DOI : 10.5392/JKCA.2011.11.7.080

소프트웨어 품질 개선을 위한 실증 연구 : A社 정보시스템 고도화 프로젝트 사례

A Case Study on the Software Quality Improvement : Information Systems Advancement Project of A Company

김재생*, 최상균*, 김경훈**, 경태원***
김포대학*, 경희대학교**, R&D특허센터***

Jae-Saeng Kim(jskim@kimpo.ac.kr)*, Sang-Gyun Choi(skchoi@kimpo.ac.kr)*,
Kyung-Hun Kim(magiclamp@khu.ac.kr)**, Tae-Won Kyung(twkyung@rndip.re.kr)***

요약

눈에 보이지 않는 소프트웨어의 품질을 측정하고 관리한다는 것은 쉬운 일이 아니다. 하지만 소프트웨어 개발 프로젝트가 대형화, 복잡화됨에 따라 그 중요성을 간과할 수 없다. 본 연구에서는 소프트웨어 품질 개선을 위한 방법들의 특징을 비교·분석하였다. 그리고 코드 인스펙션(Code Inspection)을 위한 규칙들을 수립하고 실제 프로젝트에 적용함으로써 그 효율성과 효과성을 검증하였다. 본 연구를 통해 다음과 같은 결과를 얻었다. 첫째, 소프트웨어 검토 방법들에 대한 이론적 내용을 정리한 점이다. 둘째, 코드 인스펙션을 위한 규칙을 수립하고 사례를 통해 성과를 증명하였다. 셋째, 코드 인스펙션을 통해 정량적 데이터 관리를 수행하였다.

■ 중심어 : | 인스펙션 | 코드 인스펙션 | 소프트웨어 품질 |

Abstract

It is no simple matter to measure and manage the quality of Software which is invisible. However the importance cannot be ignored as the software development project is getting bigger and complex.

This study analyses and compares the methods of software quality improvement. Further it formulates rules for Code Inspection and verifies the efficiency and the effectiveness by applying to practical project. The study get the following results. First, the study organizes theoretical content about the methods of Software review. Second, the study formulates rules for Code inspection and demonstrates the result through the case study. Third, the study performs the quantitative dater management through code inspection.

■ keyword : | Inspection | Code Inspection | SW Quality |

I. 서 론

소프트웨어 개발에서 품질은 중요한 문제이다. 만들어진 소프트웨어의 품질에 따라 프로젝트의 성패가 좌우하게 된다. 결함의 존재 여부나 그 양은 품질에 많은 영향을 미치게 된다. 이때 결함을 발견하고 수정하는

* 본 연구는 김포대학의 연구과제로 수행되었습니다.
접수번호 : #110512-001
접수일자 : 2011년 05월 12일
심사완료일 : 2011년 05월 25일
교신저자 : 경태원, e-mail : twkyung@rndip.re.kr

작업에는 상당한 시간과 비용이 소요된다. 특히 테스트나 유지보수 단계에서 결함을 검출하고 수정하는 비용과 시간은 개발 단계 초기에 수행하는 것보다 10~100배에 이르게 된다[1]. 이러한 이유 때문에 결함을 조기에 발견하고자 하는 시도가 끊이지 않았다. 이러한 결함 발견을 위한 대표적인 방법 중 하나가 인스펙션(Inspection)이다. 인스펙션은 1976년 Michael Fagan에 의해 제안되었으며[2], 인스펙션을 통해 모든 결함의 60~90% 정도를 찾을 수 있고, 인스펙션으로부터 피드백을 받음으로써 프로그래머는 같은 실수를 피할 수 있다고 주장하였다[3]. 따라서 본 연구에서는 소프트웨어의 완성도와 품질을 높이기 위한 방법들 중 하나인 인스펙션과 기존의 검토 기법들을 정리 하고자 한다. 그리고 코드 인스펙션(Code Inspection)을 위한 규칙들을 수립하고 실제 프로젝트에 적용함으로써 그 효율성과 효과성을 검증하고자 한다. 본 연구에서는 A社의 정보시스템 고도화 프로젝트를 대상으로 코드 인스펙션의 효율적 수행을 위한 규칙들을 수립하고 적용하였다.

