효율적인 자동화 코드 인스펙션(Automated Code Inspection)을 위한 필수 결함 검출 규칙 수립

곽수정*, 최진영**
*고려대학교 컴퓨터정보통신대학원 디지털정보미디어공학과
** 고려대학교 컴퓨터 · 전파통신공학과
e-mail : *soojung@korea.ac.kr, **choi@formal.korea.ac.kr

Rule of Defect Detection for the Effective Automated Code Inspection

Soo-Jung Kwak*, Jin-Young Choi**
*Dept. of Digital Information & Media Engineering, Korea University
** Division of Computer Science & Radio Communications Engineering, Korea University

요 약

프로젝트 개발에서 소프트웨어의 품질을 높이기 위한 방법 중 하나는 소스코드에 대한 잠재적인 결함을 초기에 발견하는 것이다. 이를 실현하기 위해 정형화된 기법으로 코드 인스펙션을 자동화하였으며, 개발자들이 ACI 규칙을 수립하였다. 논문에서는 실제 진행 중인 프로젝트를 기반으로 하여 결함 점검 수행에 따른 결함 발견 건수와 결함밀도가 감소되는 증명을 다룬다.

1. 서론

소프트웨어 산업의 발달로 프로젝트 개발 기간이 짧아 지고, 단기 개발을 선호하고 있지만, 소프트웨어 품질에서는 결함(Defect) 없는 높은 품질의 소프트웨어를 생산하기를 요구하고 있다. 특히, 테스트나 유지보수 단계에서 결함을 검출하고 수정하는 비용과 시간은 개발 초기 단계에서 수행하는 것보다 10~100 배에 이르게 된다[1].

본 논문에서는 인스펙션을 자동화하고, 소스코드의 결함을 조기에 검출하기 위해 5 개 社의 결함 검출 결과를 토대로 결함에 대한 필수 규칙을 수립하여 실제 프로젝트에 적용하여 효율성을 검증하고자 한다.

2. 인스펙션(Inspection)

목적은 결함을 조기에 발견 하고 제거하여 생산성을 높이는 것이다. 1976 년 Michael Fagan 에 의해 처음으로 정의되었다[2]. 인스펙션을 수행하여 프로젝트 개발을 위한 순수 절감은 35%~50% 이고, 전체 개발 일정은 25%까지 단축하였음을 보고하고 있다[3][4].

3. 코드 인스펙션 규칙 수립

3.1 Manual Code Inspection 의 문제점

Manual Code Inspection 방법은 가장 공식적이고 많이 사용하는 방법이다. 점검 과정의 모든 작업이 수동으로 진행되다 보니 코드의 직 · 간접적인 결함, 성능 저하 코드의 생성에 대한 결함을 발견하지 못하는 경우가 발생하게 되고 다량의 소스코드에 대한 전수 검사의 어려움과 검수 시간이 과하게 소요되고 있다.

3.2 결함(Defct) 검출 규칙 수립

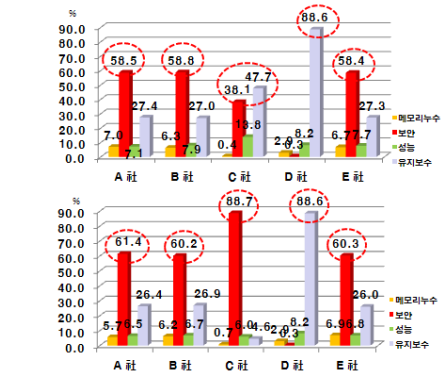
코드 결함을 ACI (Automated Code Inspection) 기법으로 효율적인 코드 인스펙션을 수행하기 위해 Sun 社에서 제공하는 자바 코드 표준 권고안을 토대로 하였으며, 5 개 社의 JAVA 또는 JSP 로 개발 된 프로젝트 산출물의 결함 검출 항목을 분석하여 메모리, 보안, 성능, 유지보수 4 가지 유형으로 분류하였다. 그리고 유형별로 결함 건수와 결함 밀도를 <표 1>에서 유형별 결함 비율은 (그림 1)에서 보여주고 있다.

<표 1> 5 개 社의 유형별 결함 검출

(단위:건수)

프로젝트명	A社		B社		C社		D社		E社	
개발언어	JAVA	JSP	JAVA	JSP	JAVA	JSP	JAVA	JSP	JAVA	JSP
점검 규칙 설정(개)	62	57	76	70	172	135	73	89	70	65
점검 배일 수	3,550	625	5	233	150	769	95	62	145	623
결함밀도(Defects/KLOC) - 코드1,000 라인 당 결함건수	60.2	2.0	530.0	174.7	23.3	75.7	18.0	5.3	9.2	63.7

유형	결함건수	결함건수	결함건수	결함건수	결함건수
메모리	123	66	95	63	68
보안	12,936	579	9	586	520
성능	4,698	76	269	79	72
유지보수	16,201	271	2,889	268	274
합계	33,958	992	3,262	996	934



(그림 1) 결함 비율

5 개 社의 프로젝트 산출물의 결함 검출 항목을 분석한 결과 개발자들이 공통적으로 범하는 결함 중에서 40 여 개를 필수 항목으로 추출하고 유형별로 분류하여 ACI(Automated Code Inspection) 규칙을 아래의 <표 2> 와 같이 수립하였다.

