EJB 컴포넌트의 인스펙션 프로세스 모델

남준기° 한혁수
상명대학교 일반대학원 컴퓨터 과학과
{namfund° , hshan}@sangmyung.ac.kr

Component Inspection process for EJB

Jun-Ki Nam° Hyuk Soo Han
Computer Science Dept, Sangmyung Univ.

요 약

인스펙션은 소프트웨어 개발 프로세스에서 작성된 산출물에 대한 품질 보증 활동의 하나이다. 산출물에서 결함를 찾을 수 있는 가장 정형적, 효율적, 경제적인 방법이 인스펙션이라고 알려져 있다[1]. 인스펙션은 테스트 전에 결함을 검출하여 소프트웨어의 품질을 높이고, 테스트 후에 발견되는 결함을 줄임으로써 유지보수에 드는 시간과 비용을 절감시킨다. 본 논문에서는 인스펙션 중에서도 코드에 관련된 인스펙션을 연구하고자 한다. 현재의 개발 패러다임은 CBD(Component Based Development)로 가고 있다. 소프트웨어 개발이 많아지고 규모가 커짐에 따라 재사용성이 뛰어난 컴포넌트로 개발들이 이루어지고 수행되고 있다. 컴포넌트를 개발할 때에도 품질은 중요한 문제이기 때문에 연구가 필요하다. 기존의 패러다임에서는 품질을 위한 인스펙션에 관한 많은 연구들이 있었고 진행되었다. 하지만 CBD에서의 품질을 위한 인스펙션에 관한 연구는 많이 이루어지지 않고 있다. 본 논문에서는 이러한 문제점을 알아보고 EJB(Enterprise Java Beans) 컴포넌트에 대한 코드 인스펙션을 통해 결함을 줄이고 품질을 향상시키기 위한 컴포넌트 인스펙션 프로세스(Component Inspection Process)를 제시한다. 프로세스의 각 단계에서는 UML 다이어그램, EJB의 콜백 메소드 흐름 다이어그램, 그리고 체크리스트를 사용한다. CIP 사용함으로써 CBD에서의 많은 결함을 줄이고 품질을 향상시킬 수 있게 될 것이다.

1. 서 론

소프트웨어 개발에서 품질은 중요한 문제이다. 만들어진 소프트웨어의 품질이 좋으냐 나쁘냐는 곧 그 프로젝트의 성패를 좌우하게 된다. 결함의 존재 여부나 그 양은 품질에 많은 영향을 미치게 된다. 이 때 결함을 발견하고 수정하는 작업에는 많은 시간과 비용이 소요된다. 특히 테스트나 유지보수 단계에서 비용과 시간은 개발 단계 초기에 수행하는 것보다 10배에서 100배에 이르게 된다[2]. 따라서 초기에 결함을 찾는 것은 중요한 문제이다. 인스펙션은 개발 프로세스에서 작성된 산출물에 대해 초기에 결함을 찾는 방법이다. Fagan은 인스펙션을 통해 모든 결함의 60-90 퍼센트 정도를 찾을 수 있고 인스펙션으로부터 피드백을 받음으로써 프로그래머는 같은 실수를 피할 수 있다고 주장하였다[3]. Rusell도 보고서에서 인스펙션을 통해 유지보수에 드는 시간을 많이 줄일 수 있다고 보고하였다[4]. 이처럼 프로젝트에서 인스펙션에 많은 노력을 기울인다면 시간과 비용을 줄이면서 많은 결함을 찾을 수 있다. 현재 소프트웨어 개발 패러다임은 CBD(Computer Based Development)가 주를 이루고 있다. 소프트웨어의 개발이 많아지고 규모가 커짐에 따라 재사용성이 뛰어난 컴포넌트를 개발함으로써 이 문제를 해결하고 있다. 컴포넌트에서도 품질은 중요한 문제이고 결함을 찾고 수정하는 작업에는 많은 시간과 비용이 소요된다. 또한 하나의 기능이나 성능을 구현해 놓은 컴포넌트에서의 논리적인 로직은 인스펙션에 있어서 중요하다. 기존의 인스펙션과 관련된 많은 연구들이 진행되었고, 수행되었는데 이런 코드 인스펙션 방법들은 EJB 컴포넌트에 적용시키기 어려운 점들이 있으며 현재의 CBD에 적합한 인스펙션 방법들과 관련된 연구도 많이 이루어지지 않고 있다. 따라서 이러한 문제점들을 해결하기 위해서는 컴포넌트의 특징에 맞게 새로운 코드 인스펙션 프로세스가 제시되어야 할 것이다. 현재 많은 컴포넌트 기반의 프로젝트들이 수행되고 있는데 그 구현 기술 중에서도 EJB는 Sun 사에서 J2EE에서의 분산 아키텍처를 기반으로 한 서버 컴포넌트 모델로 제시된 스펙이다. 많은 회사와 기업들이 EJB 스펙에 맞춰 프로젝트를 수행하고 있다. 본 논문은 EJB 컴포넌트의 코드 인스펙션을 연구함으로써 결함을 찾기 위한 컴포넌트 인스펙션 프로세스(CIP)를 제안하고자 한다. CIP에서는 EJB 컴포넌트의 코드 인스펙션을 위한 몇 가지 단계를 제시한다. 단계는 우선 시작단계, 유스케이스 단계, 시퀀스 다이어그램 단계, 클래스 다이어그램 단계, 콜백 메소드 흐름 다이어그램 단계, 체크리스트 단계를 거친다. 또한 코드에서 발견된 것을 수정하기 위한 재작업 단계, 완료 단계를 수행함으로써 프로세스는 종료된다. CIP는 첫 번째 컴파일 전에 수행될 것이며, 위의 프로세스를 통해 해당 컴포넌트의 결함을 쉽고 정확하게 찾게 될 것이다.