II. 소프트웨어 결함 검출 기법

1. 테스트(Test)

IEEE에서는 테스트를 “시스템이 특정 요구사항을 만족하는지 검증하고 예상했던 결과와 실제 결과의 차이를 수동 또는 자동화된 방식을 통해 식별하는 실험 또는 평가하는 프로세스이다” 라고 정의하였다[4].

테스트는 일반적으로 세 가지 단계로 나누어진다. 즉, 단위 테스트(Unit test), 통합 테스트(Integration test), 인수 테스트(Acceptance test) 이다. 단위 테스트는 대부분 모듈을 구현한 프로그래머가 실시한다. 단위 테스트의 주요 목적은 모듈을 정확하게 구현하였는가, 예정한 기능을 제대로 발휘하는가를 점검한다. 통합 테스트는 전체 시스템을 이루는 모듈을 모아 통합적으로 시험하는 것을 말한다. 시스템이 요구된 기능을 제대로 수행하는가를 점검하고 모듈 사이의 인터페이스를 시험하는 것이 주목적이다. 인수 테스트는 완성된 제품에 대한 시험이다. 인수 테스트의 목적은 시스템이 사용될 준비가 다 되었다고 드러내 보이는 것이다[5].

2. 동료검사(Peer Review)

동료검토는 일반적으로 형식 없이 검토가 필요할 때마다 동료들이 작업 산출물을 검토하는 방법을 의미한다. 정해진 검토 인원이나 제한시간, 검토리더는 없으며 동료와 의견을 나누듯이 검토를 진행한다. 동료검토의 장점은 자유로운 의견을 통해 다양한 아이디어를 얻을 수 있고, 검토 과정을 통하여 지식이 동료들에게 전파되는 것이다. 하지만 검토 리더가 정해져 있지 않아 정상적인 검토가 진행되지 않거나 일반 회의처럼 다수의 의견이나 힘 있는 자의 의견에 전체의 흐름이 동조될 수 있다는 단점이 있다[6].

3. 워크스루(Walkthrough)

Fewster and Graham는 워크스루를 “가상의 입력에 대하여 원시코드의 수행을 문장 하나씩 짚어보는 작업”으로 정의하였다[7]. 또한 “시스템의 형태나 프로그램 개발 사항에 대해 개발 동료들로 하여금 조기에 오류를 확인할 수 있도록 하는 검토회의”라고 정의하고 있다. 워크스루는 일반적으로 프로그램의 소스코드에 대해 실시한다. 검토자들은 소스코드를 한 줄씩 읽어나가면서 발견한 이슈를 제기한다. 워크스루는 데이터 정의부분, 매뉴얼, 명세 등 프로그램이 아닌 부분도 검토 대상이 된다.

III. 코드 인스펙션(Code Inspection)

1. 인스펙션(Inspection)

ISO 8402:1995에서는 인스펙션(Inspection)을 “표준이나 명세서에 대한 편차와 에러를 포함한 결함을 발견하고 식별하기 위해 작업 산출물에 대해 수행하는 검사”라고 정의하고 있다[8]. 인스펙션은 소프트웨어의 품질과 생산성을 높일 수 있는 방법으로 1976년 Michael Fagan에 의해 제안되었다[2]. 현재까지 품질 개선과 비용 절감을 위한 대표적인 기법 중 하나로 사용되고 있다. Fagan 인스펙션은 여러 사람이 모여 소프

트웨어에 관련된 문서를 체계적으로 검사하는 방법으로, 이 방법의 효용성은 이미 여러 연구와 적용에 의해서 입증되었다[9].

2. Fagan의 Inspection 프로세스

Formal Inspection은 가장 정형화된 검토로서 1976년 Fagan에 의해 최초로 제안되었다. 처음에는 코드(Code) 검토에만 초점을 두었으나 점차 요구사항분석과 설계 단계 등 다른 산출물에도 적용하여 품질과 생산성의 개선에 큰 역할을 하고 있다.

