CIAT: 소프트웨어 자산 변경 영향 분석 도구

이혜선*, 빈타오**, 강교철*, 이숙희***
*포항공과대학교 컴퓨터공학과
**Scality, 파리, 프랑스
***삼성전자 반도체 사업부, SW 선행연구 P/J, SE 그룹
e-mail : compial[at]postech.ac.kr

CIAT: A Tool for Change Impact Analysis of Software Assets

Hyesun Lee*, Tao T. Vinh**, Kyo Chul Kang*, and Sukhee Lee***
*Department of Computer Science and Engineering,
Pohang University of Science and Technology (POSTECH)
**Scality, Paris, France
***SE Group, Advanced SW Research P/J, Semiconductor Business, Samsung Electronics

요 약

소프트웨어 자산을 효과적으로 유지보수하기 위하여 자산의 어느 부분을 수정할 때 변경에 의해 영향을 받는 부분을 예측하여 필요한 경우 함께 수정하도록 하는 방법이 필요하다. 이러한 변경 영향 분석 방법을 사용하면 개발자는 소프트웨어 자산을 수정할 때 함께 수정해야 할 부분을 수동으로 찾는 노력을 줄일 수 있고, 함께 수정이 필요한 부분을 빠뜨리지 않을 수 있어서 자산 변경 시 발생할 수 있는 오류를 방지할 수 있다. 변경 영향 분석 방법을 기업의 소프트웨어 개발에 실제적으로 적용하기 위해서는 방법을 지원하는 도구가 필수적이다. 하지만 기존 상용도구들을 현업에서 사용할 때 한계가 있어 개발자의 요구사항을 반영한 새로운 변경 영향 분석 도구의 개발이 필요하였다. 본 논문에서는 이러한 도구의 요구사항을 설명하고, 이를 반영하여 개발한 새로운 지원도구를 제안한다. 제안하는 도구는 플래시 메모리 소프트웨어 도메인에 적용되어 검증되었다.

1. 서론

개발된 소프트웨어 자산은 새로운 고객 요구사항, 변화하는 제품 운영환경, 새로운 품질 요구사항, 에러 수정 등을 위하여 끊임없이 유지보수 되어야 한다. 이러한 유지보수 기간은 전체 생명주기의 2/3 정도 되며 [1], 비용의 약 3/4 이 유지보수에 사용된다 [2]. 따라서 소프트웨어 개발 기업의 성공을 위해서는 소프트웨어 자산의 유지보수성 (Maintainability)을 높이는 것이 중요하다.

소프트웨어 자산을 효과적으로 유지보수하기 위하여 자산의 어느 부분을 수정할 때 변경에 의해 영향을 받는 부분을 예측하여 필요한 경우 함께 수정하도록 하는 방법이 필요하다. 이러한 방법을 '변경 영향 분석 (Change Impact Analysis) 방법'이라고 한다. 이 방법을 사용하면 개발자는 소프트웨어 자산을 수정할 때 함께 수정해야 할 부분을 수동으로 찾는 노력을 줄일 수 있고, 함께 수정이 필요한 부분을 빠뜨리지 않을 수 있어서 자산 변경 시 발생할 수 있는 오류를 방지할 수 있다.

기존에 소프트웨어 자산의 변경 영향 분석 방법이 활발하게 연구되었고 이를 지원하는 도구들이 소개되었다 [3-12]. 기업의 소프트웨어 개발에 변경 영향 분석 방법을 실제적으로 적용하기 위해서는 특히 변경 영향 분석 방법을 지원하는 도구가 필수적이다. 기존에 변경 영향 분석을 지원하는 상용도구 [13, 14]가 개발되어 기업의 소프트웨어 개발에 사용되고 있다.

하지만 이러한 상용도구들을 실제 제품 (플래시 메모리 소프트웨어 제품) 개발에 사용할 때 한계가 있어 개발자의 요구사항을 반영한 새로운 변경 영향 분석 도구의 개발이 필요하였다. 본 논문에서는 이러한 도구의 요구사항을 설명하고 (2 장), 이를 반영하여 개발한 새로운 지원도구를 제안한다 (3 장).