2. 관련 연구

2.1 A. Dunsmore의 객체지향 인스펙션

A. Dunsmore는 자신의 논문에서 객체지향 코드를 인스펙션 하기 위한 방법으로 체크리스트와 유스케이스, 시퀀스 다이어그램을 제시하였다. 체크리스트는 코드 인스펙션에서 이미 제시가 되었으며 가장 많이 사용하는 방법 중 하나이다. 체크리스트는 기존의 경험을 바탕으로 질문을 만들어야 하는 어려움이 있다. 또한 질문들이 너무 일반적이고 각 개발환경에 맞지 않으며 질문이 명확하지 않다는 문제점이 있다. 그리고 현재 EJB 컴포넌트에 표준화된 체크리스트가 없다는 단점이 있다. 유스케이스, 시퀀스 다이어그램 방법은 객체지향 코드에서의 시나리오를 기반으로 객체가 사용될 수 있는 모든 방법들이 올바르게 제공되는지와 동적인 관점, 논리적인 흐름을 제공하고 있는데 이러한 방법이 객체지향 시스템에서의 코드 인스펙션에 많은 도움을 제공한다[5]. 체크리스트와 유스케이스의 코드 인스펙션은 객체 지향 시스템에 좋은 방법이지만 EJB 컴포넌트의 코드 인스펙션에는 부족한 면이 존재하며 추가되어야 할 것들이 있다. 체크리스트는 EJB 컴포넌트에 맞게끔 수정되어야하며 적합한 결함 형태를 찾아서 제시해야 한다.

2.2 IDEA 3.0 에서의 EJB 인스펙션

JetBrains 사에서 만든 IDEA 3.0은 자바 프로그램을 개발하는 도구이다. 이 도구에서는 코드 인스펙션 자동화 기능이 포함되어져 있다. 인스펙션하는 내용은 기존의 체크리스트를 이용해서 검사해왔던 선언 시의 문제점과 클래스, 메소드, 변수, 생성자, 파라미터 등의 잘못된 사용이나 사용 범위와 같은 결함들을 자동으로 검사한다. 또한 EJB 컴포넌트를 개발했을 때 EJB 명세대로 개발이 되었는지를 검사한다. 하지만 IDEA 3.0에서 제공되는 인스펙션 방법은 EJB 컴포넌트의 시나리오 상의 결함이나 논리적인 흐름에 대한 결함을 찾아낼 수는 없다.

3. 컴포넌트 인스펙션 프로세스(Component Inspection Process)

관련연구에서 살펴 본 것처럼 기존의 인스펙션 방법들은 EJB 컴포넌트에 맞는 방법이 제시되거나 시나리오 상의 결함이나 논리적인 로직에 관련된 결함을 찾는 방법은 제시하지 못한다. 따라서 기존의 인스펙션에서 부족한 점을 보완할 수 있는 방법으로 CIP를 제시한다. CIP는 그림 1과 같이 UML 다이어그램, 콜백 메소드 흐름 다이어그램, 체크리스트의 세 가지 코드 인스펙션 방법으로 구성된다.