인스펙션은 일반적으로 계획(Planning), 교육(Training), 준비(Preparation), 검사(Examination), 재작업(Rework), 해결과정확인(Follow-up)의 절차로 이루어진다. 검사 시 유의해야 할 사항은 인스펙션의 목적은 결함을 찾는 것이지 그 자리에서 결함의 원인이나 수정방법을 찾느라 시간을 낭비하지 않는 것이다. 또한 작업을 수행한 사람에게 불이익을 주어서는 안 된다.

코드검사의 필요성을 정리하면 다음과 같다.

- 개발자의 디버깅에는 한계가 있음
- 소프트웨어 개발 시 전사적인 품질 표준 필요
- 프로젝트초기에 버그(Bug) 탐지
- 어플리케이션의 안정성 및 보안성 향상
- 빠르고 적은 비용으로 개발 프로젝트 단축
- 소프트웨어 유지보수 비용 절감 및 품질관리에 용이

3. Code Inspection의 효과

Fagan의 연구뿐만 아니라 Olson Timothy의 연구에 의하면 Inspection을 수행함으로써 생산성이 2배로 증가하고, Inspection이 테스팅 단계 이전의 소프트웨어 오류의 80~90%까지 검출, 10~25%의 일정 단축, 산출물의 결함은 KSLOC당 0.01개의 수준으로 낮아졌다고 발표하였다[10].

또한 Heiser는 소프트웨어 개발 초기에 코드검사를 수행했을 경우 60~85%의 코딩 결함을 제거할 수 있고[11], Tom Glib & Dorothy Graham은 테스팅 이전에 50~90% 가량의 결함을 제거함으로써, 개발공정의 10~30%, 테스팅 비용 및 공정의 5~10배, 그리고 유지보수 비용의 2/3를 절감할 수 있다고 하였다[12].

IV. 코드 인스펙션 규칙 수립

본 절에서는 A社의 정보시스템 고도화 프로젝트를 대상으로 코드 인스펙션을 위한 규칙들을 수립하고자 한다. 코드 인스펙션 규칙 수립을 위해 참고한 자료의 구체적 명칭과 출처는 저작권 및 기관의 보안 유지를 위해 밝히지 않는다. 또한 수립된 코드 인스펙션 규칙은 본 연구에 초점이 맞춰진 것으로써 일반화시키기 위해서는 약간의 조정이 필요함을 밝힌다.

1. 코드 인스펙션 프로세스

본 연구에서는 코드 인스펙션 프로세스를 바탕으로 코드 인스펙션 과정을 수립하였다. [그림 1]은 코드 인스펙션 수행을 위한 과정을 보여주고 있다. 특히, 코드 인스펙션을 위한 규칙을 수립함으로써 인스펙션 활동의 효율과 성과를 높이고자 한다.

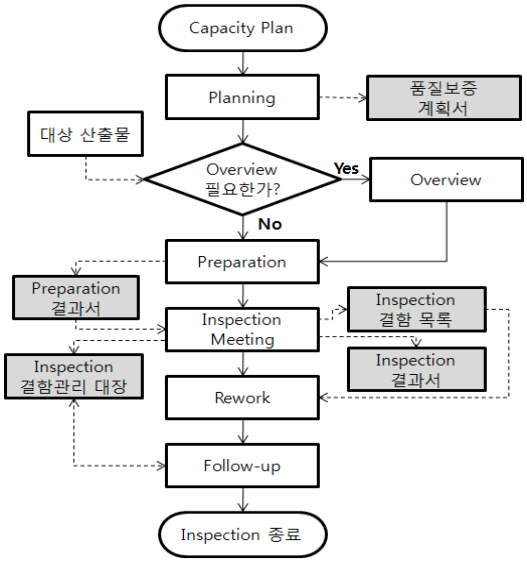


그림 1. 코드 인스펙션 프로세스

코드 인스펙션 규칙 수립을 위해 다음과 같은 자료를 참고하였다.