2. 지원도구 요구사항

지원도구의 요구사항을 파악하기 위하여 두 가지 방법을 사용하였다. 우선, 기존 상용도구 [13, 14]를 분석하여 기존 도구의 한계를 파악하였다. 그리고 개발자와의 인터뷰를 통하여 변경 영향 분석 도구를 현업에 적용할 때 필요한 VOC (Voice of Customer)를 파악하였다. 분석한 핵심 요구사항을 정리하면 다음과 같다.

첫 째: 도구가 IDE 에 플러그인 형태로 제공되어야 한다. 변경 영향 분석 도구의 사용성 (Usability)을 위하여 기존 소프트웨어 개발에 사용되는 IDE (Eclipse 또는 Visual Studio)에 플러그인 (Plug-in) 형태로 도구가 제공될 필요가 있다. 이처럼 도구를 플러그인 형태로 제공하면, 개발환경에 쉽게 설치할 수 있고, 소프트웨어 자산 개발과 유지보수를 지원하는 다른 도

구들과 쉽게 통합할 수 있다.

둘 째: 다양한 분석 단위에서 변경 영향 분석을 할 수 있어야 한다. 변경 영향 분석에서 중요한 지표 (Measurement)가 안전성 (Safety)과 정확성 (Precision)이다. 안전성은 실제 변경 영향을 받는 요소들이 얼마나 IS (Impact Set; 변경 영향 분석 도구가 변경 영향을 받을 것으로 예측하는 요소들의 집합)에 속하는지를 나타내고, 정확성은 IS 와 실제 변경 영향을 받는 요소들이 어느 정도 유사한지를 나타낸다. 변경 영향 분석을 다양한 분석 단위 (Granularity) – 즉, 코드 스테이트먼트 (Statement), 함수 (Method), 클래스 (Class) 단위 – 에서 할 수 있는데, 더 세밀한 단위에서 분석을 할수록 안전성이 높아지지만 정확성은 낮아지게 된다. 소프트웨어 유지보수 상황에 따라서 안전성과 정확성 중 어느 것이 중요한지가 다를 수 있고, 그에 적합한 변경 영향 분석 단위를 선택하여야 한다. 하지만 기존 도구에서는 이러한 기능이 부족하였다. 따라서 안전성을 위한 스테이트먼트 단위 분석과 정확성을 위한 함수 단위 및 클래스 단위 분석을 모두 지원하는 도구가 필요하다.

셋 째: IS 의 원소를 분석하는 것을 돕는 기능이 필요하다. 변경 영향 분석 도구를 사용하여 IS 를 예측한 뒤에, 개발자가 IS 의 원소 (Element)들을 확인하여 실제 변경이 필요한 원소인지 결정하는 작업이 필요하다. 하지만 기존 도구에서는 이러한 작업을 지원하지 않아서 개발자가 IS 의 어느 원소까지 확인했는지를 일일이 기억해야 했다. 또한 IS 의 원소를 분석할 때 원소가 위치한 코드부분을 실시간으로 확인을 해야 하는데, 기존 도구에서는 이를 위한 UI 가 사용하기 불편하게 설계되어 있었다. 이러한 문제점을 해결하고 개발자가 IS 원소 분석을 효과적으로 할 수 있도록 돕는 기능이 필요하다.

넷 째: 도구의 수행 속도와 UI 는 상용도구를 벤치마킹하여야 한다. 도구의 수행 속도와 UI 가 지원도구를 현업에 적용하는데 중요한 영향을 미친다. 개발자들이 기존 상용도구의 수행 속도와 UI 에 대체로 만족하기 때문에, 이를 벤치마킹 (Benchmarking)하여 제안하는 도구의 수행 속도와 UI 가 뒤떨어지지 않도록 해야 한다.

