한국컴퓨터정보학회 하계학술대회 논문집 제23권 제2호 (2015. 7)

추상 구문 트리 기반의 개발자별 소스 코드 지분 분석

이용현O, 김기섭*, 정우성*
O*충북대학교 컴퓨터공학과
e-mail:{gd0live, nadanear, wsjung}@cbnu.ac.kr*

Analyzing Developer's Share of Code Based on Abstract Syntax Tree

Yong-hyeon LeeO, KiSub Kim*, Woosung Jung*
O*Dept. of Computer Engineering, Chungbuk National University

● Abstract ●

소프트웨어 프로젝트에서 발생하는 버그의 수가 증가함에 따라 이를 수정할 적합한 개발자를 효과적으로 찾기 위한 연구들이 진행되었다. 하지만, 대부분의 연구들이 텍스트 기반의 라인 변경을 통해 얻은 정보만을 활용하기 때문에 정확도가 떨어지며, 파일, 클래스, 함수와 같은 다양한 계층 단위에 대한 처리가 어렵다. 본 논문은 개발자의 코드 변경 정보를 추적함으로써 기여도를 보다 정확하게 계산하고, 다양한 코드 수준에서의 분석을 효과적으로 지원하는 AST기반의 지분 계산 접근법을 제안한다.

키워드: 추상 구문 트리(Abstract Syntax Tree), 코드 지분(Share of Code), 코드 소유권(Code Ownership)

I. 서론

소프트웨어의 규모가 증가하면서 개발자 협업 지원을 위해 버전 관리시스템(Version Control System) Git[1], SVN[2] 등이 고안되었다. 이로써 개발자는 개발 산출물의 변경이력을 추적하고, 동시 수정 작업으로 인한 충돌을 방지할 수 있게 되었다[1]. 하지만, 코드공동소유(Collective Code Ownership)에 기반하고 있는 버전관리시스템을 사용하더라도, 프로젝트 관리자는 유지보수 과정에서 코드의 버그를 개발자에게 할당해야 하는 과제가 있다. 버그추적시스템(Bug Tracking System)의 경우는 기존 버그 보고서의 특징 정보를 추출, 분석하여 신규 버그를 담당할 개발자를 효과적으로 검색하는 연구가 진행된 반면, 소스 기반 시스템의 경우는 관련 연구가 미흡하다[2]. 버그 할당에 사용할 개발자 전문성 정보 검출을 위한 접근법은 기술적 스킬만을 고려하거나[3], 경험 정보를 추가하여 이를 확장하는 방법이 있다[4]. 하지만, 대부분의 연구가 자연어 처리를 통한 텍스트 기반이기 때문에 소스 코드를 활용하지 못해 정확도가 낮고 파일, 클래스, 함수 등 다양한 수준에서의 정밀한 분석이 불가능하다. 따라서 본 연구에서는 이러한 문제를 해소하기 위해 소스코드의 추상 구문 트리 차이를 분석하고 기여도에 따른 개발자별 코드 지분(Share of Code)을 효과적으로 계산하는 방법을 제안한다.

II. 개발자의 코드 지분

본 연구에서는 특정 코드에 대한 개발자의 기여도인 지분 계산을 위해 모든 리비전에서 추가, 변경, 삭제된 토큰을 추적하여 S+AST(Share+AST)를 구축하는 방법을 제안하였다. S+AST는 이전 리비전의 S+AST 및 word-diff 정보를 이용하여 수식(1)을 통해 계산한 지분을 재귀적으로 추가하여 소스코드 구문 분석을 통해 확보한 AST를 확장한 것이다.

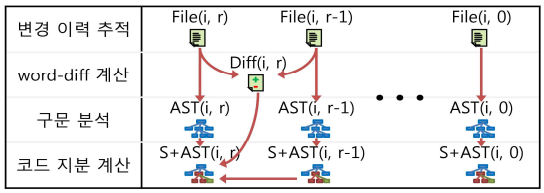
개발자 d, AST의 노드 n에 대해서

1) Git : http://www.git-scm.com/
2) SVN : http://subversion.tigris.org/

$$share_{(d,n)} = \begin{cases} 0 \text{ , } n\text{이 말단 노드이고} \\ \quad d\text{에 의해 생성되지 않았을 때} \\ 1 \text{ , } n\text{이 말단 노드이고} \\ \quad d\text{에 의해 생성되었을 때} \\ \sum_{i=1}^{childcount(n)} share_{(d,child(n,i))} \\ \quad \text{, } n\text{이 말단 노드가 아닐 때} \end{cases} \quad (1)$$