<표 2> ACI (Automated Code Inspection) 규칙

구분	검사항목	내용
개발표준	Boolean Argument Checking on Method	특정 클래스의 특정 메소드를 호출시 반드시 특정인자는 boolean으로 확실해야
	Throwing a same Exception within catch block	catch절에서 발생하지 않은 Exception 발생문을 점검합니다.
	Local Variable Naming	지역 변수는 소문자로 시작합니다
	Using a DB accounts	메서 Connection에 대한 계정이 노출되는 것을 금지하여 프로젝트의 표준 방법으로 Connection을 얻어야 합니다.
	Debug Statement	Console의 메시지를 점검합니다.
	Forbidden Method Invocation in the Catch	catch 구문내에서는 특정 메소드 호출을 하지 말아야 합니다.
	equals() called with null argument	equals 메소드호출시 null 인자는 사용하지 말아야 합니다.
	Unnecessary Constructors	불필요한 생성자는 정의하지 말아야 합니다.
	Throwing Exception in the catch Block	catch 구문의 throw 문장 사용을 점검합니다.
	Explicitly Created NullPointerException	NullPointerException을 명시적으로 생성하여 던지지(throw) 말아야 합니다.
	I/O release	I/O의 자원 해제를 점검합니다.
데이터 신뢰성	Unsafe Singleton Pattern	unsafe singleton 패턴을 사용하지 말아야 합니다.
	Floating Point Values	부동 소수점의 "==" 연산을 점검합니다.
	Using a ordered Calls for Connection	메서의 Connection에 대한 메소드 호출 순서를 점검합니다.
메트릭	Failure Definition Variable	사용하지 않는 private 변수명을 점검합니다.
	Static Variable for Connection	Static 형태로 JDBC CONNECTION을 선언하서는 안됩니다.
	Additive Compound Assignment	문자열의 "+=" 연산을 점검합니다.(순환문에서만 점검)
	Releasing a JDBC resource	ResultSet, Statement의 close() 메소드 호출을 점검합니다.
	Reusing a JDBC resource	메서에 대한 자원의 재사용성을 점검합니다.
	JMS release	JMS의 자원 해제를 점검합니다.
성능	Checking a class inherited from the Thread	Thread, Runnable, 또는 ThreadGroup 클래스를 상속한 클래스를 점검합니다.
	Using a EJBLocalHome instead of a EJBHome	EJB의 Remote 인터페이스를 상속 받음 인터페이스를 점검합니다.
	SerialVersionUID Checking	Serializable 인터페이스를 구현한 클래스는 serialVersionUID를 정의하여 합니다.
	Empty Block Body	빈 Block문장들(if, for, while, do)을 점검합니다.
	Empty try/catch/finally Block	빈 try, 빈 catch 그리고 빈 finally구문을 기술하지 말아야 합니다.
	Infinite Loop Statement	무한 루프 문장을 점검합니다.
	Array Copy in Loop	배열 복제시 System.arraycopy를 사용하여 합니다.
	Failure Definition Local Variables	사용하지 않는 지역 변수를 점검합니다.
	SELECT query with all-column	SELECT 필요이 컬럼명을 지정하지 않은 SQL을 점검합니다.
	Volatile Variables	volatile 변수는 jvm마다 구현방식이 조금 달라질수도 사용을 금지합니다.
시스템 안정성	System.gc() Invocation	System.gc() 또는 Runtime.gc() 메소드 호출을 점검합니다.
	Runtime.runFinalization() Invocation	System.runFinalization()또는 Runtime.runFinalization() 메소드 호출을 점검합
	Thread Methods Invocation	Thread의 메기능 메소드를 점검합니다.
유지 보수	Class Comment	클래스 주석을 기술합니다.
	Variable Naming	소문자를 시작하여 첫글자 이후에 "_"를 금지합니다.
	Method Comment	메소드 주석을 기술합니다.
	Duplicated Variable Names	전역 변수와 지역 변수의 중복을 이름을 점검합니다.
보안	Using a PreparedStatement instead of Statement	메서에 대한 자동화 Statement 사용을 금지합니다 (sql injection, sql parsing)
	Constant Interface	상수성 interface는 정의하지 말아야 합니다.
잠재적결함	Switch Statement	break가 없는 case 문의 존재를 점검합니다.
	Forbidden String equal-Expression	문자열 비교는 문자열 비교 메소드를 사용해야 합니다.