3. 제안하는 지원도구

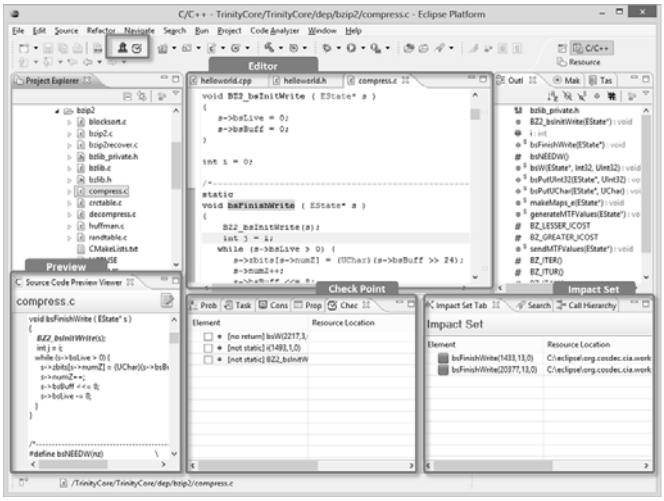
앞서 살펴 본 요구사항을 바탕으로 기존 도구의 한계를 보완한 새로운 변경 영향 분석 지원도구인 'CIAT'를 개발하였다 (그림 1). 제안하는 도구가 각 요구사항을 어떻게 해결하였는지 살펴보면 다음과 같다.

첫 째: Eclipse CDT IDE 에 플러그인 형태로 제공되어 도구의 확장성이 높다. 제안하는 도구는 Eclipse CDT IDE [15]에 플러그인 형태로 제공되어 현업 개발환경에 쉽게 설치할 수 있다. 또한 자산 개발과 유지보수를 지원하기 위한 플러그인 형태 도구들과 쉽게 통합될 수 있어서 확장성이 높다.

둘 째: 다양한 분석 단위의 변경 영향 분석을 제공한다. 제안하는 도구는 세 가지 단위, 즉, 코드 스테이트먼트, 함수, 클래스 단위의 변경 영향 분석을 제공한다. 개발자가 변경 영향 분석 전에 원하는 단위를 선택할 수 있으며, 선택된 단위에서 변경 영향 분석을 하여 IS 를 계산한다.

셋 째: IS 의 원소 분석을 돕는 기능을 지원한다. 제안하는 도구는 IS 의 원소 분석을 돕기 위해 다음의 두 가지 기능을 지원한다. 첫 째, 개발자가 IS 의 원소를 분석할 때 원소를 확인한 상태 (즉, 원소의 Status of Visit)를 기록하여, 어느 원소까지 확인하였는지를 개발자가 일일이 기억하지 않아도 되도록 지원한다. 둘 째, 원소를 확인할 때, 원소가 위치한 코드부분을 실시간으로 쉽게 확인할 수 있는 프리뷰 (Preview) 화면을 제공하여 (그림 1 의 Preview 상자), 원소 분석을 쉽게 할 수 있도록 지원한다.

넷 째: 빠른 수행 속도와 편리한 UI 를 제공한다. 제안하는 도구의 수행 속도와 UI 는 기존 상용도구를 벤치마킹하여, 빠른 수행 속도와 편리한 UI 를 제공하도록 개발되었다. 도구를 플래시 메모리 소프트웨어 도메인에 적용하여 이를 검증하였다.



(그림 1) 지원 도구 화면

제안하는 도구는 이 외에도 다음의 장점을 갖고 있다.

IS 를 Direct/Indirect 단계 별로 제공한다. IS 의 원소는 변경에 의해 직접적 (Direct)으로 제어/데이터 흐름 영향을 받는 원소가 있고, 변경에 의해 간접적 (Indirect)으로 제어/데이터 흐름 영향을 받는 원소가 있다. 제안하는 도구는 IS 의 원소들을 Direct/Indirect 단계 별로 제공하여 IS 의 원소를 효과적으로 파악할 수 있게 한다. 특히 Indirect 단계의 IS 를 미리 계산하는 것이 아니라 개발자가 요청할 때 계산하여 도구의 성능을 높였다.

코드의 에러/워닝을 찾는 체크포인트를 제공한다. 개발자가 코드를 변경한 뒤에 오류가 있는지를 확인하여 필요한 경우 개발자에게 에러 (Error) 또는 워닝 (Warning)을 주는 체크포인트 (Check Points) 기능이 필요하고, Eclipse CDT IDE 가 이를 제공하고 있다. 제안하는 도구에서는 Eclipse CDT IDE 가 확인하지 못하는 체크포인트를 확인하는 가능을 추가적으로 제공한다.