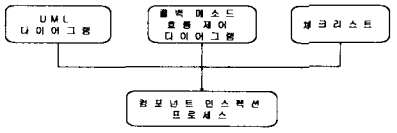

그림 1 컴포넌트 인스펙션 프로세스(CIP)의 구성

방법	범위
유스케이스	시나리오, 기능
시퀀스 다이어그램	각 메시지의 흐름
클래스 다이어그램	클래스간의 관계와 메소드, 변수
메소드 흐름 다이어그램	콜백 메소드와 비즈니스 메소드의 구현
체크리스트	네이밍(Naming),정의와 선언, 생성자, 반환형 등

표 1 각각의 방법들과 인스펙션 범위

CIP의 자세한 절차를 살펴보면 우선 유스케이스, 시퀀스, 클래스 단계를 통해 EJB 컴포넌트에 대한 동적인 관점과 논리적인 면, 그리고 시나리오에 대한 코드 인스펙션을 실시한다. 다음 단계에서는 EJB 컴포넌트의 콜백 메소드 흐름 다이어그램을 이용해서 메소드에 대한 코드 인스펙션을 수행한다. 마지막 단계에서는 체크리스트를 이용한 코드 인스펙션을 수행한다. 체크리스트는 EJB에 적합한 것으로 제공된다. 이 세 가지 방법대로 프로세스가 진행되면 결함을 검사하는 효율이 높아진다. 표 1에서는 각각의 방법과 인스펙션의 범위를 나타내고 있고 그림 2는 CIP의 각 단계와 흐름을 보여준다.

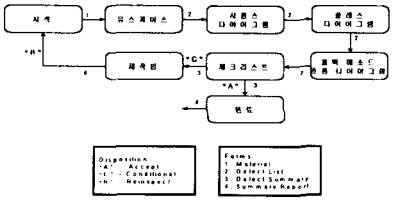

그림 2 컴포넌트 인스펙션 프로세스 (CIP)

3.1 시작 단계

시작 단계는 코드를 인스펙션하기 위한 준비단계이다. 이 단계에서는 인스펙션 하기 위한 자료들을 수집하고 그 자료들을 다음 단계로 전달한다. 전달해야 하는 자료에는 코드 문서, 요구사항 문서, 디자인 문서 등이 포함된다.

3.2 유스케이스 단계

이 단계는 유스케이스의 명세를 통해서 코드 인스펙션을 수행한다. 시나리오를 기반으로 각각의 객체가 그것이 사용될 수 있는 모든 가능한 방법에 올바르게 대응하고 있는지를 체크하게 된다. 이것은 체크리스트가 더 일반적인 문맥에서 클래스를 고려하는 것과는 다르게 객체가 사용될 수 있는 문맥에서 인스펙션을 수행하게 된다. 또한 유스케이스의 선입조건, 성공 실패조건, 예외 조건을 기반으로 시나리오를 살펴 볼 수 있다. 유스케이스는 다른 인스펙션 방법의 보완적인 방법으로 사용되고 있다. 다음의 표 2는 유스케이스 명세를 통해 은행 시스템의 자금이체 예를 보여주고 있다. 자금 이체가 요청되면 직원은 해당 정보를 시스템에 입력한다. 시스템은 직원이 입력한 정보에 대해 인증을 실시하고 자금 이체를 수행한다. 만약 해당 정보가 틀릴 경우는 예외가 발생하게 되어 그 결과를 직원에게 알려주게 된다. 그림 3은 유스케이스 명세를 통해 나타날 수 있는 흐름 다이어그램을 보여주고 있다. 이 구조도와 코드에서 볼 수 있는 구조도가 동일하면 결함은 존재하지 않는 것이고 그렇지 않을 경우 결함이 존재하는 것이다.

3.3 시퀀스 다이어그램 단계

시퀀스 다이어그램의 코드 인스펙션을 통해서 EJB 컴포넌트들 간의 메시지의 흐름을 살펴보고, 컴포넌트가 구현하고자 하는 해당 기능이 제대로 되어 있는지, 또한 시나리오 면에서 클래스의 메소드가 호출되는 것이 올바르고 일관성이 있는지를 검토할 수 있다.

엑터	직원
목적	자금이체가 수행되어야 한다.
선입조건	자금이체를 수행하기 위한 계좌가 있어야 한다.
성공조건	자금이체가 성공적으로 수행되어야 한다.
실패조건	없음
트리거	직원이 자금이체를 요청한다.
예외	계좌가 없거나 통장에 잔액이 부족한 경우에 발생한다.
단계	Flow of Events Basic Path 1. 유스케이스는 직원이 자금 이체를 선택했을 때 실행된다. 2. 직원은 계좌, 사용자 이름, 비밀번호, 금액을 입력한다. 3. 직원은 Submit을 선택한다. 4. 시스템은 해당 정보를 확인한다. 5. 시스템은 인출하려는 계좌에서 돈을 인출한다. 6. 시스템은 입금하려는 계좌에 돈을 입금한다. 7. 입금이 완료되면 직원에게 확인 정보가 보여지고, 유스케이스는 종료된다. Alternative Paths 1. 4에서 해당 정보가 잘못되면, 시스템은 직원에게 해당 정보를 다시 입력하라고 요청한다. 2. 5, 6의 실행 중에 이상이 있을 경우 수행은 롤백된다. 3. Submit을 하기 전에 직원은 유스케이스를 종료 할 수 있다.