4. ACI(Automated Code Inspection) 규칙 적용 사례

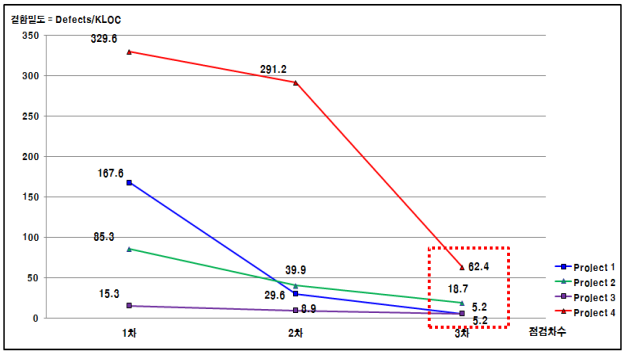
본 논문에서는 4 주에 걸쳐 4 개의 프로젝트를 대상으로 ACI 규칙을 적용하여 결함을 검출하기 위해 3 차례의 점검을 수행하였다. ACI 를 수행하기 위해 점검한 파일은 4 개 프로젝트의 총 38,930 개 이다. 점검 수행 결과 1 차에서 검출된 결함은 총 17,683 건이고, 2 차 수행에서는 20% 정도 감소한 14,143 건의 결함이 검출되었으며, 마지막 3 차 수행에서는 80% 정도 감소한 3,368 건의 결함이 검출되었다. 검출된 결함 항목을 바탕으로 점검차수에 따라 <표 3> 에 유형별로 정리하였다.

<표 3> 프로젝트 별 결함 분석표

(단위: 건수)

프로젝트		Project1			Project2			Project3			Project4		
개발언어		JAVA/JSP			JAVA/JSP			JAVA/JSP			JAVA/JSP		
점검 차수		1차	2차	3차	1차	2차	3차	1차	2차	3차	1차	2차	3차
점검 파일 수		13,288			5,683			10,226			9,733		
결함밀도(Defects/KLOC) - 코드1,000 라인 당 결함건수		329.6	291.2	62.4	167.6	29.6	5.2	85.3	39.6	18.7	15.3	8.9	5.2
중요도	유형	결함건수			결함건수			결함건수			결함건수		
필수	메모리	102	75	59	66	27	8	59	40	24	60	39	23
	보안	12,936	11,584	1,956	579	78	10	9	5	3	506	327	175
	성능	873	657	426	65	31	5	240	156	84	65	42	37
	유지보수	678	573	321	250	34	7	808	317	134	262	158	96
	합계	14,589	12,889	2,762	960	170	30	1,116	518	245	973	566	331

(그림 2)에서는 ACI 수행 결과 기간에 따라 프로젝트 별로 결함 밀도가 낮아지고 있는 추이를 보여주고 있다.



(그림 2) 결함밀도 추이

ACI 규칙을 코드 인스펙션의 기본 검증 단계로 활용한다면 프로젝트 수행 시 개발일정 단축과 비용 절감 등 소프트웨어 품질 향상을 도모할 수 있을 것으로 생각된다.

5. 결론

본 논문에서는 JAVA 나 JSP 개발 환경에서 ACI 방법으로 개발자들이 많이 발생시키는 결함에 대해 필수 결함 항목을 추출하여 규칙을 수립하였다. 결함을 사전에 발견하고, 개발 표준 준수로 가독성 및 유지보수 효율성 증대 등 지속적인 관리 수준 향상을 위한 방법으로 필수 결함 규칙을 제시하였다. 이를 실제 프로젝트 사례에 적용하여 규칙의 결함 검출 성능 수준을 결함건수와 결함밀도로 수치화하였으며, 결함이 감소되고 있어 코드 인스펙션을 효율적으로 관리할 수 있음을 증명하였다.

이 논문에서 제시한 결함 규칙을 향후 다양한 자동화 도구의 공통적인 테스트 케이스를 추가하여 정량적인 효과성에 대한 연구를 진행 할 것이다.

참고문헌

[1] Fagan, M. E., “Advannces in Software Inspections”, IEEE Transactions in Software Engineering, Vol. 12, No. 7, 1986, pp. 744~751.

[2] Fagan, M. E., “Design and Code Inspection to Reduce Errors in Programming Development”, IBM Systems, Vol. 15, No. 3, 1976.

[3] Tom Gilb, Dorothy Graham, Susannah Finzi(ed.), Software Inspection, Addison-Wesley, 1993.

[4] 이숙영, “S/W Inspection 효과 분석 및 향상 방안”, 2003, 서강대학교 석사학위 논문

[5] CMU/SEI, “The Capability Maturity Model: Guides for Improving the Software Process”, Addison Wesley, 1994.

[6] Java Code Conversions, 1997, Sun Microsystems.

[7] 정현석, “소프트웨어의 품질개선을 위한 사례 연구”, 2003, 정보처리학회논문