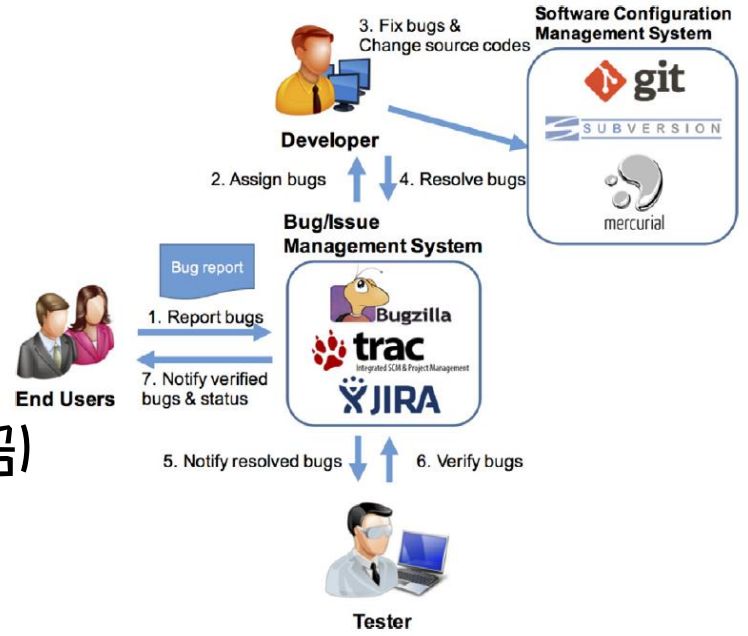


Fig. 1. Bug/issue management process.

2.2 Bug Report

- 개발자들, 주로 버그 생기는 상황을 다시 만들어내서 원인 뭔지, 소스파일 어디가 잘못된 것인지 찾는다.
- 버그 리포트: 버그 ID, 요약, 설명, 제출 날짜, 수정된 날짜, 상태, 등 관련 항목. 언제 exception이 생겼는지 알 수 있는 Stack trace포함하고 있는 리포트도 있다.
  ex) 버그가 발생한 상황 만들고 버그 리포트의 class name이나 method name에 기반해 있다고 가정
- 버그 리포트에 있는 comment에서 오류 상황에 대한 추가적인 정보를 얻을 수 있다.
- 저장소에 있는 비슷한 버그 리포트를 찾는다. 비슷한 버그 리포트를 찾으면 수정된 파일을 후보로 생각.

2.3. IR-based bug localization

- 각각의 버그 리포트는 search query이고 소스파일은 document 묶음이다.
- 각 버그리포트에 대해 의심 가는 소스파일의 순위를 알려준다.
- (전형적인) 전처리 3 step
  1. Text normalization
  2. Stopword removal
  3. Stemming
- 소스파일과 버그리포트 전처리해서 분해할 수 있는 용어를 만들고 유사도를 계산한다.

1. Text normalization

- 점(.) 제거, 용어 tokenization, identifier 분해
- 소스파일의 AST(Abstract Syntax Tree)이용해 identifier 뽑아낸다. Identifier는 Camel Case splitting[27]로 분해됨.
  ex) combine-AnalyzedScore → combine, Analyzed, Score

2. Stopword
   - 주제와 관련 없는 용어들을 stopword list에서 제거.
   - 정확성 개선하고 가짜 match 줄일 수 있다.
   - Programming language keyword도 stopword

3. Stemming
   - 단어들의 root 형태만 남긴다. Porter stemming algorithm 사용함

- 전처리 과정 끝나면 소스파일에 index 생긴다

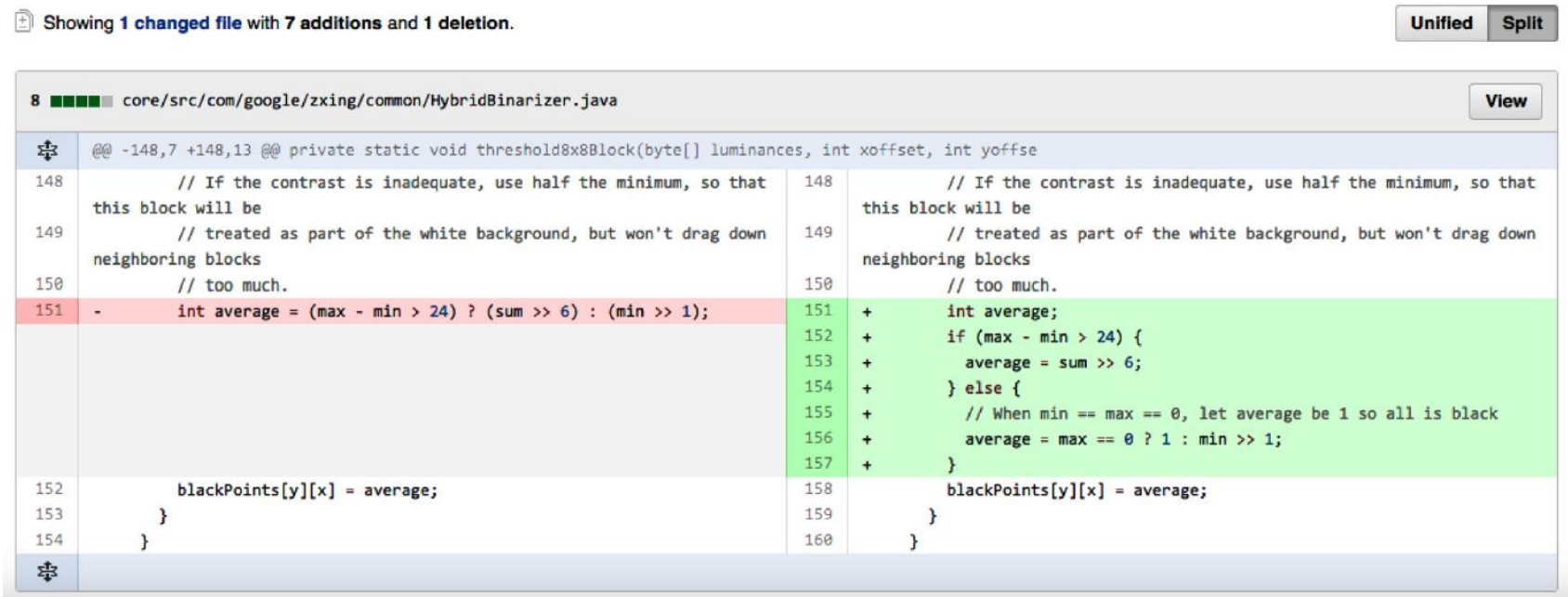
  ex) TF(term frequency;해당 문서에 특정 용어가 나오는 횟수), DF(document frequency; 해당 용어가 나오는 문서 수), IDF(inverse DF; 버그 리포트와 소스파일 내용 유사도 계산할 때 쓰임)
- 버그 리포트랑 소스 파일 TF,IDF 벡터로 표현된다.
- IR 모델: SUM, LDA, LSI, VSM, rVsm(다른 모델보다 성능이 좋다)

2.4. Code change history

BACKGROUND

SCM system(Software Configuration Management)는 개발의 중심에 있다.
개발팀은 SCM, 버전 히스토리 관리: SCM시스템 (git,svn,mercurial) 활용
Ex) 깃허브 프로젝트에서 코드 바뀐 히스토리를 볼 수 있다.

→ 의심가는 파일과 method를 추적할 수 있는 (분해할 수 있는) 데이터를 제공
→ 후보를 추적하고 나면 개발자들은 포커스를 맞춰서 테스트할 수 있다.



3

APPROACH

3.1. Analyzable inputs

3.1. analyzable inputs

APPROACH

Analyzable Inputs

1. 버그 리포트
2. 소스 파일들
3. 유사한 해결된 버그 리포트
4. 소스코드 change history (commit messages & chang된것 간의 차이)

- 버그 리포트가 포함하고 있는 것: report date, 오류 일어났을 때 시나리오, product version, 리포터, 상태 등
- 오류 발생 → (유저가) 버그 리포트 시나리오를 stack trace와 제출 → (개발자)버그시나리오 이해하고, 버그리포트의 핵심 단어를 포함하는 의심 가는 소스파일 찾는다
- Stack trace

  버그 발생 소스파일과 라인을 찾는데 중요. Method나 class 이름으로 line 찾을 수 있다.
- 버그리포트의 코멘트에도 버그리포트에는 명시되지 않은 추가적인 정보가 있다. Stack trace나 crash log 포함
- 고쳐진 것 중 유사한 버그 리포트가 있으면 파일 후보에 추가
- SCM 시스템의 소스코드 파일 히스토리와 commit log 찾아 최근 수정된 것 중에 원인이 없는지 찾는다.

  두 버전의 파일, method, line간의 차이 분석해 정확한 bug location 찾는다.

3.2. Analysis flow of BLIA

APPROACH

rVSM 버그 리포트 분석 + IR모델 버그 로케이터

BLIA1.0 → BLIA1.5 : method level, 버그리포트의 comment 포함

버그 리포트의 상세 설명 추가

Stored data

Fixed method: Fixed file과 commit log에서 추출

소스파일의 Structured information에 대한 통합적인 분석

버그리포트의 stack trace 분석

Commit log 분석

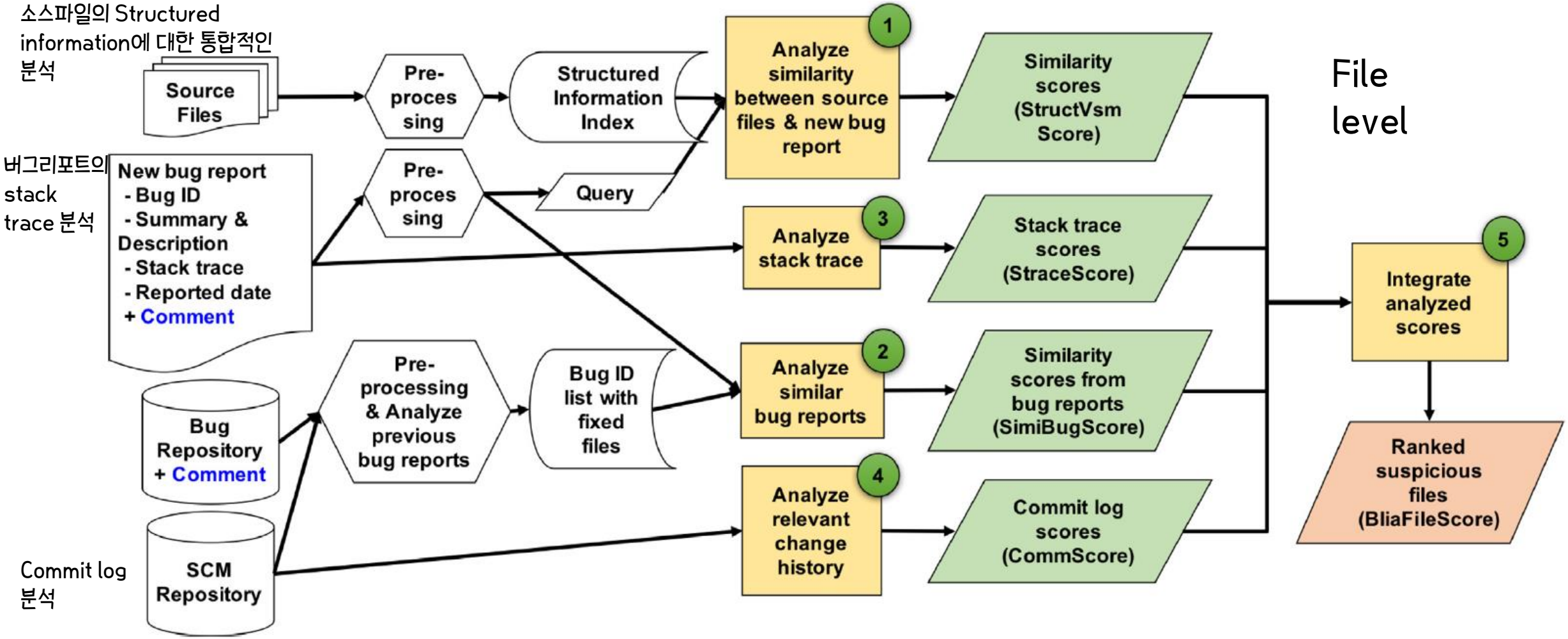


Fig. 3. File level analysis flow of BLIA. (For interpretation of the references to color in the text, the reader is referred to the web version of this article.)