표 2 자금이체 유스케이스 명세

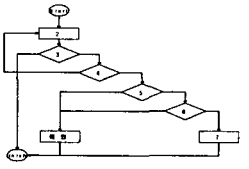

그림 3 유스케이스의 구조도

그림 4는 자금 이체에 대한 시퀀스 다이어그램이다. EJB 컴포넌트의 세션빈이 직원으로부터 요청을 받아 해당 엔티티빈을 수행시키는 흐름을 보여주고 있다. 이것을 통해 컴포넌트 코드의 메시지 호출의 적절성과 일관성을 검사 할 수 있다. 표 3은 시퀀스 다이어그램과 소스 코드와의 메시지를 통한 인스펙션 방법을 보여주고 있다. 디자인 단계에서 나온 시퀀스 다이어그램을 기반으로 만들어진 메시지 리스트는 메시지가 코드 상에 존재하는 지와 올바르게 사용되는지를 인스펙션하게 한다.

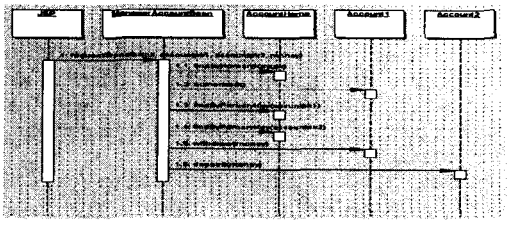

그림 4 자금이체를 위한 시퀀스 다이어그램

메시지 리스트

시퀀스 다이어그램		소스코드			확인 (OX)
메시지 번호	메시지 명	메시지 보내는 위치	메시지 받는 위치	설명	
1	requestTransfer	Trans.jsp	ManagerAccountBean.java	클라이언트가 자금이체를 요청	
1.1	findByPrimaryKey	ManagerAccountBean.java	AccountBean.java	자금이체 시 pk를 통해 해당 사용자 정보를 요청	
⋮	⋮	⋮	⋮	⋮	

표 3 시퀀스 다이어그램과 소스코드의 메시지 리스트 비교

3.4 클래스 다이어그램 단계

클래스 다이어그램 단계에서는 EJB의 컴포넌트들이 제대로 구현되어 있는지, 각 빈들의 비즈니스 메소드가 올바르게 구현되었는지를 코드 인스펙션한다. 클래스 다이어그램은 컴포넌트를 구성하는 해당 빈들 간의 관계와 빈 내부의 멤버들의 구현을 살펴봄으로서 불필요하거나

추가되어야 할 내용과 연관관계에서의 코드 결함을 찾는다. 그림 5는 클래스 다이어그램으로 자금 이체를 나타낸 것이다. 표 4와 5는 클래스다이어그램과 코드 상의 일치를 살펴봄으로써 인스펙션 하는 것을 보여준다. 이를 통해 EJB 컴포넌트의 전체적인 구성과 각 빈들의 올바른 사용여부, 그리고 상호 관계를 코드 인스펙션 할 수 있다.

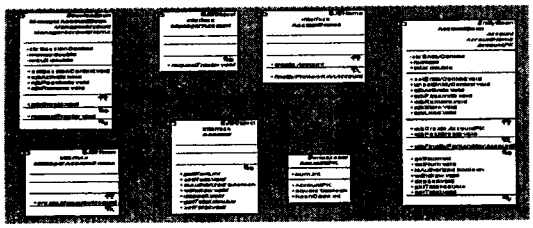

그림 5 자금이체 클래스 다이어그램

에소드 리스트

1	withdraw	void	String id, double money	자금이체에서 엔티티 빈을 호출하여 자금이체 할 금액을 이체하는 계좌에서 뺀다.	
2	deposit	void	String id, double money	자금이체에서 엔티티 빈을 호출하여 자금이체 할 금액을 이체 받는 계좌에 입금한다.	
⋮	⋮	⋮	⋮	⋮	⋮

표 4 클래스 다이어그램과 소스코드에서의 에소드 리스트

변수 리스트

1	money	전역	클래스 내	자금이체 시 자금이체 금액을 나타내기 위해서 사용된다.	
2	result	지역	authorize 에소드	자금이체 시 자금이체 인증을 위해 사용되는 변수이다.	
⋮	⋮	⋮	⋮	⋮	⋮

표 5 클래스 다이어그램과 소스코드에서의 변수 리스트

3.5 콜백 에소드 흐름 다이어그램 단계

현재 EJB 콜백 에소드의 흐름이나 관계에 대한 다이어그램은 제공되지 않고 있다. 그래서 에소드의 흐름에 관한 다이어그램을 제공함으로써 검사자가 이것과 코드를 비교함으로써 결함을 효율적으로 찾게 된다.