- 발주사 프로그램 작성가이드
- 컨설팅사 개발환경 및 코딩가이드
- 개발사 품질보증팀 기준

표 1. 코드 인스펙션 적용 규칙

검사 영역	대분류	중분류	설명
File	Naming Convention	Import Declaration	import 문은 정해진 형식으로 정의해야 한다.
		File Comment	파일 주석 여부 점검
		Class Declaration	한 파일 내 inner class 존재여부 검사
		Line Length	라인길이(180) 초과 여부
Class	Organization	Failure Definition Variable	미사용 클래스 변수
		Final Variables	final 변수는 메모리의 효율적 사용을 위해 static final 로 선언하여야 함
		Constant Interface	상수만 정의하는 인터페이스 금지
		Abstract Class without Abstract Method	추상메서드가 없는 추상클래스 금지
		Unnecessary Constructors	불필요한 public 생성자 금지
		Potential Local Variable of Private Variable	특정 함수에서만 사용하는 함수는 지역변수로 선언할것
		Unsafe Singleton Pattern	불완전한 싱글톤 패턴 사용금지
		Class Name Length	클래스 Naming 길이 위배
		Constant Variable Naming	상수 명명 규칙 위배
	Statement	Synchronized Statement	"Synchronized" 키워드를 점검한다.
Method	Organization	Method Comment	함수 주석 누락
		Parameter Counts	primitive tyep의 파라미터의 개수 제한(6개 이상 금지)
		Failure Definition Method	미사용 private 함수
		Never thrown Exception for the throws clause	throws가 정의된 함수에서 본문을 모두 try_catch로 감싼 경우
		Reassigned Parameter	파라미터에 값 할당 금지
		Method Naming	함수 명명규칙 위배(함수명 끝자리에 숫자 사용금지)
	Statement	Switch Default Clause	switch문은 반드시 default문으로 종료
		Empty Block Body	빈 제어문 금지
		Empty try/catch/finally Block	빈 try/catch/finally문 금지
		Static Variable Access	정적변수의 접근 방식점검(불필요한 객체 생성금지)
		Used Static Variables	final이 아닌 정적변수의 사용 점검
		Static Method Invocation	정적함수의 접근 방식 점검(불필요한 객체 생성금지)
		Method Invocation in Loop Condition	순환식의 조건문에서 함수의 호출
		synchronized method call in the loops	순환문에서 Synchronized문 호출문장 점검
		Infinite Loop Statement	무한 루프 서술문 금지
		Explicitly Created Object in Loops	순환문에서 명시적인 객체 선언금지
		ExplicitlyCreatedNullPointerException	Null Pointer Exception 을 throw 하는 경우 금지
		SimpleDateFormat with argument Locale	SimpleDateFormat클래스는 생성시 Locale인자를 사용해야 한다.
		equals() called with null argument	Equals 메소드 호출시 null 인자를 사용하지 말아야 한다.
		String Instantiation	String 자료형은 new연산자로 생성하지 말아야 한다.
		String.valueOf()	String 클래스의 valueOf()메소드를 불필요하게 사용하지 말아야 한다.
		String.toString()	String 클래스의 toString()메소드를 사용하지 말아야 한다.
		String.trim().length()	빈 문자열 점검을 위해 String 객체의 trim()을 통한 length()메소드를 사용하지 말아야한다.(existsCharacter(검증하고자 하는 문자)형태로 사용할 것)
		StringBuffer Instantiation	StringBuffer의 생성자내에 하나의 상수만을 사용해야 한다.
		Assignment in Condition Clause	조건절에서는 대입연산자를 사용하지 말아야 한다.
		ExplicitlyCreatedRawExceptions	Raw Exception을 명시적으로 생성하여 던지지(throw) 말아야 한다.
		Unnecessary Boolean Expression	boolean 변수와 상수간 비교는 하지 말아야 한다.
		Unnecessary Created Primitives Constructor	기본자료형을문자열로전환시불필요한임시객체생성을하지말아야한다

표 1. 코드 인스펙션 적용 규칙 (계속)