3.2. Analysis flow of BLIA

step1 : 소스파일과 새 버그 리포트의 유사도 분석. 전 처리 과정에서 소스파일에 대한 structured information 얻을 수 있다. 새 버그 리포트도 전 처리해서 query 만든다. 쿼리를 structured information에 보내서 점수(StructVsmScore: 버그리포트와 소스코드 사이의 유사도)를 매긴다.

step2 : 수정된 버그 리포트와 새 버그리포트의 유사도 분석. 버그리포트와 코멘트 전처리과정 거친다. 각각 리포트에서 수정된 파일 저장 해놓는다. 비슷한 버그 리포트를 찾으면 수정된 파일에 대한 SimiBugScore 계산.

step3 : 버그 리포트에 stack trace가 있으면 소스 파일에 대한 정보를 추출. 소스코드 파일에 대한 Strace Score(오류 의심 가는 파일) 매긴다.

step4 : SCM에서 새 버그리포트 작성 전 근래에 commit된 파일이나 commit log 메시지를 찾는다. Commit 된 파일에 대한 CommScore 계산. (관련 소스파일 → 관련 메소드) Method level analysis에 사용됨

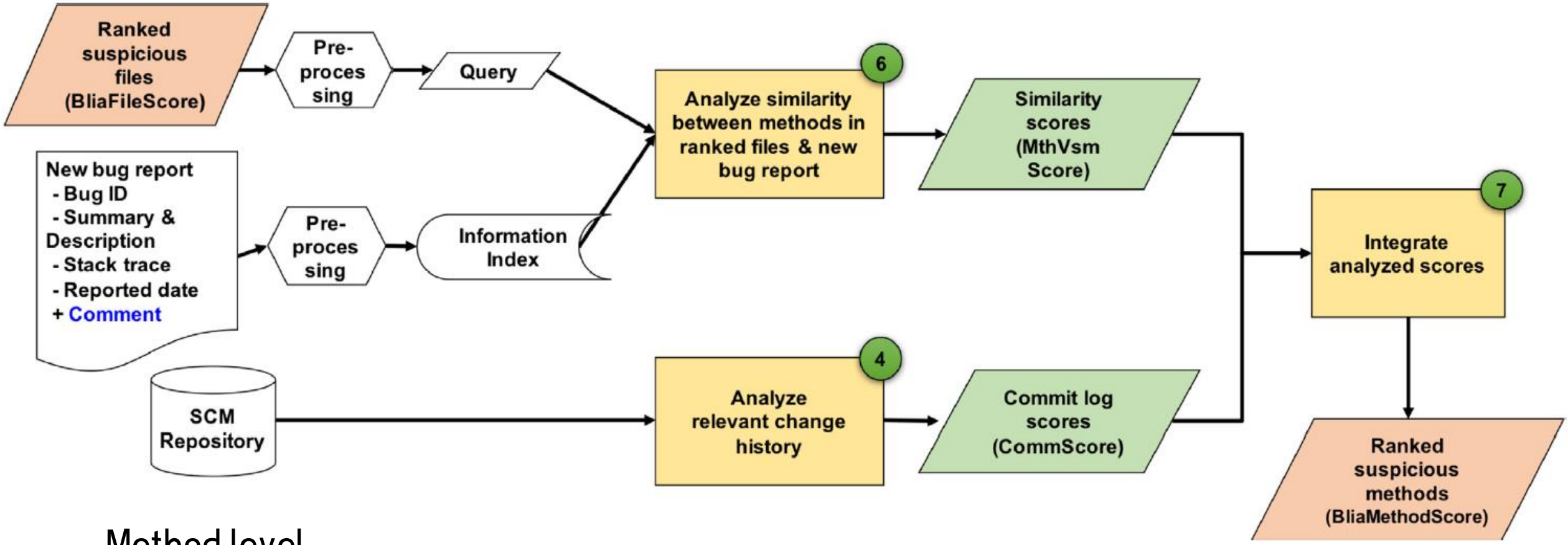
step5 : 위에 4개의 점수를 3개의 control parameter로 통합해서 각각 소스파일에 대한 BliaFileScore 계산. 파일 레벨로 결함 위치 찾을 수 있다.

File level

3.2. Analysis flow of BLIA

APPROACH

Step 5에서 얻은 파일



Method level

Fig. 4. Method level analysis flow of BLIA. (For interpretation of the references to color in the text, the reader is referred to the web version of this article.)

step6 : Step5에서 얻은 파일들 전처리 과정 거쳐서 method 정보를 query로 사용. 버그리포트도 전 처리 과정 거치는데 step1의 query 대신에 information index 만든다. 메소드와 버그 리포트 간의 MthVsmScore 계산

step7 : Step4에서 얻은 각 메소드의 commit log 점수와 메소드-버그리포트 유사도 점수를 control parameter 사용해서 합한다. 각각 메소드에 대한 의심 가는 점수 BliaMethodScore를 얻는다.

3.3. Architecture of BLIA

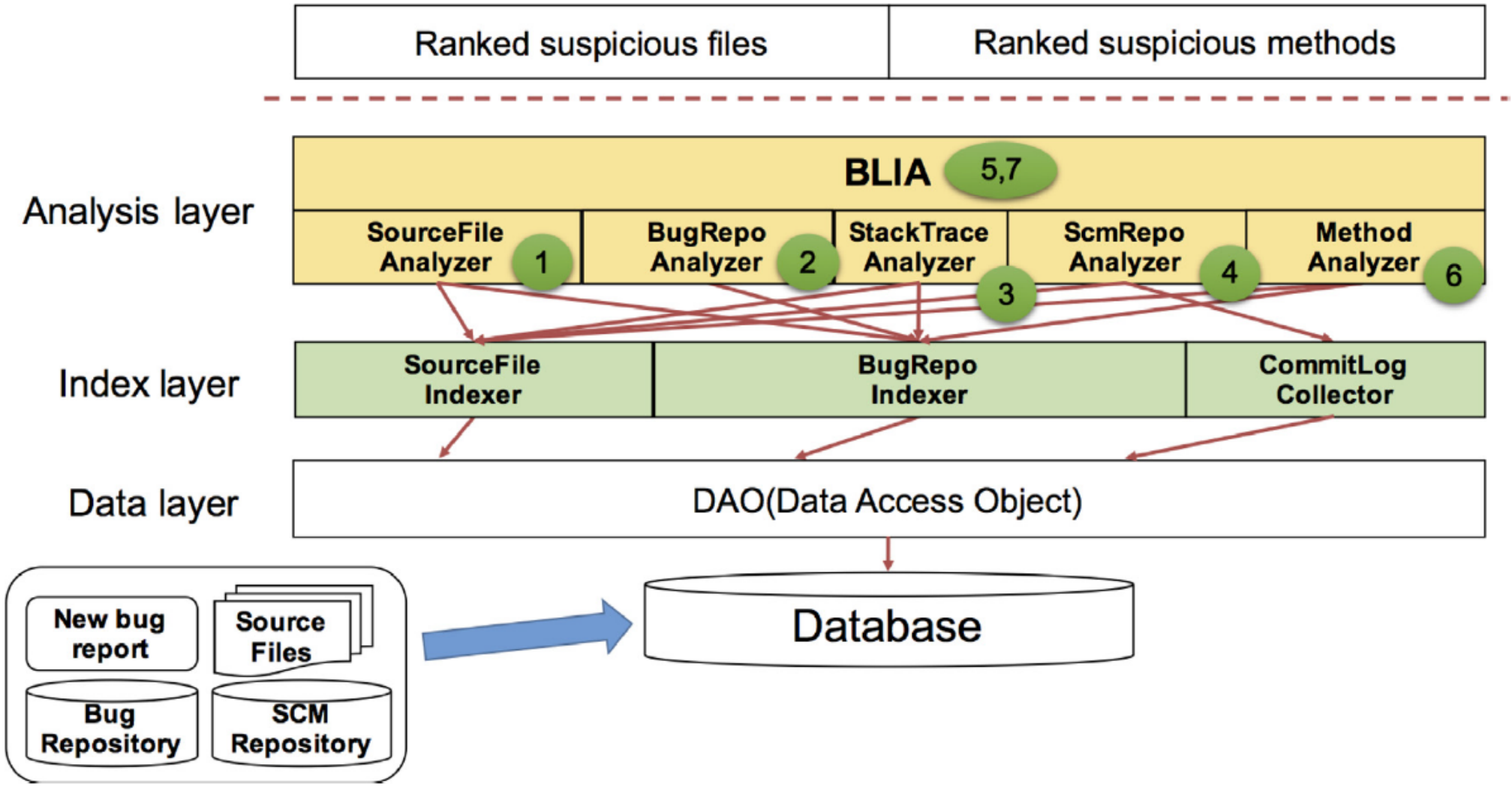


Fig. 5. BLIA architectural design. (For interpretation of the references to color in the text, the reader is referred to the web version of this article.)

소스파일 인덱스, 버그 리포트인덱서, 커밋로그 콜렉터를 인덱스 레이어에서 사용해서 소스파일과 버그리포트의, 커밋 로그 메시지의 용어들을 파싱한다.
데이터 레이어에서 데이터를 내부 데이터베이스에 저장한다. Embedded 된 H2데이터베이스 (h2database.com/html/main.html)는 가볍고 설치가 필요없다. 이걸로 인덱스된 데이터를 관리하고 데이터를 분석한다.
Analysis 레이어의 5개의 모듈은 인덱서 모듈을 불러 각 스텝의 suspicious score을 계산한다.
앞서 언급된 각 스텝들은 이 analyzer 모듈을 사용해서 각각 점수를 계산한다. BLIA모듈은 분석된 점수를 통합하고 suspicious 파일과 메소드에 대한 최종점수를 계산한다. 3.5.에서 BLIA analysis 모듈에 대한 알고리즘 상세 순서 설명

3.4. Extension of bug repositories

본 연구에서 bug repository 늘렸다.

