소프트웨어 최근 변경을 클래스 다이어그램으로 가시화

심재경[O1], 조희태[1], 박종열[2], 이선아[1, 2]
[1]항공우주 및 소프트웨어공학전공, 경상대학교
[2]정보과학과, 경상대학교
vxeimv@naver.com, cht1992@naver.com, bjng237@gnu.ac.kr, saleese@gnu.ac.kr

Visualizing a Class Diagram from the Recent Revision of Software Change History

JaeKyeong Sim[O1], HeeTae Cho[1], Jongyeol Park[2], Seonah Lee[1, 2]
[1]Department of Aerospace and Software Engineering, Gyeongsang National University
[2]Department of Informatics, Gyeongsang National University

요 약

개발자들이 소프트웨어 시스템을 이해하고 변경을 수행할 때 도움을 줄 수 있다는 방향에서 소프트웨어 가시화 연구가 진행되어 왔다. 최근에 들어 제시되고 있는 상향식 소프트웨어 가시화 도구들은 개발자에게 직접적으로 연관이 되는 코드 정보만을 보여주는 이점으로 개발자들의 활용에 있어 효과를 입증하고 있다. 그러나 보여주는 정보를 개발자 탐색 활동에 한정되게 가시화하는 제약점이 있다. 본 논문은 상향식 가시화 도구에서 좀 더 유용한 정보를 제공하기 위하여 소프트웨어 개정 이력을 개발자들이 소프트웨어 시스템을 변경할 때 보여줄 수 있는 도구를 제시한다. 제시하는 가시화는 개발자들이 커밋한 정보를 클래스 다이어그램으로 볼 수 있도록 도와주어, 코드 변경에 대한 빠른 이해를 돕는다.

1. 서 론

소프트웨어 가시화는 개발자의 프로그램 이해를 돕기 위해 30년 이상 연구하고 개발한 연구 분야이다 [1]. 초기의 연구는 주로 소프트웨어 전체를 가시화하는 방법에 초점을 두었고 [2], 또한 프로그램 전체의 클래스 다이어그램을 자동 생성하는 도구를 찾아볼 수 있다 [3]. 그러나 전체 프로그램의 구조를 보여주고 그 다음 부분을 조회하고자 하는 하향식 소프트웨어 가시화 도구에서의 유용성은 좀처럼 증명되지 않았다.

최근 연구에서는 개발자가 탐색하는 코드 파트에 대해 가시화하는 상향식 접근 방식을 제안하였다 [4]. 또한 후속 연구에서 상향식 가시화 방법을 통해 개발자로부터 긍정적인 평가와 반응을 이끌어 냈다 [5]. 이러한 상향식 방법은 개발자가 방금 작업한 코드에 대해서 다시 상기할 때 도움을 준다. 하지만, 그 보여주는 부분이 개발자가 이미 기억을 하고 있을 때에는 도움이 되지 않는 한계점이 있다.

본 논문은 개발자의 긍정적인 영향을 확인한 상향식 가시화 방법에서 추가적인 연구를 진행한다. 본 논문에서 새로 제시하는 가시화는 개발자가 최근에 변경하여 소프트웨어 변경 관리 도구에 커밋한 코드 변경 기록을 클래스 다이어그램으로 조회할 수 있도록 하는 부분이다. 이는 이번 개발 세션에서 탐색하는 파트를 가시화하고 이를 개발자가 조회하는 것으로 한정하지 않고, 개발자가 어제 개발하거나 한동안 떠나있다가 다시 해당 개발 프로젝트로 참여할 때 최근의 코드 변경 내역을 쉽게 조회하여 기억을 되살리는 부분을 돕고자 한다.

본 논문의 저자들은 이러한 최근 변경에 대한 개정 이력 조회를 통해서 개발자가 최근의 변경에서 함께 변경한 클래스들과 그 클래스들의 관계에 대해서 쉽게 상기하고, 또한 최근 변경한 메소드와 필드에 대한 조회를 함으로써 그 다음의 후속 변경 작업을 위한 프로그램 이해를 쉽고 빠르게 할 수 있으리라 기대한다.

본 논문의 구성은 다음과 같다. 먼저 2절에서는 개발자들의 사용 기대 시나리오를 기술한다. 3절에서는 우리가 개발한 도구의 접근 방법을 기술한다. 4절에서는 개발한 도구에서 가능한 시나리오의 데모를 보인다. 5절에서는 관련 연구를 간단히 요약한다. 6절에서는 결론을 지으며 논문을 마무리한다.