검사 영역	대분류	중분류	설명
Method	Statement	Unnecessarily Used Local Variable within Return Statement	Return시 불필요한 로컬변수를 사용하지 말아야 한다.
		Unnecessarily Used Parentheses within Return Statement	Return시 불필요하게 수식에 괄호를 사용하지 말아야 한다.
		Unnecessarily created Objects with Primitive Types	기본형 자료형에 대한 변환시 불필요한 wrapper 객체를 생성하지 말아야 한다.
		Array Copy in Loop	배열 복사 시 시스템 함수 사용
		Local Variable Naming	지역변수 명명규칙 준수 여부
		Failure Definition Local Variables	미사용지역변수 금지
		Collection Declarations	Vectror, Hashtable 사용금지
	Local Variable	Debug Statement	System.out.println, e.printStackTrace 사용 금지
		I/Orelease	IO의 close 여부 검사
		JMS release	JMS 의 close 여부 검사
	System Statement	Class.forName() Invocation	Class.forName() 함수 호출금지
		System.gc() Invocation	System.gc() 함수 호출금지
		System.exit() Invocation	System.exit() 함수 호출금지
		Runtime.runFinalization() Invocation	Runtime.runFinalization() 함수 호출금지
		Runtime.runFinalizersOnExit() Invocation	Runtime.runFinalizersOnExit() 함수 호출금지
		Runtime.exec() Invocation	Runtime.exec() Invocation
		Thread Methods Invocation	Thread 관련 함수 호출 금지
		Don't use the ThreadGroup	Thread 함수 중 JDK API에서 권장하지 않는 함수 호출 금지

프로젝트에 참여한 이해당사자들과 세 개 기관의 소프트웨어 개발 가이드라인을 바탕으로 본 연구에 적용하기 위한 코드 인스펙션 규칙들을 수립하였다. 또한 수립된 코드 인스펙션 규칙들은 유사한 성격을 갖는 항목별로 분류하여 3개 영역, 10개 대분류, 60개 중분류로 정리하였다.

[표 1]은 코드 인스펙션 규칙들과 각 규칙들에 대한 내용을 정리한 것이다. 적용 규칙에는 파일(File), 클래스(Class), 메소드(Method)의 3개 검사영역, Naming Convention외 6개 대분류, Import Declaration 외 59개 중분류로 구성하였다.

V. 코드 인스펙션 적용 결과

1. 영역별 코드 인스펙션 분석

[표 2]는 코드 인스펙션 대상인 파일, 클래스, 메소드별로 검사 대상, 검사 규칙, 검사 결과를 정리하였다.

표 2. 코드 인스펙션 대상

Scope	검사 대상	검사 규칙	검사 결과 (수정 수)
File	308	4	393
Class	308	10	12
Method	3,252	46	890

[그림 2][는 File Audit 결과이다. 308개 파일을 대상으로 코드 인스펙션 결과 파일에 주석을 처리하지 않은 경우인 'File Comment' 위반이 307건으로 전체 78%, 'Import Declaration' 위반 79건(20%), 'Line Length' 위반 7건(2%) 등 393건의 수정사항이 발생되었다.

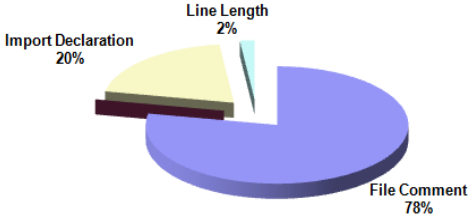


그림 2. File Audit

Class scope의 경우, 308 클래스를 대상으로 12개 규칙을 적용하여 조사한 결과 'Abstract Class without Abstract Method' 위반 9건, 'Final Variable' 위반 3건 등 총 12건의 수정사항이 발생되었다.

Abstract Class without Abstract Method 항목의 경우 추상 메서드가 없는 추상 클래스를 사용한 경우 이고, Final Variables 항목의 경우 메모리의 효율적 사용을 위해 static final로 선언하지 않은 경우이다.

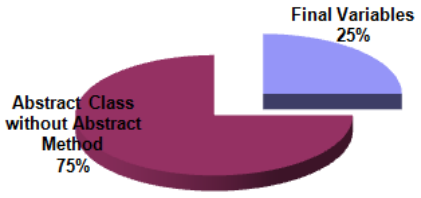


그림 3. Class Audit

[그림 3]은 Class Audit 결과이다. 전체 308개의 클래스를 검사한 결과 추상 메서드 없이 추상클래스를 사용한 경우가 75%인 231건이 검출되었다. 그리고 static final로 선언하지 않은 변수가 25%인 77건이 검출되었다.