1. 버그리포트의 코멘트를 새 버그리포트와 수정된 버그리포트의 유사도 계산할 때 사용함. 가끔 stack trace가 포함된 경우도 있었다.
2. 각 버그 리포트에 대한 수정된 method를 추가. Jgit 라이브러리를 사용해 SCM의 히스토리를 참고해서 수정된 메소드를 추출.

Jgit는 오픈소스, 자바라이브러리 깃 버전 컨트롤 시스템

SCM에서 소스파일 change history와 버그리포트의 상태 히스토리에서 git repository의 fixed commit를 찾음. 그리고 나서는 추가되거나 삭제되거나 수정된 라인을 Jgit의 git diff 사용해서 추출함. 변경된 라인에 기반해서 변경된 메소드를 AST를 사용해 계산. 수정된 메소드의 메소드 이름, 리턴타입, 변수를 추출해서 버그 리포지터리에 추가함

3. 각 버그 리포트에 대한 Fixed commit은 추가적인 정보를 제공한다. 이 정보가 BLIA에 직접 사용되고 있지는 않지만 다른 리서치에서 사용될 수 있다. 정확도를 높이기 위해. Suspicious change나 commit들을 localize 할 수 있다.

소스파일은 document collection이룬다. 각각의 버그 리포트는 IR-based 버그 localization의 쿼리로 사용됨. 각 소스파일을 하나의 문서로 취급함으로서 파일의 용어와 용어의 벡터가 만들어진다. 텍스트 파일로 취급하는 대신 소스파일인덱서가 각 파일의 AST를 활용해 소스파일을 파싱해서 class name, method name, variable name, commet로 나누어 준다.
BLIA1.0.과 비교했을 때 소스파일인덱서가 메소드 정보까지 계산. 소스파일의 이런 Structured information이 suspicious file 랭킹의 정확도를 높여준다. 실제 버그리포트를 보면 합리적인데 message나 error case의 log에 class name과 method name이 포함되어 있다.
AST에서 얻은 모든 아이덴티파이어는 camel case splitting 기법으로 구성단어로 분리된다. 분리된 용어와 풀 아이덴티파이어 모두 인덱스한다. 둘 다 인덱스하는게 효율적임을[20]에서 입증함.

preprocessing
전처리 과정에서 stopword 제거 할 때 자주 사용되는 프로젝트 키워드 제거함(args, param, string). 패키지 이름도 프로젝트 키워드에 포함.

코멘트는 다르게 취급함. 자연어로 쓰여져 있고 클래스나 메소드를 설명하기 위해 Javadoc tag나 html tag를 포함할 수 있다. 정확성을 높이기 위해 태그를 제거함.

3.5. Analyzing structured source file information

APPROACH

소스파일 전 처리 과정 후에 각 파일의 인덱스된 데이터의 split term으로 length 스코어를 계산[19].
버그리포트도 전 처리 과정 후에 BugRepoIndexer로 쿼리를 만든다.
SourceFileAnalyzer는 각 용어의 TF와 IDF를 분석하여 각 소스파일과 버그 리포트의 벡터 값을 계산한다.

TF(term frequency;해당 문서에 특정 용어가 나오는 횟수), DF(document frequency; 해당 용어가 나오는 문서 수), IDF(inverse DF)

f : source file
$\vec{f}$: weighted term frequency vector of f
b : bug report
$\vec{b}$: weighted term frequency vector of b
$\{x_1 \dots x_m\}$: exctracted terms of source file
$\{y_1 \dots y_n\}$: exctracted terms of bug report
m: 소스파일의 term 총 개수
n: 버그 리포트의 term 총 개수

$$\vec{f} = (tf_f(x_1)idf(x_1), tf_f(x_2)idf(x_2), \cdots, tf_f(x_m)idf(x_m)) \quad (1)$$

$$\vec{b} = (tf_b(y_1)idf(y_1), tf_b(y_2)idf(y_2), \cdots, tf_b(y_n)idf(y_n)) \quad (2)$$

TF

$$tf_d(t) = \log(f_{td}) + 1, \quad idf(t) = \log\left(\frac{d}{d_t}\right)$$

문서에 나오는 해당 용어의 수 / 전체 문서 수 / 해당 용어 나오는 문서 수

BugLocator[19]에서 식 가져옴

BugLocator에서는 클래식 VSM 사용했지만 본 연구에서는 소스파일의 structured information 활용하는 수정된 rVSM 사용해서 소스파일과 버그 리포트의 유사도 계산.
[20] 소스파일의 structured information 활용해서 파일 레벨의 버그 localization 정확도 높임.
BLUiR: 소스 파일을 4개의 document로 나눔(class, method, variable, comments). 버그파일을 2개의 쿼리로 나눔 (summary, description). 위의 식 3개로 벡터로 변환할 수 있다.

3.5. Analyzing structured source file information

APPROACH

$$cos(\vec{f_p}, \vec{b_p}) = \frac{\vec{f_p} \bullet \vec{b_p}}{|\vec{f_p}||\vec{b_p}|} \quad (4)$$

Cosine similarity

$$s(\vec{f}, \vec{b}) = \sum_{b_p \in b} \sum_{f_p \in f} (cos(\vec{f_p}, \vec{b_p}) \times w_{f_p}) \quad (5)$$

Cosine similarity를 그냥 합[20]하지 않고 weight 두고 합함. w_{fp} 는 소스파일 part의 weight value. Heuristically 코멘트 파트에는 50%하고 다른 부분에는 100%

$$len(f_c) = \frac{1}{1 + e^{-N(f_c)}} \quad (6)$$

소스파일의 length score. fc는 소스파일의 총 용어 수

$$N(X) = \frac{x - x_{min}}{x_{max} - x_{min}} \quad (7)$$

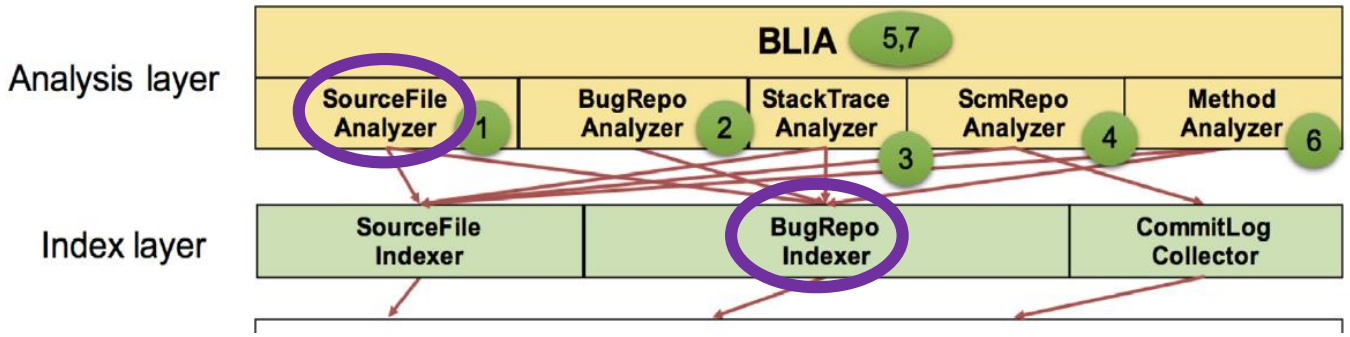
fc를 정규화 [19]에서 사용한 방법 Xmax: 가장 많은 용어 수. Xmin: 가장 적은 용어 수

$$StructVsmScore(\vec{f}, \vec{b}) = len(f_c) \times s(\vec{f}, \vec{b}) \quad (8)$$

본 연구에서 제안하는 수정된 rVSM

$\vec{f_p}$: 소스파일의 individual structured information part

$\vec{b_p}$: 버그 리포트의 each part



- BugRepoIndexer : 버그리포트도 전 처리 과정 후에 쿼리를 만든다.
- SourceFileAnalyzer: 각 용어의 TF와 IDF를 분석하여 각 소스파일과 버그 리포트의 벡터 값을 계산한다.

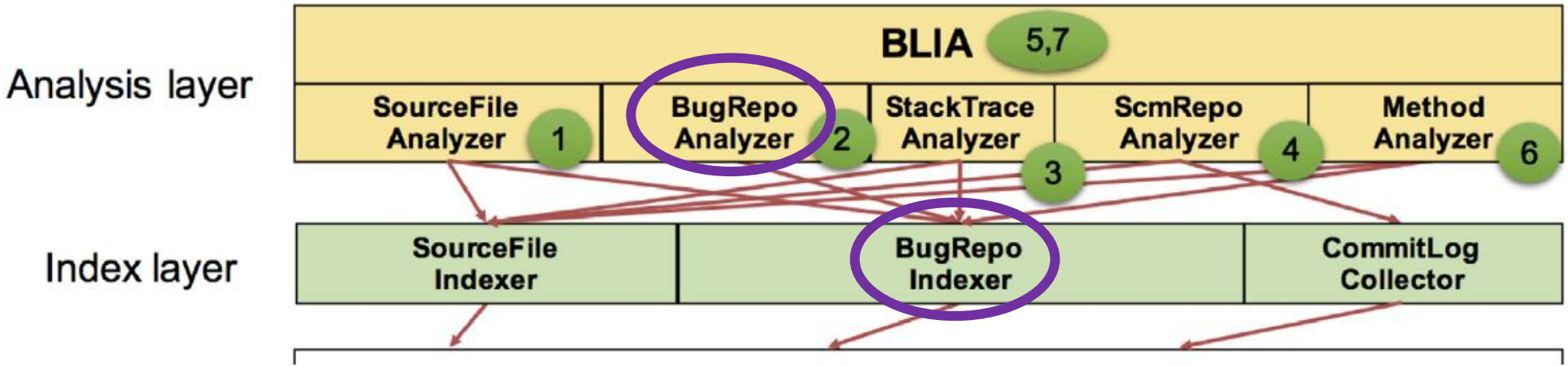
3.6. Analyzing similar bug reports

BugRepoIndexr: 수정된 버그 리포트 parse해서 데이터베이스에 인덱스 저장.

BugRepoAnalyzer: 새 버그리포트와 수정된 버그리포트의 유사도 계산. 새 버그리포트는 summary, description. 예전 버그 리포트는 summary, description, comment.

$$SimiBugScore(\vec{f}, \vec{b}) = \sum_{b' \in \{b' | b' \in B \wedge f \in b'_{fix}\}} \frac{cos(\vec{b}, \vec{b'})}{|b'_{fix}|} \quad (9)$$

b: new bug report
b': fixed bug report
[19]에서 가져온 식



3.7. Analyzing stack trace information in the bug report

APPROACH

Stack trace 없는 버그 리포트도 있지만 stack trace에 나타나는 파일에 버그 있을 확률 많다.

