2014
IIBC
CONFERENCE

www.iibc.kr

종합학술대회 논문집

제12권 제1호

일시 | **2014. 11. 13(목)~14(금)**

장소 | 13(목) 을지대학교(성남캠퍼스), 14(금) 제주 그랜드호텔

주관 및 주최 | (사)한국인터넷방송통신학회(IIBC), (사)국제문화기술진흥원(IPACT), 을지대학교

후원 | 미래창조과학부, 방송통신위원회, 한국연구재단, 한국과학기술단체 총연합회, 한국인터넷진흥원, 정보통신산업진흥원, 전자신문 디지털타임스

협찬 | LG히다찌, LGNSYS, (주)지에스인스트루먼트, 대보정보통신(주), (주)아이지, (주)세인, (주)헬로웹, (주)맨엔텔, 영일교육시스템, (주)한백전자, 드림아이, 올포랜드, (주)씨이랩, (주)콤텍시스템, (주)경봉, 더블유에프지연구소(주), SJ정보통신

IIBC (사)한국인터넷방송통신학회
The Institute of Internet, Broadcasting and Communication

논문 목차 (구두)
11/13(목) 9:30~10:30

레거시 시스템의 코드 복잡도를 개선하기 위한 적절한 코딩 규칙 추출

Appropriate Coding Rule Extraction for Improving Code Complexity of Legacy System

문소영*, 김영수**, 이상은**, 박용범***, 김영철*

So-Young Moon*, YoungSoo Kim**, SangEun Lee**, Young B. Park***, R. Young Chul Kim*

msy@selab.hongik.ac.kr*, ysgold@nipa.kr**, selee@nipa.kr**, ybpark@dankook.ac.kr***, bob@hongik.ac.kr*

요 약

소프트웨어 시장 규모는 IT 융·복합화가 진행되면서, SW의 역할과 기능이 점차 확대되고 있으며, SW가 최종제품의 부가가치를 높이는 역할을 수행할 뿐만 아니라 최종제품의 경쟁력을 좌우하는 핵심 요소가 되었다. 이러한 이유로 소프트웨어의 비가시성이라는 특징과 복잡도가 증가되었다[1]. 소프트웨어 개발 과정 중에서 구현은 높은 비중을 차지한다. 그러나 개발 방향의 일관성, 가독성, 유지보수성, 버그 조기 방지와 성능 향상을 위해 개발자들이 준수해야 할 사항들을 작성해 놓은 코딩 가이드를 무시하고 개발하는 경우가 많다. 본 논문에서는 NIPA의 소프트웨어 가시화 기법을 통해 코드 모듈간의 결합도를 살펴보고, 코드 LOC와 사이클로 매틱 복잡도 측정 도구에 Code rules 통해 코드 복잡도를 측정 비교화 이다. 이 결과를 통해 적절한 코딩 규칙은 코드 복잡성을 개선하는 것을 확인이 가능하다.

키워드 : Code Complexity, Coding Guide, SW Visualization

I. 서 론

고품질의 소프트웨어를 개발하기 위해 다양한 소프트웨어 개발 프로세스가 개발되어 왔다. 그러나 좋은 소프트웨어 개발 프로세스를 통해 소프트웨어를 개발하였다 해도 잠재되어 있는 버그가 발생하여 납기지연, 품질저하 등의 문제가 발생할 수도 있고, 이는 SW의 복잡성을 증가시킬 수 있다[1]. 프로젝트의 크기에 따라서 구현은 전형적으로 프로젝트 전체 소요시간 중에서 30~80% 정도를 차지한다. 소스 코드는 항상 최신의 정보로 갱신되므로 소스 코드는 항상 최고의 품질을 유지해야 하고 소스 코드만이 소프트웨어를 정확하게 설명한다[2].

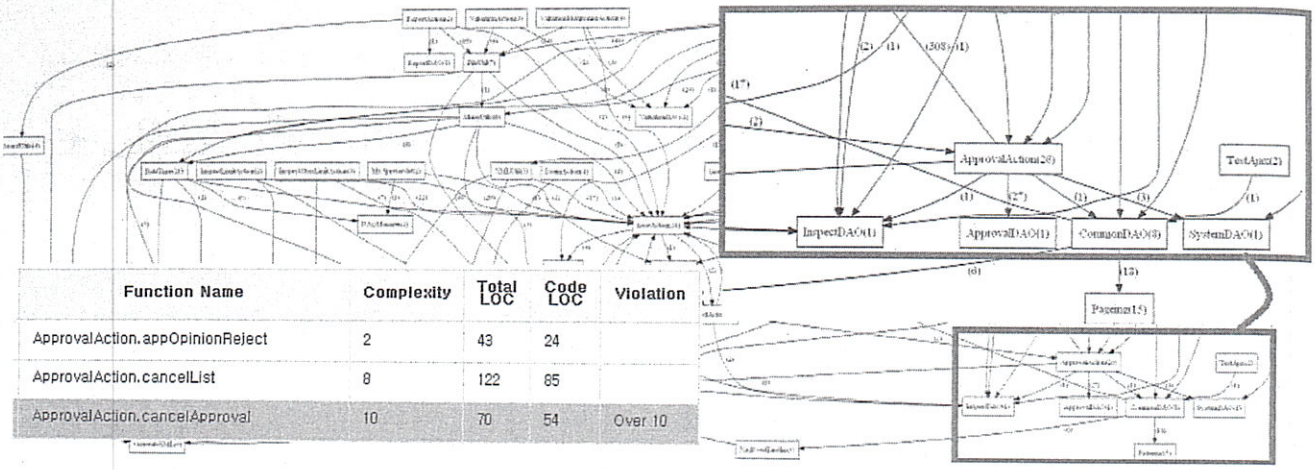
레거시 시스템의 유지보수를 어렵게 만드는 소프트웨어의 특성으로는 복잡성, 비가시성, 변경성이 있다[3]. 소프트웨어의 품질 관리와 유지보수 향상을 위해 SW Visualization을 통한 역공학 기법이 필요하다. 그리고 코딩 규칙이나 표준을 준수하지 않고 코딩을 한다면 그 소프트웨어에는 개발자의 나쁜 습관과 관련된 "Bad Smell"이 잠재적인 버그를 증가시키며 코드 단순화와 가독성을 저하시켜 소프트웨어의 품질관리에 나쁜 영향을 미친다.