2. 시나리오

본 논문의 사용 시나리오에서의 질문은 다음과 같다: “개발자인 내가 어제 작성, 변경했던 코드를 클래스 다이어그램으로 한 번에 조회할 수 있는가?” 이를 위하여 가정한 시나리오는 다음과 같다.

1) 개발자 A는 Java언어로 구현한 프로젝트를 관리하기 위하여 Eclipse IDE와 Git 변경 관리 도구를 이용한다.
2) 개발자 A는 프로젝트에 변경을 만들 때마다 변경 이력을 저장한다. 최근 클래스 X, Y, Z에 메소드 추가 및 수정 작업을 하고 하루 일을 끝낸다.
3) 개발자 A는 다음날 다시 Eclipse IDE를 오픈하고 작업할 프로젝트를 선택한다. 개발자 A는 어제 변경을 커밋했던 이력에 해당하는 클래스들과 그 클래스에서 변경한 메소드들을 조회하기를 원한다.

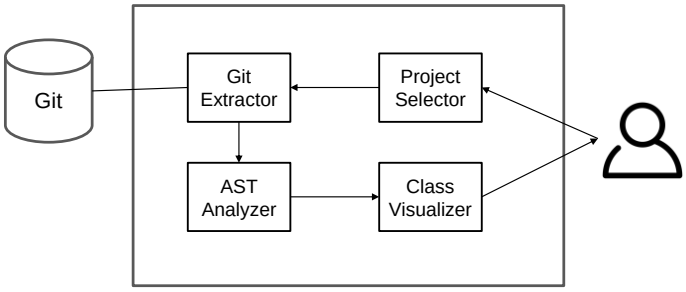

그림 1. 제안 도구에 대한 구조

4) 현재 변경 관리 도구 Git에서는 최근 변경한 내역에 대해서 클래스들을 트리 뷰에서 목록으로 보여주고 각 클래스들을 선택하면 추가와 삭제 내용을 보여주지만 이러한 방식은 변경한 코드 구조를 일목요연하게 보여주지 않는다.

상기 시나리오에서 개발자 A가 원클릭으로 변경한 클래스와 그 사이의 관계, 그리고 변경한 메소드의 목록을 간략히 조회할 수 있는 클래스 다이어그램은 개발자의 이해를 도울 수 있다. 개발자는 코드에서 어디까지 작업했는지를 확인하고, 그 다음 작업을 위한 코드 구조를 쉽게 파악할 수 있다.

3. 접근 방법

제시하는 도구의 기능, 품질 요구사항은 다음과 같다.

1) 최근의 변경 기록을 가져와서 변경한 클래스와 메소드를 명시한다.
2) 명시한 클래스와 메소드를 UML 클래스 다이어그램으로 표현한다.
3) 사용이 용이해야 한다. 예를 들어 원 버튼 클릭으로 어제의 이력을 클래스로 표현할 수 있어야 한다.

이러한 요구사항을 반영하기 위한 제시 도구의 구조는 그림 1과 같다. 먼저 개발자가 선택한 프로젝트를 인식한다 (그림 1: Project Selector). 다음으로 선택한 프로젝트의 Git 시스템에 접근하여 최근의 개정 이력을 가져온다 (그림 1:Git Extractor). 개정 이력을 가진 파일 중 소스 코드에 해당 하는 파일에 대해서는 구조를 분석한다 (그림 1: AST Analyzer). 마지막으로 분석하여 도출한 변경 클래스와 그들의 관계, 또한 변경 메소드와 필드는 클래스 다이어그램으로 보여 준다 (그림 1: Class Visualizer). 제안 도구의 각 모듈에서 하는 일은 다음과 같다.

프로젝트 선택기 (Project Selector): 개발자가 프로젝트를 선택한 이벤트에 대한 리스너로서, 선택한 프로젝트가 Git 시스템과 연결하여 변경 이력을 관리하는 프로젝트인지 아니면 변경 이력을 관리하지 않고 워크스페이스에서 생성한 일반 프로젝트인지를 구별하며, Git 시스템과 연결된 프로젝트일 경우 프로젝트에 접근할 수 있는 경로를 획득한다.

Git 도출기(Git Extractor): Git에 있는 변경 기록에서 선택한 프로젝트에서 저장한 최근의 변경 이력을 가져온다. 최근 변경 이력은 변경한 파일의 목록과, 각 파일의 변경 타입, 그리고 변경 전, 후의 파일 내용을 담고 있다. Git 도출기를 위해서는 Git을 분석하는 라이브러리인 JGit을 이용하여 구축하였다.