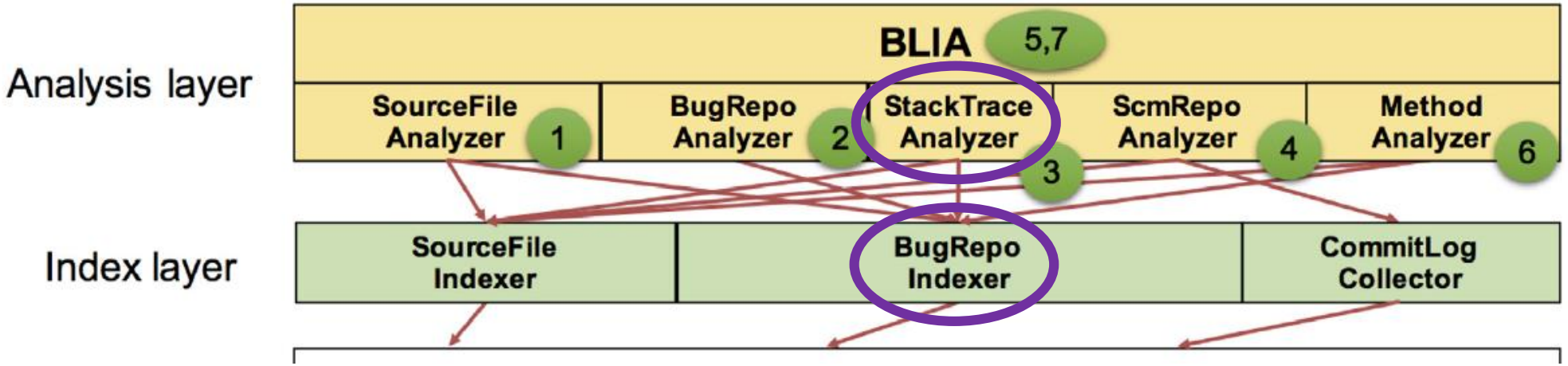
[23]에서 BRTracer 제안. [24] stack trace와 코드 element의 구조적 유사도 분석위한 Lobster 제안.

$$StraceScore(f,b) = \begin{cases} \frac{1}{rank} & f \in b_{strace} \wedge rank \leq 10 \\ 0.1 & f \in b_{strace} \wedge rank > 10 \\ 0.1 & f \in b_{strace.import} \\ 0 & otherwise \end{cases} \quad (10)$$

BRTracer; low computation cost, simple

BugRepoIndexer: regular expression으로 파일 이름과 stack trace를 추출한다.

StackTraceAnalyzer가 StraceScore 계산.



3.8. Analyzing code change history information

APPROACH

[21]에서 (11)(12) 식에 빠르고 심플하다는 걸 보여준다. [35]실험에 근거해서 알고리즘을 변경했다.

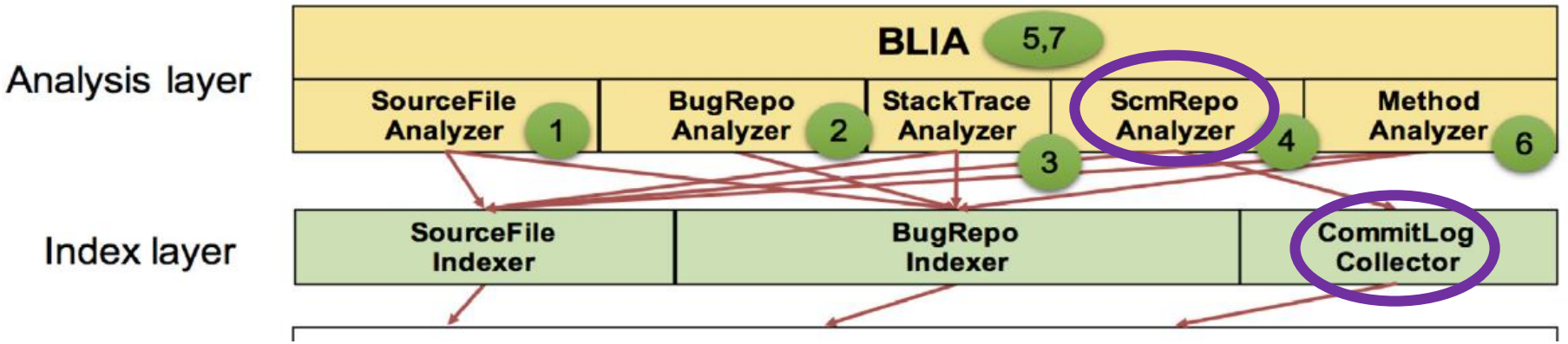
C: 지난 k일 동안 관련있는 commit 의 set
c: commit / d_c:commit c와 버그리포트간 경과된 일 수

$$CommScore(f,b,k) = \sum_{c \in C \wedge f \in c} \frac{1}{1+e^{12(1-(\frac{k-d_c}{k}))}} \quad (11)$$

$$CommScore(m,b,k) = \sum_{c \in C \wedge m \in c} \frac{1}{1+e^{12(1-(\frac{k-d_c}{k}))}} \quad (12)$$

- CommitLogCollector: SCM repository에서 commit log message 모은다. 커밋된 파일+수정된 메소드 수집 (각 커밋의 차이점 이용해서)
- ScmRepoAnalyzer: 버그리포트 제출 전의 관련된 commit log 분석

  모든 commit log 대신 (. *fix. *)|(. *bug. *)|(. *issue. *)|(. *fail. *)| (. *error. *). 와 매치 되는 log만
- 지난 k day 동안의 commit log를 분석
- 관련 commit log의 suspicious score 계산



3.9. Combining analyzed scores at the file level

APPROACH

$$c(f,b,\alpha) = (1-\alpha) \times N(StructVsmScore(f,b)) + \alpha \times N(SimiBugScore(f,b)) + StraceScore(f,b) \quad (13)$$

모든 버그 리포트에 있는게 아님

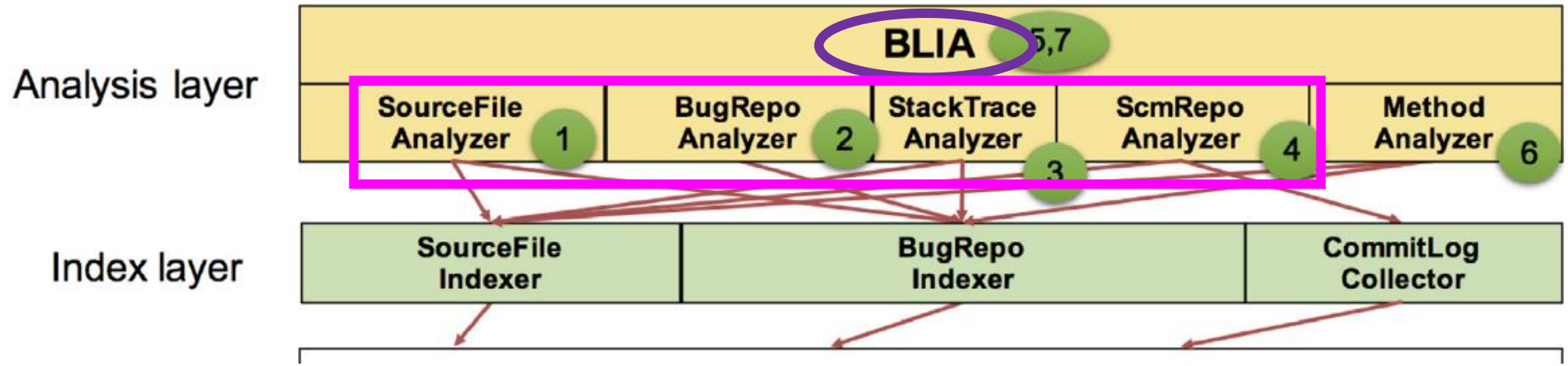
$$BliaFileScore(f,b,\alpha,\beta,k) = \begin{cases} (1-\beta) \times c(f,b,\alpha) + \beta \times CommScore(f,b,k), & c(f,b,\alpha) > 0 \\ 0, & otherwise \end{cases} \quad (14)$$

소스파일과 버그 리포트 관련 없는 경우

[19]와 [24], [21]에서 가져온 식

알파, 베타로 weight

BLIA 모듈에서 4개의 analyze score 합한다.



3.10. Analyzing method information

- 3.9.에서 Top10에 랭크된 파일만 본다고 가정, top10안의 메소드에 대해서만 계산
- 단어를 쿼리로 취급, 서치할 버그리포트를 document collection으로 본다.

$$\vec{m} = (tf_f(z_1)idf(z_1), tf_f(z_2)idf(z_2), \cdots, tf_f(z_q)idf(z_q)) \quad (15)$$

$$\vec{b} = (tf_b(y_1)idf(y_1), tf_b(y_2)idf(y_2) \cdots, tf_b(y_r)idf(y_r)) \quad (16)$$

$$MthVsmScore(\vec{m}, \vec{b}) = \cos(\vec{m}, \vec{b}) = \frac{\vec{m} \bullet \vec{b}}{|\vec{m}||\vec{b}|} \quad (17)$$

버그리포트와 메소드간의 유사도 점수

3.11. Combining analyzed scores at the method level

$$\begin{aligned} &BliaMethodScore(m, b, \gamma, k) \\ &= (1-\gamma) \times N(MthVsmScore(m, b)) \\ &+ \gamma \times CommScore(m, b, k) \end{aligned}$$

4

EVALUATION METRICS

4.1. Subject projects

Details of subject projects.

Project	Description	Period of registered bug (Used source version)	#Bugs	#Source Files
AspectJ	Aspect-oriented extension to Java	12/2002~ 07/2007 (org.aspectj-1_5_3_final)	284	5188
SWT	Widget toolkit for Java	04/2002~ 12/2005 (swt-3.659)	98	738
ZXing	Barcode image processing library	03/2010~ 09/2010 (Zxing-1.6)	20	391

본 실험에서 사용한 프로젝트들

- [19]BugLocator와 [23]BRTracer에서 사용한 데이터셋 적용. 온라인에서 가져옴. http://bug.inf.usi.ch/index.php

- 버그리포트 상태와 수정 날짜 유효성 확인
- Repository data 늘리기 위해 버그 리포트의 수정된commit 찾음
- 수정된 메소드를 'git diff' 커맨드 사용해서 추출 (3.4.에 언급)
- 세 개의 프로젝트에서 마지막으로 수정된 버그와 commit log의 수정된 소스파일 다운로드

- AspectJ 데이터셋 유효성 체크해서 두 개의 버그는 제외시킴. (section 6 에 상세 설명)

4.1. Evaluation metrics

1. Top N rank ; IR-based bug localization에 많이 평가됨
2. Mean average precision (MAP); popular한 IR 테크닉 평가 metric
3. Mean reciprocal rank (MRR); popular한 IR 테크닉 평가 metric

Top N rank: 적어도 하나의 버그가 있는 소스 파일이나 메소드가 발견되어 N 개의 검색 결과에서 순위가 매겨진 버그 리포트 수를 계산. 예를 들어 top N 결과 중에서 적어도 하나의 파일이나 메소드가 수정된게 있으면 성공. 초기 정확성을 강조

Mean Average Precision(MAP): 모든 랭크된 파일이나 메소드를 고려한다. 모든 쿼리에 대한 평균 정확도

$$AP = \sum_{i=1}^{M} \frac{P(i) \times pos(i)}{number\ of\ positive\ instances} \quad (19)$$

$$P(i) = \frac{\#buggy\ files}{i} \quad (20)$$

쿼리에 대한 평균 정확도

i : suspicious object의 랭크
M: 랭크된 object

P(i) : top i rank에서 buggy 파일이나 메소드의 정확도
pos(i) : i번째 랭크된 obect가 버그인지 아닌지 알려주는 2진 indicator

Mean Reciprocal Rank (MRR): 첫번째 버그 파일/메소드가 있는 것의 랭크의 역수

$$MRR = \frac{1}{|Q|} \sum_{|Q|}^{i=1} \frac{1}{rank_i}$$

- Top N Rank는 전반적인 퀄리티를 고려하지 않지만 MAP과 MRR은 전반적인 랭크 퀄리티를 본다. 결과값이 클수록 정확하다.
- 추가적으로 improvemen가 통계적으로 의미가 있는 정도인지 확인하기 위해 statistical 테스트 진행.