본 논문에서는 SW Visualization과 코딩 규칙을 통해 레거시 시스템의 코드 복잡도를 개선시키고자 한다.

*홍익대학교 컴퓨터정보통신공학과
**정보통신산업진흥원 소프트웨어공학센터
***단국대학교 컴퓨터과학과

II. 코드 복잡도 분석

SW Visualization[1] 기법은 소프트웨어 품질관리와 유지보수 향상을 위해 NIPA SW 공학센터에서 개발한 것으로 소스 코드와 개발 프로세스 관리를 목적으로 시각화, 문서화를 통해 고품질 SW 관리를 한다. 소스 코드의 SW Visualization을 위한 ToolChain 프로세스를 통한 코드 결합도는 그림 1과 같다. 레거시 시스템의 결합도와 코드 복잡도 이다. 네모 박스로 표시된 ApprovalAction은 밖으로 나가는 값이 308로결합도가 매우 높다. 결합도가 높을수록 소프트웨어의 품질은 저하되므로 수정이 필요하다. ApprovalAction 클래스의 코드 복잡도 측정 결과 ApprovalAction.cancelApproval() 메서드에서 복잡도가 10이 나오는 것을 확인할 수 있다.



Function Name	Complexity	Total Loc	Code Loc	Violation
ApprovalAction.appOpinionReject	2	43	24	
ApprovalAction.cancelList	8	122	65	
ApprovalAction.cancelApproval	10	70	54	Over 10

그림 1. 레거시 시스템의 결합도와 코드 복잡도

표 1. 코드 복잡도를 개선시키기 위한 코딩 규칙

항목	설명	좋은 예	나쁜 예
공통	함수, 변수명은 가급적 16 이내의 이름을 사용[2].	numTeamMembers, teamPointsMax	np, numberOfPeopleOnTheUsOlympicTeam
	비슷한 이름은 사용을 배제[3].		persistentObject, persistentObjects

중첩	중첩을 최소화	if (userResult == SUCCESS){
		reply.writeErrors("error");
		} else {
		reply.writeErrors(userResult);
		}
		reply.done();
			if (userResult == SUCCESS){
			if (permissionResult != SUCCESS){
			reply.writeErrors("error");
			relpy.done();
			return;
			}
			reply.writeErrors("");
			} else {
			reply.writeErrors(userResult);
			}
			reply.done();

III. 코딩 규칙 추출

소프트웨어를 구현할 때 프로젝트의 규모에 따라 한 명 이상의 개발자가 코딩을 한다. 구현 단계에는 개발자들의 코딩 습관과 논리가 묻어나기 때문에 공통의 코딩 규칙과 코드 복잡성을 줄일 수 있는 규칙을 정하면 잠재적인 버그 "Bad Smell"을 줄이고, 소프트웨어 품질관리에 좋은 영향을 준다. 표 1은 코드 복잡도를 개선시키는 규칙이다. 표 1의 코딩 규칙을 적용하여 코딩한 클래스와 그렇지 않은 클래스들은 그림 1에서 보는 것처럼 눈에 띠게 결합도와 복잡도가 다른 것을 알 수 있다.

IV. 결 론

본 논문은 코드 복잡도 개선에 도움을 주는 코딩 규칙을 추출하여 구현에 적용하였다. SW Visualization의 ToolChain 기법을 통해 얻은 코드 결합도와 LOC에 따른 사이클로매틱 복잡도를 적용한 도구를 통해 코드 복잡도를 측정한 결과, 코딩 규칙을 적용하여 개발하면 코드 복잡도 개선에 좋은 영향을 미친다는 것을 알 수 있다.

향후에는 코드 복잡도를 줄 일 수 있는 코딩 규칙을 더 추출하여 소프트웨어 품질관리에 좋은 영향을 줄 수 있도록 할 것이다.

※ 이 논문은 2014년도 정부(미래창조과학부)의 재원으로 한국연구재단 - 차세대정보 · 컴퓨팅기술개발사업(No. 2012M3C4A7033348)과 2014년도 정부(교육부)의 재원으로 한국연구재단의 지원을 받아 수행된 기초연구사업임 (NRF-2013R1A1A2011601).

참 고 문 헌

[1] NIPA SW공학센터, SW 개발 품질관리 매뉴얼, NIPA 정보통신산업진흥원, 2013.

[2] Steve McConnell, Code Complete (2nd ed.), Microsoft Press, 2001.

[3] Dustin Boswell, Trevor Foucher, The Art of Readable Code, O'Reilly, 2011.