AST 분석기(AST Analyzer): 상기 Git 도출기에서 가져온 파일의 목록 중 Java 소스 파일을 분석한다. 각 파일 단위로 분석하여 파일에 담긴 클래스, 메소드, 필드명을 도출한다. 또한 변경 전, 후의 파일에 대해 분석한 구조를 이용하여 변경한 클래스, 메소드, 필드를 명시한다.

클래스 시각기(Class Visualizer): AST 분석기에서 도출한 변경 클래스와 클래스 간의 관계, 변경 메소드 및 필드에 대해서 시각화한다. 또한 소스 파일이 아닌 파일들에 대해서는 클래스가 아닌 사각형 노드로 표시한다.

4. 데모

3절의 접근 방법으로 구현한 도구의 사용 데모는 다음과 같다. 사용자는 Eclipse IDE에서 프로젝트를 선택한다. 이 때 프로젝트는 Git 저장소와 연동이 되어 있어야 한다. 사용자는 프로젝트를 선택한 후 클래스 뷰 맨 앞에 존재하는 노란색 아이콘이 나타내는 revision history 버튼을 누른다. 클릭 시 제안 도구는 선택한 프로젝트의 가장 최신의 변경 이력을 가져와 클래스 다이어그램으로 보여준다.

그림 2는 이러한 결과로서의 클래스 다이어그램을 보여준다. 클래스 다이어그램에서 각각의 노드들은 가장 최근 개정 이력에서 변경된 클래스 들과 메소드들을 나타낸다. 예를 들어 그림 2에서는 ShowHistoryAction, JavaEditorSelectionListener, Java-Modification 클래스에서 1~3개의 메소드를 변경하였음을 볼 수 있다. 노드들 사이의 선은 그들 사이의 관련성을 표현한다. 만약 사용자가 노드를 더블클릭하면, 그 노드에 맞는 자바 파일을 에디터에 표시한다. 추가 기능으로 Auto/Manual버튼으로 노드들의 Layout 모드를 선택할 수 있다. Clear버튼으로 클래스 뷰의 다이어그램을 지울 수 있다.

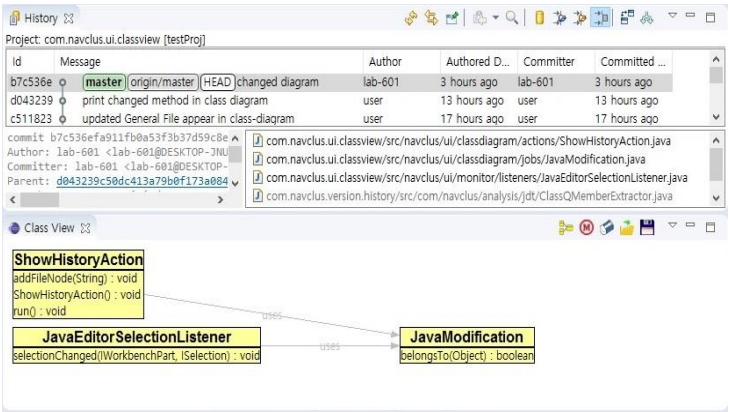

그림 2. 데모 화면

5. 관련연구

개발자들의 활동을 중심으로 코드를 가시화하는 여러 연구들이 있었다. 첫째는 인터랙션 히스토리를 기반으로 클래스 다이어그램을 가시화하는 연구이다. 2013년 Lee et al. 은 개발자 인터랙션 히스토리를 기반으로 클래스 다이어그램을 만드는 도구를 제시하였다 [6]. 또한 2016년에는 구현한 도구를 활용하여 어떠한 상황에서 개발자들이 클래스 다이어그램을 활용하는지를 조사하였다 [7]. 이러한 접근 방법은 상향식 방향에서 진행하였다는 점에서 본 논문과 유사하지만, 다른 종류의 인터랙션 히스토리를 가시화하였다는 점에서 본 논문과 다르다.

둘째는 소프트웨어 변경에 대한 가시화 연구이다 [8]. Yoon et al.은 인터랙션 히스토리를 바탕으로 상세한 코드 변경 이력을 가시화하는데 초점을 두었다. 저자들이 제시한 시각화는 타임라인 시각화 (timeline visualization)와 코드 변경의 시각화 (code history diff view)이다. 또한 개발자들이 필요로 하는 정보를 시각화를 통해 쉽게 제공할 수 있음을 시나리오를 제시를 통해 논의하였다. 연구 [8]도 연구[6][7]와 같이 인터랙션 히스토리를 사용하였다.