4.1. Evaluation metrics

Wilcoxon Rank-Sum test로 다음 가설에 대해 테스트

1. MAP 결과에 큰 차이가 없다.
2. MRR에 큰 차이가 없다.

- MAP 또는 MRR에서 큰 차이가 있는지 One-tailed test: 두 변인 중 한 변인의 가능성이 크다고 여김
- Bonferroni correction: 여러 접근법을 비교
- 각 가정에 대해 5가지 비교(BLIA v1.5 vs. BugLocator, BLUiR, BRTracer, AmaL- gam, and BLIA v1.0)
- Boferroni correction은 각 가설을 α / n의 유의 수준에서 테스트하여 많은 수의 비교를 보완한다. 여기서 α는 원하는 전반적인 알파 수준이고 n은 테스트 또는 비교의 수 → n이 5이고 원하는 α = 0 인 경우, Boferroni correction 테스트는 각 비교를 α=0.05/5, 즉 α=0.01로 두고 한다.
- Cliff's delta 사용해서 평균값의 차이의 크기를 비교했다. Cliff의 delta (또는 d)를 사용했으며, ordinal 데이터의 비모수 효과 크기 측정입니다. 효과 크기 | d | <0.33: 작다, | d | <0.474: 중간, 다른 값: 크다.
- Cliff's delta d를 효과 크기로 선택한 이유 : 변수에 적합하고 다른 레벨(작은, 중간, 큰)에 대한 정의가 있기 때문에 해석하기가 쉽다.

EXPERIMENTAL RESULTS

RQ1 : 파일 레벨에서 다른 localization tool 보다 성능이 더 좋은가?
RQ2: 파일 레벨에서 각 점수에 대해 가장 좋은 변수 값은?
RQ3: 파일 레벨에서 어떤 점수가 가장 영향이 큰가
RQ4: 메소드 레벨에서 BLIA가 다른 bug localization tool보다 성능이 좋은가?

5.1. Bug localization at the file level

EXPERIMENTAL RESULTS

Comparison between including comments and excluding comments of new bug reports at the file level. (The bold values in comparisons mean the highest values of each given metric.)

Project	Case	Top1 (%)	Top5 (%)	Top10 (%)	MAP	MRR
AspectJ (α=0.3, β=0.2, k=120)	Including new bug's comments	**41.5**	**71.1**	**80.6**	**0.39**	**0.55**
	Excluding new bug's comments	40.0	67.0	75.4	0.35	0.52
SWT (α=0.0, β=0.0, k=120)	Including new bug's comments	**67.3**	**86.7**	**89.8**	**0.65**	**0.75**
	Excluding new bug's comments	65.3	83.7	86.7	0.62	0.73
ZXing (α=0.2, β=0.0, k=120)	Including new bug's comments	**55.0**	**75.0**	**80.0**	**0.62**	**0.64**
	Excluding new bug's comments	50.0	65.0	70.0	0.54	0.58

코멘트 포함 여부를 비교 하면
코멘트를 포함 한 게 더 높은 점수
→ 다른 실험에서도 코멘트 추가함

Comparison with other approaches at the file level. (The bold values in comparisons mean the highest values of each given metric.)

Project	Approach	Top1 (%)	Top5 (%)	Top10 (%)	MAP	MRR
AspectJ	**BLIA v1.5** (α=0.3, β=0.2, k=120)	41.5	**71.1**	**80.6**	**0.39**	**0.55**
	BLIA v1.0	37.7	64.4	73.2	0.32	0.49
	BugLocator	30.8	51.1	59.4	0.22	0.41
	BLUiR	33.9	52.4	61.5	0.25	0.43
	BRTracer	39.5	60.5	68.9	0.29	0.49
	AmaLgam	**44.4**	65.4	73.1	0.33	0.54
SWT	**BLIA v1.5** (α=0.0, β=0.0, k=120)	67.3	**86.7**	**89.8**	**0.65**	**0.75**
	BLIA v1.0	**68.4**	82.7	89.8	0.64	0.75
	BugLocator	39.8	67.4	81.6	0.45	0.53
	BLUiR	56.1	76.5	87.8	0.58	0.66
	BRTracer	46.9	79.6	88.8	0.53	0.60
	AmaLgam	62.2	81.6	89.8	0.62	0.71
ZXing	**BLIA v1.5** (α=0.2, β=0.0, k=120)	**55.0**	**75.0**	**80.0**	**0.62**	**0.64**
	BLIA v1.0	50.0	60.0	80.0	0.51	0.57
	BugLocator	40.0	60.0	70.0	0.44	0.50
	BLUiR	40.0	60.0	70.0	0.39	0.49
	BRTracer	N/A	N/A	N/A	N/A	N/A
	AmaLgam	40.0	65.0	70.0	0.41	0.51

BugLocator[19]
BLUiR[20]
BRTracer[23]
AmaLgam[21]

알파, 베타, 케이 최적값은 5.2에 나와있음

CommScore의 베타와 케이는 커밋 description quality와 release cycle에 의해 결정된다.

IR-base Bug locater's accuracy:
버그 리포트와 커밋 코멘트의 퀄리티에 의해 결정됨 (한계)

5.1. Bug localization at the file level

Statistical analysis results of Wilcoxon Rank-Sum test (p-value) and Cliff's delta (d).

Data set	Compared approach	p-value	d	Effect size
MAP	vs. BugLocator	0.2	0.56	large
	vs. BLUiR	0.13	0.67	large
	vs. BRTracer	0.2	0.25	small
	vs. AmaLgam	0.25	0.44	medium
	vs. BLIA v1.0	0.35	0.33	medium
MRR	vs. BugLocator	0.05	0.67	large
	vs. BLUiR	0.2	0.44	medium
	vs. BRTracer	0.2	0.25	small
	vs. AmaLgam	0.2	0.44	medium
	vs. BLIA v1.0	0.35	0.33	medium

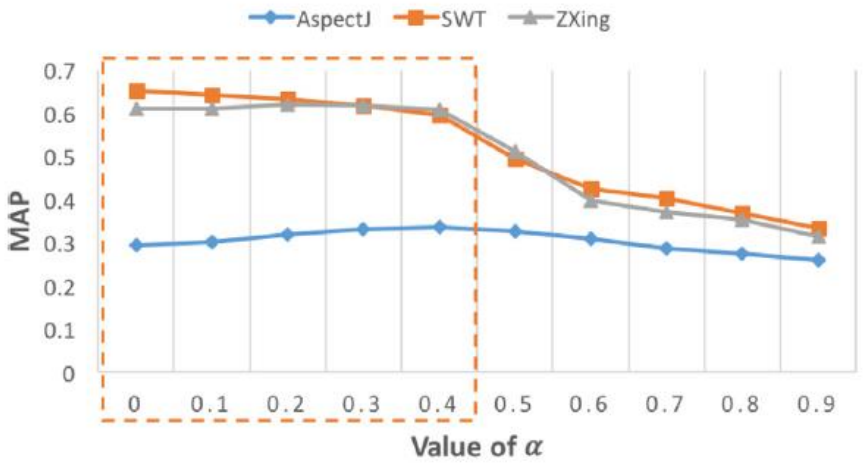
Bonferroni correction 후에 α=0.01

p-value: null hypothesis 거절할 수 없다.

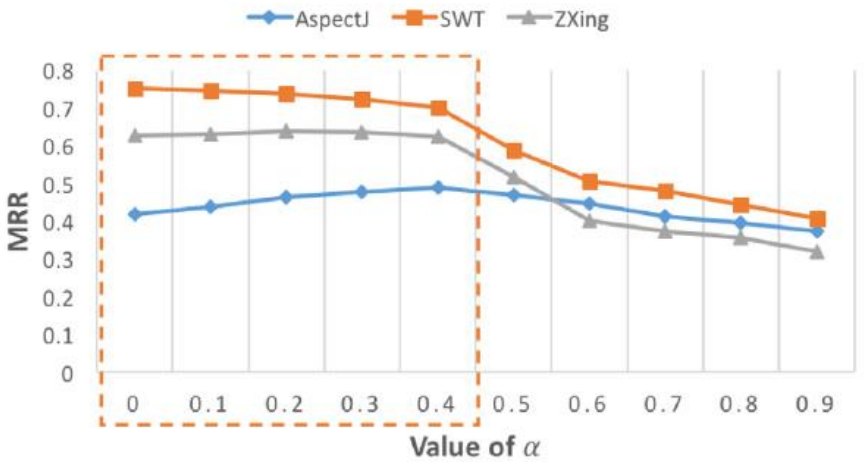
d값은 BRTracer를 제외하고는 중간이거나 큰 사이즈

5.2. Optimized combination of analyzable inputs

α의 값, $(\beta = 0, k = 120)$ 0.0~0.4일 때 stable



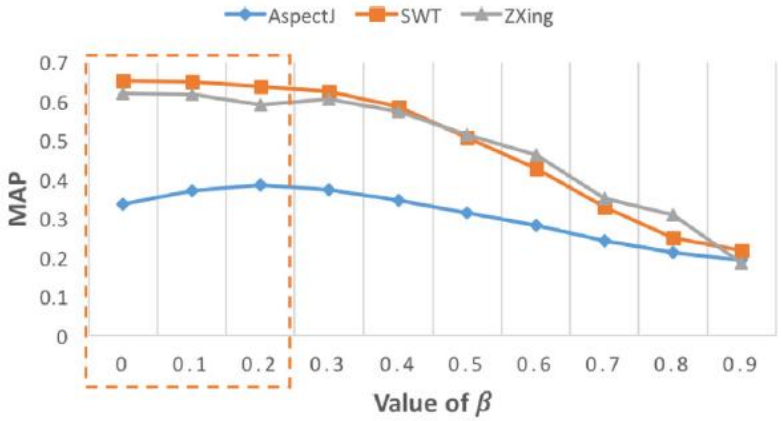
(a) MAP



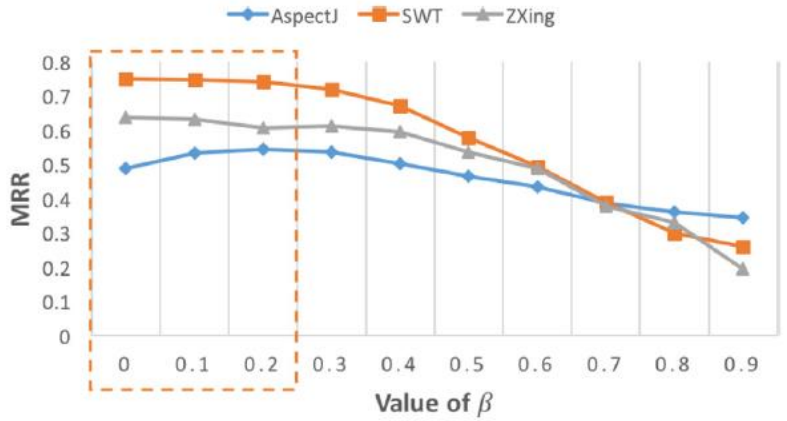
(b) MRR

Fig. 6. Impact of the value of α on BLIA in terms of MAP and MRR.