Method scope의 경우 3,252개를 대상으로 46개 규칙을 적용하여 조사한 결과 890개의 수정 사항이 발견되었다. 항목별로 살펴보면, 'Method Naming' 규칙 위반이 371건이 발견되었으며, 'Method Comment' 위반이 328건 등 890건의 위반사항이 발견되었다. Method Naming은 함수명 끝자리에 숫자를 사용한다든지 함수명명규칙을 위반한 경우이며, Method Comment는 함수에 주석을 달지 않은 사항이다. 규칙별 위반 현황을 보면 그림4와 같다.

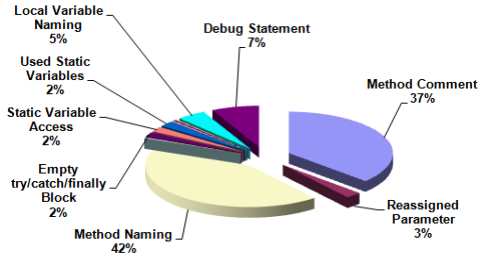
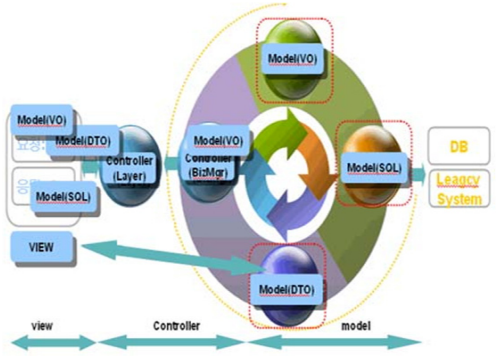


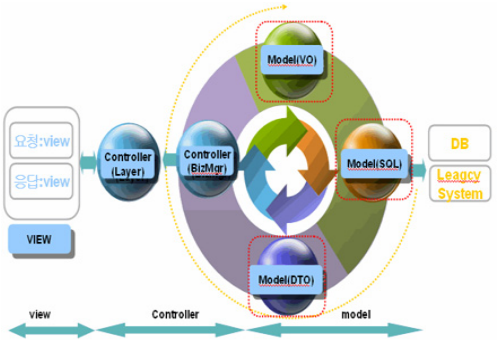
그림 4. Method audit

2. MVC 준수여부 검증 결과

MVC(Model-View-Controller) 패턴을 준수하도록 유도함으로써 향후 유지 보수 시 가독성 향상 및 모듈화가 진전될 것으로 분석되었다.



(a) 수정 전



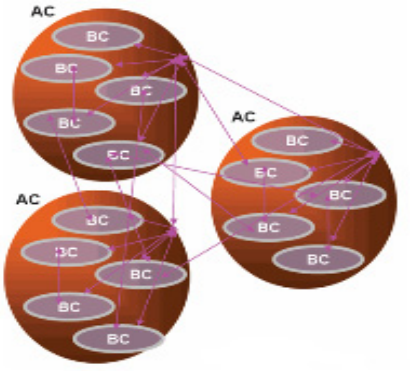
(b) 수정 후

그림 5. MVC 준수여부 검증 결과

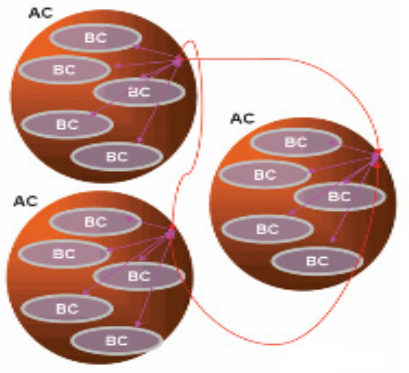
[그림 5]에서는 MVC 준수여부에 대한 수정 전과 수정 후의 결과를 도식화 하였다.

3. Component 개발 준수 여부 검증

Component의 인터페이스규칙이 지켜지도록 검증함으로써 향후 컴포넌트 접근에 대한 일괄 정책 수립이 가능하며 컴포넌트의 독립성이 향상될 것으로 분석되었다.