(예를 들어, 추가된 변수가 사용되지 않을 경우 워닝을 주어야 하지만 Eclipse CDT IDE 에서는 이를 지원하지 않는다.) 특히, 컴파일 (Compilation) 없이도 체크포인트를 제공할 수 있도록 하였다.

4. 결론 및 향후 연구

제안하는 소프트웨어 변경 영향 분석 지원도구는 현업의 요구사항을 반영하여 기존 상용도구의 한계를 보완하였기 때문에, 현업 소프트웨어 개발 및 유지보수에 효과적으로 적용될 것으로 기대된다. 개발된 도구는 플래시 메모리 소프트웨어 도메인에 적용되어 검증되었다.

현재 지원도구에서는 코드 레벨의 의존관계 (Dependency)를 기반으로 변경 영향을 분석한다. 하지만 코드 레벨의 의존관계를 기반으로 파악한 IS 의 원소가 실제로 변경되어야 한다는 당위성이 부족하다. 이를 해결하기 위하여 코드 레벨의 의존관계뿐만 아니라, 문제 사이의 개념적 관계에 의한 IS 추출 방법이 필요하다. 따라서 후속연구로 기능 – 즉, 휘처 (Feature) – 레벨의 개념적 관계를 기반으로 한 변경 영향 분석 방법을 할 연구할 계획이다. 또한 개발한 도구를 지속적으로 여러 도메인에 적용하고 개발자의 피드백을 받아서 도구의 기능과 품질을 개선할 계획이다.

참고문헌

[1] M. V. Zelkowitz, A. C. Shaw, and J. D. Gannon. “Principles of software engineering and design.” Prentice-Hall, 1979.

[2] A. A. Takang, and P. A. Grubb, “Software maintenance concepts and practice.” Thompson Computer Press London, UK, 1996.

[3] K. B. Gallagher and J. R. Lyle. “Using program slicing in software maintenance.” IEEE Transactions on Software Engineering, vol. 17, no. 8, 1991, pp. 751-761.

[4] J. P. Loyall and S. A. Mathisen. “Using dependence analysis to support the software maintenance process.” In Software Maintenance, 1993. In: CSM-93, pp. 282-291. IEEE, 1993.

[5] D. Kung, J. Gao, P. Hsia, F. Wen, Y. Toyoshima, and C. Chen. "Change impact identification in object oriented software maintenance." In: International Conference on Software Maintenance (ICSE) 1994, pp. 202-211. IEEE, 1994.

[6] X. Dong and M. W. Godfrey. "System-level usage dependency analysis of object-oriented systems." In: ICSM 2007, pp. 375-384. IEEE, 2007.

[7] J. Jász, Á. Beszédes, T. Gyimóthy, and V. Rajlich. "Static execute after/before as a replacement of traditional software dependencies." In: ICSM 2008, pp. 137-146. IEEE, 2008.

[8] M. Acharya and B. Robinson. "Practical change impact analysis based on static program slicing for industrial software systems." In: the 33rd International Conference on Software Engineering (ICSE), pp. 746-755. ACM, 2011.

[9] M. Petrenko and V. Rajlich. "Variable granularity for improving precision of impact analysis." In Program Comprehension, In: ICPC'09, pp. 10-19. IEEE, 2009.

[10] J. Buckner, J. Buchta, M. Petrenko, and V. Rajlich. “JRipples: A Tool for Program Comprehension during Incremental Change.” In: IWPC, vol. 5, pp. 149-152. 2005.

[11] K. Chen and V. Rajlich. "RIPPLES: tool for change in legacy software." In: ICSM'01, p. 230. IEEE Computer Society, 2001.

[12] S. Gwizdala, Y. Jiang, and V. Rajlich. "JTracker-a tool for change propagation in Java." In: 2011 15th European Conference on Software Maintenance and Reengineering, pp. 223-223. IEEE Computer Society, 2003.

[13] Source Insight. http://www.sourceinsight.com/.

[14] Visual Assist-X. http://www.wholetomato.com/.

[15] Eclipse CDT. http://www.eclipse.org/cdt/.