β값 0.0~0.2



(a) MAP

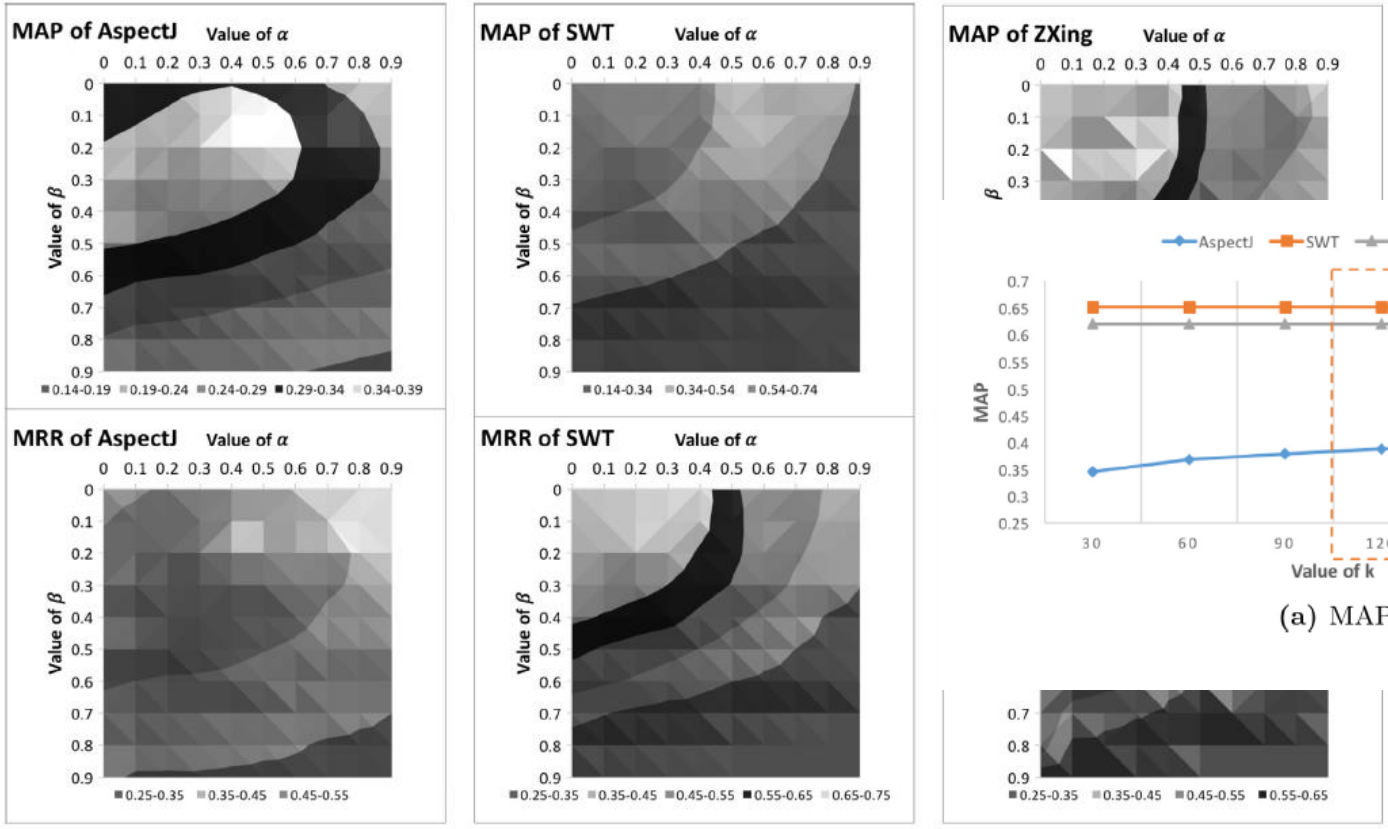


(b) MRR

Fig. 7. Impact of the value of β on BLIA in terms of MAP and MRR.

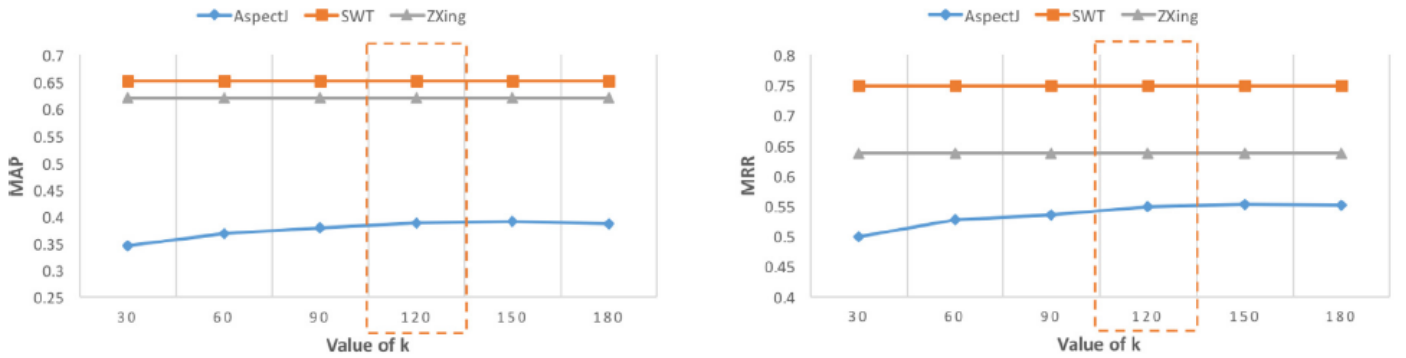
5.2. Optimized combination of analyzable inputs

α(0.1~0.5)와 β(0.1~0.3)의 관계
β 값을 변화시키는게 MAP, MRR 점수에 영향이 더 크다.
버그 리포트 수가 많을수록, 프로젝트 사이즈가 클수록 highest score 지역이 좁아진다.
k=120일 때 점수가 높다.



(a) AspectJ (b) SWT (c) ZXing

Fig. 8. Relation analysis between α and β on BLIA.



(a) MAP (b) MRR

Fig. 9. Impact of the value of k on BLIA in terms of MAP and MRR.

5.3. Influence rate of each type of analyzed data

어떤 변수 값이 좋은지 알기 위해서 최적화된 알파, 베타값에서 조사

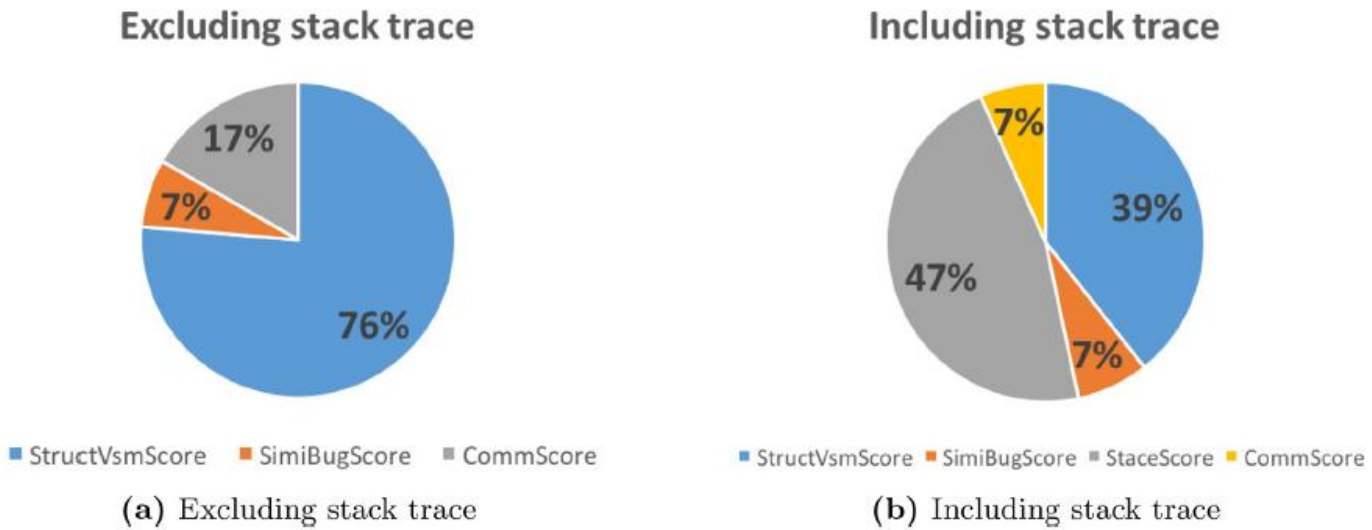


Fig. 10. Effects of each analyzed score.

- 통합 점수인 BliaFileScore에서 influenced rate.
- 평균적인 influence rate

- StraceScore=StructVsmScore+SimiBugScore
- source file is boosted only when a bug report contains stack traces [23]
- stack trace 포함 여부에 따라 influence rate 분석함.
- stack trace가 포함되었을 때 큰 부분을 차지하기 때문에 정확성 높이기에 필요한 정보이다.
- 버그리포트의 summar와 descriptio은 모듈 이름(class, method, variable) 사용해서 자세히 적어야함
- 버그리포트-소스파일간의 유사도 계산 위해 모듈 이름은 human-readable 단어여야 한다.

5.4. Bug localization at the method level

EXPERIMENTAL RESULTS

Table 8
Result of BLIA v1.5 at the method level.