(a) 가이드 전



(b) 가이드 후

그림 6. Component 개발 준수 여부 결과

[그림 6]에서는 Component 개발 준수에 대한 가이드 전후 모습을 도식화 하였다.

Ⅵ. 결론 및 향후 연구방향

코드 인스펙션은 그 효과가 널리 알려져 있음에도 불구하고 국내에서 체계적으로 수행된 사례는 많지 않다. 따라서 본 연구에서는 코드 인스펙션과 기존의 검토방법들을 정리함으로써 소프트웨어 개발에 관련된 이해당사들에게 코드 인스펙션의 이론적 배경을 제공하고, 코드 인스펙션의 중요성과 필요성을 전달하고자 하였다. 또한 코드 인스펙션을 위한 규칙을 수립하고 실제 프로젝트에 적용함으로써 그 성과를 분석하였다.

A社의 정보시스템 고도화 프로젝트에 코드 인스펙션을 적용함으로써 이해당사자들(경영층, 품질 관리자, 개발자)은 다음과 같은 효과를 기대할 것으로 판단된다. 경영층에서는 소프트웨어 개발 비용 및 시간 절감, 고객 신뢰도 향상, 기업 이미지 향상 등을 기대할 것으로 판단된다. 품질 관리자는 정량적인 소스 품질 관리 확보, 실시간 소스 품질 관리 프로세스 확립, 코드 결함 및 생산성 정보 확보, 아웃소싱 및 자체 개발 어플리케이션에 대한 일관된 코드 검사가 가능할 것으로 판단된다. 그리고 개발자에게는 코딩 표준 인식 고취, 일관된 코딩 표준에 의한 코드 개발, 고품질·고성능 어플리케이션(Web or C/S) 개발 등을 기대할 것으로 판단된다.

본 연구는 다음과 같은 측면에서 의의를 찾을 수 있다.

첫째, 소프트웨어 검토 방법들에 대한 이론적 내용을 정리한 점이다.

코드 인스펙션은 소프트웨어의 품질을 높이기 위한 검토 방법들 중 하나로써 그 정의와 목적, 그리고 수행방법에 있어 기존 방법들과 분명한 차이가 있다. 하지만 대부분의 개발자나 관리자들이 기존 방법인 동료검토, 워크스루, 그리고 테스트와 혼동을 하거나 명확히 구분을 하지 못하고 있다. 따라서 본 연구에서는 문헌자료를 바탕으로 코드 인스펙션과 기존 검토 방법들을 정리하였다.

둘째, 코드 인스펙션을 위한 규칙을 수립하고 사례를 통해 성과를 증명하였다.

본 연구에서는 A社의 정보시스템 고도화 프로젝트를 대상으로 코드 인스펙션을 위한 규칙들을 수립하였다. 물론 특정 프로젝트를 대상으로 수립된 규칙이기 때문에 일반화시키거나 표준화시키기에는 다소 무리가 있다. 하지만 본 연구에서 수립한 규칙들은 소프트웨어 개발 시 암묵적으로 지켜온 방법들을 기초로 수립된 것으로써 충분한 가치가 있다고 판단된다.

셋째, 코드 인스펙션을 통해 정량적 데이터 관리를 수행하였다.

코드 인스펙션 대상을 각 업무별, 기능별, 세부 항목

에 따라 정량적으로 파악하고 결함의 규모와 빈도수를 분석하였다. 코드 인스펙션 대상을 영역별, 기능별로 적용 규칙을 수립하여 결함을 조사하였다. 그리고 결함율 0%를 목표로 코드 인스펙션과 재작업 과정을 반복함으로써 코드의 완성도를 높이고 소프트웨어 개발의 효율과 품질을 높이기 위해 노력하였다.