그림 1은 이러한 개발자 지분 계산 과정을 재귀적으로 보여준다. 주어진 그림에서 File(i,r)은 리비전 r에 속하는 전체 파일 중 i로 구분되는 특정 파일을 뜻하며, AST(i,r)은 File(i,r)의 AST를 의미하고, Diff(i,r)은 File(i,r)과 File(i,r-1)과의 코드 차이를 나타낸다. 그러므로, File(i,r)에 대한 지분 계산을 위해 필요한 S+AST(i,r)은 Diff(i,r)로부터 해당 개발자의 변경 기여도를 확인한 후 S+AST(i,r-1)의 지분을 이용해 다음의 3단계의 과정을 통해 계산한다.

1) S+AST(i, r-1)에서 삭제된 노드의 지분 제거
2) 남아 있는 지분을 S+AST(i, r)에 복사
3) S+AST(i, r)에서 추가된 노드의 지분 계산



※ i는 화일 식별자, r은 리비전을 의미함.

그림 2. File(i,r)의 지분 계산을 위한 S+AST(i,r) 구축 과정

III. S+AST의 구축

S+AST를 효과적으로 표현하기 위해서 그림 2와 같이 JSON을 이용하였다. "type"은 노드의 타입을 의미하며, "shares"는 개발자 지분을 나타낸다.

```
{ "type" : 노드의 타입명,          "start" : 시작 바이트,
  "length": 바이트 길이,           "code" : 소스코드,
  "shares": {  개발자의 이름 : 개발자의 지분, ... } }
```

그림 28. JSON 형식의 S+AST 구조

그림 3은 두 명의 개발자의 의해 생성, 변경된 Java 소스 코드 일부에 대해서 JSON 형식의 S+AST 구축 예시를 보여준다. 이는 일부 소스 코드(int MAX = 10;)가 Abraham에 의해 생성되고, Britney에 의해 초기값만 100으로 변경되었을 때 구축된 S+AST의 내용이다.

```
{"type":"FieldDeclaration", ..., "code":"int MAX = 100;\n",
"shares": {"Abraham":2, "Britney":1} }
{"type":"PrimitiveType", ..., "code":"int",
 "shares": {"Abraham":1 } }
{"type":"VariableDeclarationFragment", ...,
 "code":"MAX = 100","shares": {"Abraham":1, "Britney":1} }
{"type":"SimpleName", ..., "code":"MAX",
 "shares": {"Abraham":1 } }
{"type":"NumberLiteral", ..., "code":"100",
 "shares": {"Britney":1 } }
```

그림 29. 변수 선언부 변경에 대한 S+AST 구축 예시

IV. 결론

본 논문에서는 AST를 기반으로 소스코드에 대한 개발자별 지분을 파악하는 연구를 진행하였다. 이를 위해 S+AST를 구축하는 과정에서 word-diff의 결과를 이용해 변경위치 기반으로 AST 차이를 계산하였다. 이 때문에 도구의 성능에 따라 AST 차이가 달라질 수 있다. 하지만, 이는 제안하는 접근법과는 독립적으로 변경 및 개선이 가능하며, 스택 추적(Stack Trace) 정보를 이용할 경우 버그추적시스템에서 특정 버그에 가장 적합한 개발자를 선정하는 연구에도 도움을 줄 수 있다. 향후에는 코드 변경 시점으로부터 지분 계산 시점까지의 시차를 가중치로 반영하여 시간이 오래될수록 개발자의 지분을 낮춤으로써 지분 계산의 정확도를 개선할 계획이다.

References

[1] W. F. Tichy, "RCS—A System for Version Control", Proceedings of the Software-Pracetice & Experience, Vol. 15, pp.637-654, Jul 1985.

[2] J. Anvik, L. Hiew, and G. C. Murphy, "Who Should Fix This Bug," Proceedings of the 28th International Conference on Software Engineering, pp.361-370, 2006.

[3] C. Teyton, M. Palyart, J. Falleri, F. Morandat, and X. Blanc, “Automatic Extraction of Developer Expertise,” Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering, pp.8:1-8:10, 2014.

[4] P. Bhattacharya, I. Neamtiu, and M. Faloutsos, “Determining Developers' Expertise and Role: A Graph Hierarchy-Based Approach,” Software Maintenance and Evolution (ICSME), 2014 IEEE International Conference on, pp.11-20, Sept 2014.