Project	Approach	Top1 (%)	Top5 (%)	Top10 (%)	MAP	MRR
AspectJ	**BLIA v1.5** (α=0.3, β=0.2, γ=0.4, k=120)	6.0	14.8	21.5	0.08	0.11
SWT	**BLIA v1.5** (α=0.0, β=0.0, γ=0.5, k=120)	10.2	20.4	32.7	0.10	0.17
ZXing	**BLIA v1.5** (α=0.2, β=0.0, γ=0.3, k=120)	15.0	25.0	30.0	0.20	0.20

[38]과 비교하려 했으나 공개적인 데이터셋 없어서, spectrum-based bug localization 프로그램 실행이 필요해서 정확한 비교가 어렵다 → 통계적으로 분석하기 때문에 computation cost 줄여준다.

모든 버그 리포트가 코드 라인 고치는거로만 수정되지는 않음
ex) javadoc문서가 수정되었고 관련된 member variable들이 수정된다.

감마 값을 0~0.9 바꿔봄
highest value 평가하기 힘듬

하지만 메소드 레벨의 접근 가능성 발견 → 향상 위해 연구 필요

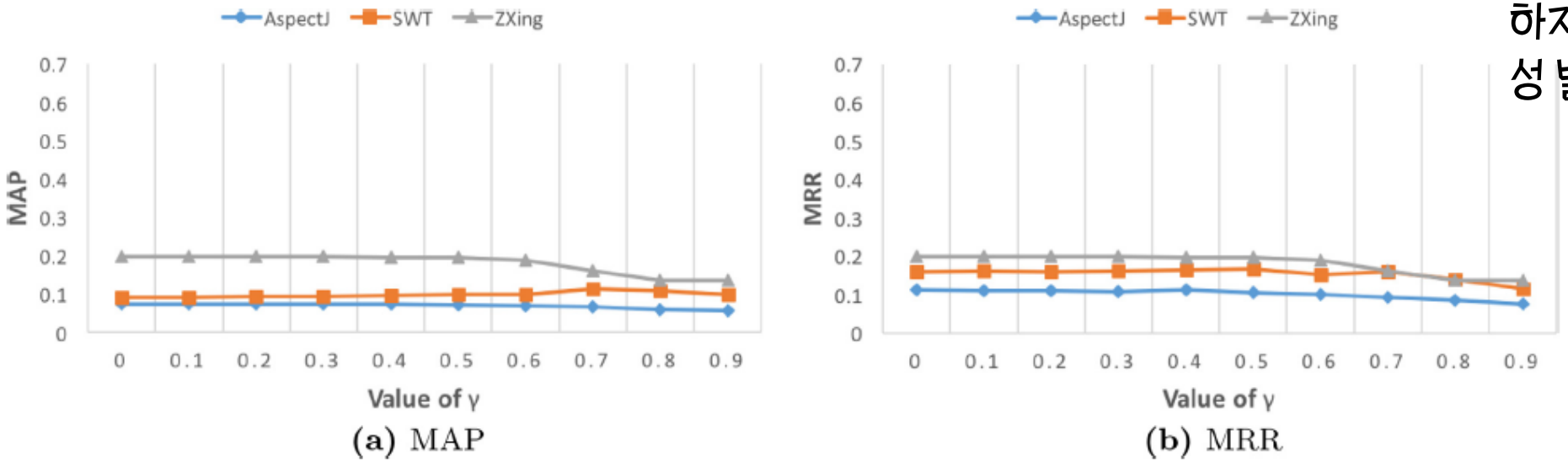


Fig. 11. Impact of the value of γ on BLIA in terms of MAP and MRR.

THREATS TO VALIDITY

한계, generalizability의 유효성

- internal validity
- external validity
- construct validity
- reliability

6.1. Internal validity

[19][23]와 동일한 데이터셋 사용함 (3개의 프로젝트).
데이터에 오류가 있을 수도 있기 때문에 버그 리포트를 검토함 → status가 “REOPENED”, “NEW”인 거 두개 발견
fixed date가 bug repository와 매치되지 않는 버그들이 많아서 수정함

6.2. External validity

선택한 프로젝트; 앞선 연구에서 사용된것, 유명하지만 generalizable?
improvement 평가를 위해 통계적 테스트를 함(Wilcoxon Rank-Sum test) → 의미가 있는 정도의 향상인가?
(미래) 다른 종류의 프로젝트로 확장, improvement가 significant한지 statistical test.

6.3. Construct validity

evaluation metric: Top N rank, MAP, MRR
[19-21,23] 에서 사용된 방법이고 잘 알려진 IR metric이다. → construct validity 강하다.

6.4. Reliability

알파, 베타, 감마의 최적값을 찾았다. 하지만 parameter의 종류에 따라 다른 결과를 얻는다. BLIA 뿐만 아니라 다른 툴들도 그렇다. best accuracy 비슷한 방법으로 찾지만 각 변수 사용법 다름.

ex) Buglocator: 알파로 rVSMScor과 SimiScore 묶는다. (BLIA: StructVsmScored과 SimiBugScore)

BLUiR: 두 변수를 쓰지만 묶을 때 쓰는게 아니다

BRTracer: 알파를 rVSMseg와 SimiScore과 묶는다 (BLIA: StructVsmScore, SimiScore

(미래) parameter를 generalize 하기 위해서 머신 러닝 / generic algorithm 사용

RELATED WORK

7.1. Change impact analysis

CIA (software change impact analysis

변경된 사항의 효과나 수정되야하는 필요성 계산.[5]
여러 종류가 있는데 static analysis와 dynamic analysis로 크게 분류될 수 있다 [7-11]
대표적인 [6]은 Java를 위한 두 버전의 어플리케이션을 분석해서 차이점을 알아낸다

FaultTracer[12]

change impact & regression fault analysis tool for evolving programs
spectrum-based ranking 테크닉을 CIA에 적용

7.2. Spectrum-based bug localization

spectrum-based fault localization.
Runtime execution 필요. 프로그램의 spectra information을 분석한다.

[39] spectrum-based fault localization + specification based analysis combine 하는 것 제안
granularity(세부 단위 퀄리티)
[40] frequent item set mining, method call의 패턴, variant 제안
[41] learning-to-rank 머신 러닝 접근법. 버그 있는 메소드의 failure,
[42] algebraically, probabilistically analyze metric of spectrum-based fault localization, 가장 effectiv
한 방법 추천.

7.3. IR-based bug localization

버그리포트는 bug localization의 시작점이자 핵심 키다 .
많은 연구들이 있었다. [19-24, 26, 38, 43-45]
정보들을 결합해서 suspicious ranking의 정확도 향상
ex) 버그 리포트의 stack trace 분석, 소스파일의 structured file, 소스파일을 나누는 것, program dependency graph
[21,31,32,46-48] 버그/이슈 management syste의 데이터 마이닝과 소스코드 히스토리의 분석
[49] 사용되고 있는 IR-based bug localization
IR-based localizer 성능 비교를 위한 experimental platform: BOAT [50]
오픈소스 프로젝트의 온라인dataset [51,52]

spectrum-based 접근과 IR-based 접근을 결합하는 방법 제안됨[38] → granularity 향상
[53] 딥러닝 중 DNN(deep neural network)을 IR테크닉인 rVSM과 결합
rVSM은 버그리포트와 소스코드의 textual similarity 모은다.
DNN으로 버그리포트의 용어들을 학습한 다음에 소스코드 파일에 연관짓는다.
[54] lexical gap 을 잇는 방법 제안 : natrul language statemen와 code snippet을 project해서 IR-base localization 성능 향상. Word embedded technique: 관련 sw 다큐먼트(API doc, tutorial, reference document)를 학습해서 문서들간의 semantic 유사도
[55] IR framework – special code proximity & term ordering relationship (accuracy 향상)

8

CONCLUSION

[56-58]에서 좀 더 나은 버그 리포트 작성법 제안했지만 자연어로 적기 때문에 표준화하기 힘들다.
최근 나온 방법이 IR과 데이터 마이닝을 사용하는 것

BLIA는 통계적으로 통합된 IR-based bug localization 분석이다.
버그리포트에서 text, stack trace, comments를 분석.
소스 파일의 structured information, 소스 코일 변화 히스토리 분석
모든 분석된 스코어를 합한다. 파일의 랭킹을 보여준다.
랭크된 파일 중에서 메소드와 버그리포트, 소스파일, SCM 의 유사도 계산해서 합한다 → 랭킹

메소드 레벨에서 정확성을 높이는데 의의
여러 인포메이션 조합하면서 influence rate 조사.
bug repository 늘림(코멘트, fixed method, fixed commit)

(미래) 1. [59] 리뷰어 추천, [60]비슷한 버그 추천연구는 버그 리포트를 해결한다. BLIA를 확장하고 generalize
2. Buglocalizer[49]는 버그/이슈 매니지 시스템이 추가되었다. Buzilla. 의심가는 랭킹 파일 제공. BLIA를 bug/issue management system에 추가.