이처럼 본 연구에서는 코드 인스펙션 규칙들을 수립하고, A社의 정보시스템 고도화 프로젝트에 적용하여 그 성과를 분석하였다. 이러한 노력은 향후 소프트웨어의 원활한 개발과 유지보수 그리고 프로젝트의 품질을 높이는데 긍정적인 영향을 줄 것으로 기대된다. 하지만 코드 인스펙션은 기존에 수행하던 활동이 아니며 초기에 개발자들이 평균적으로 더 많은 작업시간을 필요로 하기 때문에 코드 인스펙션을 공식적인 개발활동의 일부로 인정받을 수 있는 환경조성이 매우 중요하다. 이를 위해서는 개발자뿐만 아니라 경영진 그리고 고객의 합의와 참여를 이끌어 내기 위한 대책 마련이 필요하다. 또한 소프트웨어 코드 인스펙션의 지속적인 적용과 연구를 바탕으로 체계적인 소프트웨어 품질향상의 기반을 구축해야 할 것이다.

참 고 문 헌

[1] A. Davis, *Software Requirements: Analysis and Specification*, Prentice-Hall, p.20, 1990.

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

[3] M. E. Fagan, "Advances in Software Inspections," IEEE Transactions in Software Engineering, Vol.12, No.7, pp.744-751, 1986.

[4] IEEE, *IEEE Standard Glossary of Software Engineering Terms*, IEEE Society Press, 1983.

[5] 최은만, *소프트웨어 공학*, 정익사, 2006.

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

[7] M. Fewster and D. Graham, *Software Test Automation, Effective Use of Test Execution Tools*, Addison-Wesley, 1999.

[8] ISO 8402:1995, *Quality Management and Quality Assurance. Vocabulary*, 1995.

[9] J. C. Kelly, J. S. Sherif, and J. Hops, "An Analysis of Defect Densities Found During. Software Inspections," Journal of Systems Software, Vol.17, 1992.

[10] Olson Timothy, *World-Class Software Inspection*, SEI 1996 SEPG Conference Proceeding, 1996.

[11] J. E. Heiser, J. E., *Overview of Software Testing, IEEE Transactions On Software Engineering*, 1997.

[12] Tom Glib & Dorothy Graham, *Software Inspection*, Addison-Wesley Professional, 1994.

저 자 소 개

김 재 생(Jae-Saeng Kim) 종신회원

- 1988년 2월 : 경희대학교 컴퓨터공학과(공학사)
- 1990년 8월 : 경희대학원 컴퓨터공학과(공학석사)
- 1997년 8월 : 경희대학원 컴퓨터공학과(공학박사)
- 1998년 3월~현재 : 김포대학 이-비즈니스과 교수

<관심분야> : SW공학, 컴포넌트 평가, 웹기반 SW

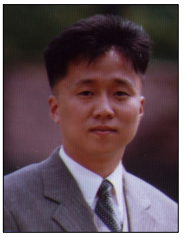
최 상 균(Sang-Gyun Choi) 정회원

- 1986년 2월 : 한남대학교 컴퓨터공학과 졸업(공학사)
- 1993년 8월 : 서강대학교 대학원 정보처리학과 졸업(이학석사)
- 2005년 8월 : 경희대학교 대학원 컴퓨터공학과 졸업(공학박사)

- 1995년 7월 : 전자계산조직응용 기술사
- 1986년 6월 ~ 1998년 2월 : 한국생산기술연구원 선임연구원
- 1998년 3월 ~ 현재 : 김포대학 e-비즈니스과 부교수

<관심분야> : 컴포넌트 소프트웨어, 정보시스템 품질, 정보시스템 감리

김 경 훈(Kyung-Hun Kim) 정회원



- 2000년 2월 : 삼육대학교 컴퓨터과학과(이학사)
- 2002년 2월 : 경희대학교 전자계산공학과(공학석사)
- 2004년 2월 : 경희대학교 전자계산학과 박사 수료

<관심분야> : 형상관리, 웹서비스, 의료시스템, 컨텐츠

경 태 원(Tae-Won Kyung) 정회원

- 1998년 2월 : 호원대학교 전자계산학과(공학사)
- 2002년 2월 : 경희대학교 전자계산공학과(공학석사)
- 2008년 8월 : 경희대학교 산업공학과(공학박사)
- 2008년 8월 ~ 2010년 6월 : 한국생산기술연구원
- 2010년 7월 ~ 현재 : R&D특허센터

<관심분야> : IT 프로젝트 관리/기획, SW공학