끝으로 2016년 Wittenhagen et al.은 코드 일부 정보를 가시화하는 크로니클러 (Chronicler)라는 도구를 제시하였다 [9]. 해당 도구는 트리 플로우뷰와 소스코드뷰로 구성되어 있다. 사용자가 소스코드 히스토리를 표현하는 트리 플로우 뷰에서 특정 위치를 클릭할 때 관련 버전의 파일 내용을 소스코드뷰로 보여준다. 연구 [9]는 소스코드 히스토리를 가시화하였으나, 히스토리 중 한 파일의 일부 코드의 변경 정보를 보여주는 제약점이 있다.

본 논문은 개발자들이 일반적으로 사용하는 코드 개정 히스토리를 바탕으로 과거의 이력 중 최근 이력을 가시화한다. 우리의 연구는 클래스들에 대한 최근 변경을 빨리 조회할 수 있도록 시각화하는 데 초점을 두고 있어 시각화 대상이 기존과 다르다.

6. 결론

제안 도구는 개발자가 최근 변경한 여러 클래스 간의 구조를 쉽게 파악할 수 있도록 변경 기록을 클래스 다이어그램으로 가시화한다. 이를 위하여 사용자가 선택한 프로젝트의 리비전 히스토리에서 최근 개정 이력을 가져와서 변경한 코드를 분석한 후 클래스 다이어그램으로 모델링한다. 이러한 가시화는 개발자들이 다음날 혹은 여러 날 지난 후에 다시 작업을 시작할 때 최근 변경 내역을 바로 보여주어 코드 변경 빨리 상기하여 후속 변경 작업으로 나아가는 것을 도와줄 수 있다.

제시하는 도구는 현재 프로토타입으로서 연구, 개발을 통해 여러 이슈들을 도출할 수 있었다. 예를 들어 Git 저장소에는 여러 개의 프로젝트를 함께 등록할 수 있는데, 이 경우 선택한 프로젝트에서 가장 최근에 변경한 이력과 Git 저장소의 최근의 변경 이력이 동일하지 않음을 발견하였다. 또한 최근 변경 이력이 아닌 하루 종일이라는 기간, 최근 일주일 동안 변경한 이력을 한 번에 보고자 하는 가시화가 필요한지도 논의하였다. 그리고 클래스 다이어그램에서 아직 변경된 내역을 표시하고 있지 않은데, 어떤 방식으로 변경된 클래스와 메소드를 표기할지도 논의하였다. 향후 연구로는 이러한 이슈를 해결함과 동시에 사용자 연구를 수행하여 사용자 피드백을 모아 제시 도구를 완성된 도구로 개발하고자 한다.

7. 참고문헌

[1] Storey, M-A. "Theories, methods and tools in program comprehension: Past, present and future." Program Comprehension, 2005. IWPC 2005. Proceedings. 13th International Workshop on. IEEE, 2005.

[2] Storey, Margaret-Anne, et al. "SHriMP views: an interactive environment for information visualization and navigation." CHI'02 Extended Abstracts on Human Factors in Computing Systems. ACM, 2002.

[3] Class Visualizer, https://www.class-visualizer.net/, accessed in Mar. 29. 2017.

[4] Sinha, Vineet, David Karger, and Rob Miller. "Relo: Helping users manage context during interactive exploratory visualization of large codebases." Visual Languages and Human-Centric Computing, 2006. VL/HCC 2006. IEEE Symposium on. IEEE, 2006.

[5] Bragdon, Andrew, et al. "Code bubbles: a working set-based interface for code understanding and maintenance." Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. ACM, 2010.

[6] Seonah Lee, Sungwon Kang, Matt Staats "A Graphical Recommender for Assisting Code Exploration," 35th IEEE International Conference on Software Engineering (ICSE Formal Demonstrations 2013).

[7] Lee, Seonah, and Sungwon Kang. "What situational information would help developers when using a graphical code recommender?." Journal of Systems and Software 117 (2016): 199-217.

[8] Yoon, YoungSeok, Brad A. Myers, and Sebon Koo. "Visualization of fine-grained code change history." Visual Languages and Human-Centric Computing (VL/HCC), 2013 IEEE Symposium on. IEEE, 2013.

[9] Wittenhagen, Moritz, Christian Cherek, and Jan Borchers. "Chronicler: Interactive exploration of source code history." Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems. ACM, 2016.