References

[1] K.C. Youm , J. Ahn , J. Kim , E. Lee , Bug localization based on code change his-tories and bug reports, in: 2015 Asia-Pacific Software Engineering Conference (APSEC), IEEE, 2015, pp. 190–197 . [2] R.D. Banker , S.M. Datar , C.F. Kemerer , D. Zweig , Software complexity and main-tenance costs, Commun ACM 36 (11) (1993) 81–95 . [3] M.J.C. Sousa , H.M. Moreira , A survey on the software maintenance process, in: Software Maintenance, 1998. Proceedings., International Conference on, IEEE, 1998, pp. 265–274 . [4] M.K. Davidsen , J. Krogstie , A longitudinal study of development and mainte-nance, Inf Softw Technol 52 (7) (2010) 707–719 . [5] B. Li , X. Sun , H. Leung , S. Zhang , A survey of code-based change impact anal-ysis techniques, Software Testing, Verification and Reliability 23 (8) (2013) 613–646 . [6] X. Ren , F. Shah , F. Tip , B.G. Ryder , O. Chesley , Chianti: a tool for change im-pact analysis of java programs, in: ACM Sigplan Notices, vol. 39, ACM, 2004, pp. 432–448 . [7] X. Ren , O.C. Chesley , B.G. Ryder , Identifying failure causes in java programs: an application of change impact analysis, Software Eng., IEEE Trans. 32 (9) (2006) 718–732 . [8] J. Wloka , B.G. Ryder , F. Tip , Junitmx-a change-aware unit testing tool, in: Pro-ceedings of the 31st International Conference on Software Engineering, IEEE Computer Society, 2009, pp. 567–570 . [9] M. Acharya , B. Robinson , Practical change impact analysis based on static pro-gram slicing for industrial software systems, in: Proceedings of the 33rd inter-national conference on software engineering, ACM, 2011, pp. 746–755 . [10] N. Rungta , S. Person , J. Branchaud , A change impact analysis to characterize evolving program behaviors, in: Software Maintenance (ICSM), 2012 28th IEEE International Conference on, IEEE, 2012, pp. 109–118 . [11] B. Li , X. Sun , J. Keung , Fca–cia: an approach of using fca to support cross-level change impact analysis for object oriented java programs, Inf Softw Technol 55 (8) (2013) 1437–1449 . [12] L. Zhang , M. Kim , S. Khurshid , Faulttracer: a spectrum-based approach to lo-calizing failure-inducing program edits, Journal of Software: Evolution and Pro-cess 25 (12) (2013) 1357–1383 . [13] W.E. Wong , V. Debroy , A Survey of Software Fault Localization, Department of Computer Science, University of Texas at Dallas, 2009 . Tech. Rep. UTDCS-45 9. [14] L. Naish , H.J. Lee , K. Ramamohanarao , A model for spectra-based software di-agnosis, ACM Trans. Software Eng. Methodol. (TOSEM) 20 (3) (2011) 11 . [15] J.A. Jones , M.J. Harrold , Empirical evaluation of the tarantula automatic fault- -localization technique, in: Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering, ACM, 2005, pp. 273–282 . [16] W.E. Wong , V. Debroy , R. Gao , Y. Li , The dstar method for effective software fault localization, Reliability, IEEE Transactions on 63 (1) (2014) 290–308 . [17] C.D. Manning , P. Raghavan , H. Schütze , et al. , Introduction to Information Re-trieval, vol.1, Cambridge University Press, Cambridge, 2008 . [18] D. Binkley , D. Lawrie , Information retrieval applications in software main-tenance and evolution, in: Encyclopedia of Software Engineering, 2010, pp. 454–463 . [19] J. Zhou , H. Zhang , D. Lo , Where should the bugs be fixed? more accurate infor-mation retrieval-based bug localization based on bug reports, in: Software En-gineering (ICSE), 2012 34th International Conference on, IEEE, 2012, pp. 14–24 . [20] R.K. Saha , M. Lease , S. Khurshid , D.E. Perry , Improving bug localization using structured information retrieval, in: Automated Software Engineering (ASE), 2013 IEEE/ACM 28th International Conference on, IEEE, 2013, pp. 345–355 . [21] S. Wang , D. Lo , Version history, similar report, and structure: Putting them to-gether for improved bug localization, in: Proceedings of the 22nd International Conference on Program Comprehension, ACM, 2014, pp. 53–63 . [22] S. Davies , M. Roper , Bug localisation through diverse sources of information, in: Software Reliability Engineering Workshops (ISSREW), 2013 IEEE Interna-tional Symposium on, IEEE, 2013, pp. 126–131 . [23] C.-P. Wong , Y. Xiong , H. Zhang , D. Hao , L. Zhang , H. Mei , Boosting bug-re-port-oriented fault localization with segmentation and stack-trace analysis, in: 2014 IEEE International Conference on Software Maintenance and Evolution (ICSME), IEEE, 2014, pp. 181–190 . [24] L. Moreno , J.J. Treadway , A. Marcus , W. Shen , On the use of stack traces to im-prove text retrieval-based bug localization, in: 2014 IEEE International Confer-ence on Software Maintenance and Evolution (ICSME), IEEE, 2014, pp. 151–160 . [25] A. Schroter , N. Bettenburg , R. Premraj , Do stack traces help developers fix bugs? in: Mining Software Repositories (MSR), 2010 7th IEEE Working Con-ference on, IEEE, 2010, pp. 118–121 . [26] S. Davies , M. Roper , M. Wood , Using bug report similarity to enhance bug lo-calisation, in: Reverse Engineering (WCRE), 2012 19th Working Conference on, IEEE, 2012, pp. 125–134 . [27] G. Antoniol , G. Canfora , G. Casazza , A. De Lucia , E. Merlo , Recovering traceabil-ity links between code and documentation, Software Engineering, IEEE Trans-actions on 28 (10) (2002) 970–983 . [28] C.D. Manning , H. Schütze , Foundations of Statistical Natural Language Process-ing, vol.999, MIT Press, 1999 . [29] D.M. Blei , A.Y. Ng , M.I. Jordan , Latent dirichlet allocation, J. Mach. Learn. Res. 3 (2003) 993–1022 . [30] S. Deerwester, Improving Information Retrieval with Latent Semantic Indexing (1988). [31] B. Sisman , A.C. Kak , Incorporating version histories in information retrieval based bug localization, in: Proceedings of the 9th IEEE Working Conference on Mining Software Repositories, IEEE Press, 2012, pp. 50–59 . [32] C. Tantithamthavorn , R. Teekavanich , A. Ihara , K.-i. Matsumoto , Mining a change history to quickly identify bug locations: A case study of the eclipse project, in: Software Reliability Engineering Workshops (ISSREW), 2013 IEEE International Symposium on, IEEE, 2013, pp. 108–113 . [33] S. Kim , T. Zimmermann , E.J. Whitehead Jr , A. Zeller , Predicting faults from cached history, in: Proceedings of the 29th international conference on Soft-ware Engineering, IEEE Computer Society, 2007, pp. 4 89–4 98 . [34] F. Rahman , D. Posnett , A. Hindle , E. Barr , P. Devanbu , Bugcache for inspections: hit or miss? in: Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering, ACM, 2011, pp. 322–331 . [35] C. Lewis, R. Ou, Bug prediction at google, 2011. URL http://google-engtools. blogspot.sg/2011/12/bug- prediction- at- google.html [36] W. Conover , Practical Nonparametric Statistics, Wiley Series in Probability and Statistics, third ed., Wiley, 1999 . [37] R.J. Grissom , J.J. Kim , Effect Sizes for Research: A Broad Practical Approach, Lawrence Erlbaum Associates, 2005 .

[38] T.-D.B. Le , R.J. Oentaryo , D. Lo , Information retrieval and spectrum based bug localization: better together, in: Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, ACM, 2015, pp. 579–590 . [39] D. Gopinath , R.N. Zaeem , S. Khurshid , Improving the effectiveness of spec-tra-based fault localization using specifications, in: Automated Software En-gineering (ASE), 2012 Proceedings of the 27th IEEE/ACM International Confer-ence on, IEEE, 2012, pp. 40–49 . [40] G. Laghari , A. Murgia , S. Demeyer , Fine-tuning spectrum based fault localisa-tion with frequent method item sets, in: Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering, in: ASE 2016, ACM, 2016, pp. 274–285 . [41] T.-D.B. Le , D. Lo , C. Le Goues , L. Grunske , A learning-to-rank based fault lo-calization approach using likely invariants, in: Proceedings of the 25th Inter-national Symposium on Software Testing and Analysis, in: ISSTA 2016, ACM, 2016, pp. 177–188 .[42] S.-F. Sun , A. Podgurski , Properties of effective metrics for coverage-based sta-tistical fault localization, in: 2016 IEEE International Conference on Software Testing, Verification and Validation (ICST), IEEE, 2016, pp. 124–134 . [43] R.K. Saha , J. Lawall , S. Khurshid , D.E. Perry , On the effectiveness of informa-tion retrieval based bug localization for c programs, in: 2014 IEEE Interna-tional Conference on Software Maintenance and Evolution (ICSME), IEEE, 2014, pp. 161–170 . [44] A.T. Nguyen , T.T. Nguyen , J. Al-Kofahi , H.V. Nguyen , T.N. Nguyen , A topic-based approach for narrowing the search space of buggy files from a bug report, in: Automated Software Engineering (ASE), 2011 26th IEEE/ACM International Conference on, IEEE, 2011, pp. 263–272 . [45] Q. Wang , C. Parnin , A. Orso , Evaluating the usefulness of ir-based fault local-ization techniques, in: Proceedings of the 2015 International Symposium on Software Testing and Analysis, ACM, 2015, pp. 1–11 .
[46] S. Wang , D. Lo , J. Lawall , Compositional vector space models for improved bug localization, in: 2014 IEEE International Conference on Software Maintenance and Evolution (ICSME), IEEE, 2014, pp. 171–180 . [47] S. Wang , F. Khomh , Y. Zou , Improving bug localization using correlations in crash reports, in: Mining Software Repositories (MSR), 2013 10th IEEE Working Conference on, IEEE, 2013, pp. 247–256 . [48] M. Gethers , B. Dit , H. Kagdi , D. Poshyvanyk , Integrated impact analysis for managing software changes, in: 2012 34th International Conference on Soft-ware Engineering (ICSE), IEEE, 2012, pp. 430–440 . [49] F. Thung , T.-D.B. Le , P.S. Kochhar , D. Lo , Buglocalizer: integrated tool support for bug localization, in: Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, ACM, 2014, pp. 767–770 . [50] X. Wang , D. Lo , X. Xia , X. Wang , P.S. Kochhar , Y. Tian , X. Yang , S. Li , J. Sun , B. Zhou , Boat: an experimental platform for researchers to comparatively and reproducibly evaluate bug localization techniques, in: Companion Proceedings of the 36th International Conference on Software Engineering, ACM, 2014, pp. 572–575 . [51] S. Rao , A. Kak , Morebugs: a new dataset for benchmarking algorithms for in-formation retrieval from software repositories (trece-13-07), Purdue University, School of Electrical and Computer Engineering, Tech. Rep 4 (2013) . [52] V. Dallmeier , T. Zimmermann , Extraction of bug localization benchmarks from history, in: Proceedings of the 22nd IEEE/ACM International Conference on Au-tomated Software Engineering, ACM, 2007, pp. 433–436 . [53] A .N. Lam , A .T. Nguyen , H.A . Nguyen , T.N. Nguyen , Combining deep learning with information retrieval to localize buggy files for bug reports (n), in: Auto-mated Software Engineering (ASE), 2015 30th IEEE/ACM International Confer-ence on, IEEE, 2015, pp. 476–481 . [54] X. Ye , H. Shen , X. Ma , R. Bunescu , C. Liu , From word embeddings to document similarities for improved information retrieval in software engineering, in: Pro-ceedings of the 38th International Conference on Software Engineering, ACM, 2016, pp. 404–415 . [55] B. Sisman , S.A. Akbar , A.C. Kak , Exploiting spatial code proximity and order for improved source code retrieval for bug localization, Journal of Software: Evolution and Process (2016) . [56] N. Bettenburg , S. Just , A. Schröter , C. Weiß, R. Premraj , T. Zimmermann , Quality of bug reports in eclipse, in: Proceedings of the 2007 OOPSLA Workshop on Eclipse Technology EXchange, ACM, 2007, pp. 21–25 . [57] P. Hooimeijer , W. Weimer , Modeling bug report quality, in: Proceedings of the 22nd IEEE/ACM International Conference on Automated Software Engineering, ACM, 2007, pp. 34–43 . [58] T. Zimmermann , R. Premraj , N. Bettenburg , S. Just , A. Schröter , C. Weiss , What makes a good bug report? Software Engineering, IEEE Transactions on 36 (5) (2010) 618–643 . [59] Y. Yu , H. Wang , G. Yin , C.X. Ling , Reviewer recommender of pull-requests in GitHub, in: 2014 IEEE International Conference on Software Maintenance and Evolution (ICSME), IEEE, 2014, pp. 609–612 . [60] H. Rocha , G. De Oliveira , H. Marques-Neto , M.T. Valente , Nextbug: a Bugzilla extension for recommending similar bugs, J. Softw. Eng. Res. Dev. 3 (1) (2015) 1–14 .