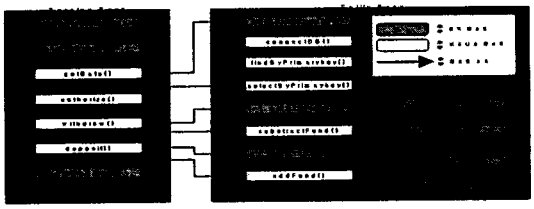

그림 6 콜백 에소드 흐름 다이어그램

EJB 컴포넌트는 하나의 프레임워크이다. 여기에서 프레임워크의 의미는, 정해놓은 틀 안에서 순서대로 수행하는 것을 말한다. 예를 들어, 자바의 애플릿은 init(), start(), stop() 에소드로 구성되는 프레임워크이다. 개발자는 이러한 에소드를 반드시 구현해야 하며, 원하는 내용을 각 콜백 에소드에 추가하거나 수정한다. 애플릿에 실행 시 정해진 순서대로 수행된다. EJB에서도 빈들을 만들 때 자바 애플릿과 같이 ejbCreate(), ejbActivate(), ejbPassivate(), ejbPostCreate(), ejbLoad(), ejbStore(), ejbRemove()라는 콜백 메소드가 존재한다. EJB 컴포넌트에 대한 코드 인스펙션도 이런 콜백 메소드를 통해 수행되어야 코드에 대한 정확한 이해를 할 수 있고 코드 결함을 발견하기가 쉬우며 효율성도 높아진다. 그림 6은 자금이체에서 사용되는 콜백 메소드와 비즈니스 메소드의 흐름을 보여주고 있다.

3.6 체크리스트 단계

체크리스트는 앞 단계에서 사용되어진 방법들로 발견할 수 없는 결함을 찾기 위해서 사용된다. 코딩 표준에 대한 것을 체크하고 EJB 스펙에 대해서 검사를 수행한다. 코드 인스펙션에서 가장 많이 사용되는 방법인 체크리스트는 사용하기가 편리하고 코드 인스펙션을 실시하는 검사자에게 많은 도움을 제공한다. 체크리스트 사용 시에는 코드에 있을법한 결함을 사전에 질문하고 검사자는 해당 체크리스트의 질문에 대해 코드가 제대로 구현되어 있는지를 비교 확인하면 된다. 체크리스트는 작성 시에 경험적인 자료로부터 많은 영향을 받게 된다. 표 6은 체크리스트를 보여준다.

인스펙션 체크리스트 (컴포넌트명: 자금이체)

				확인 (OX)
정의 결함	에소드	콜백 에소드	1. 모든 콜백 에소드가 정의 되었는가?	
			2. 각 콜백 에소드의 반환형과 파라미터가 적합한가?	
			⋮	
		비즈니스 에소드	1. 각 에소드의 이름이 혼동되거나 비슷하지는 않는가?	
			2. 에소드의 트랜잭션 정의가 적합한가?	
			⋮	
	변수		1. 변수의 이름이 혼동되거나 비슷하지는 않는가?	
			2. 각 변수는 적절하게 초기화 되었는가?	
⋮			⋮	
비교 결함			1. 각 비교 연산자가 올바르게 사용되는가?	
			2. 비교가 부적당하지는 않는가?	
			⋮	
⋮			⋮	

표 6 자금이체 인스펙션 체크리스트

3.7 재작업 및 완료 단계

재작업 단계는 전 단계까지 발견된 결함을 디자이너와 개발자가 수정하는 단계이다. 완료 단계는 인스펙션을 통해 결함이 없다는 것이 검증되어서 CIP가 완료되는 단계이다.

4. 결론

소프트웨어 개발에서 품질은 중요한 문제이다. 품질에 큰 영향을 미치는 결함을 처리하기 위하여 이를 발견해야 하는 시점은 개발이 완료된 후가 아닌 초기이고 이 때에 발견하는 것이 많은 비용과 시간을 줄일 수 있다. 소프트웨어의 규모가 커지고 소프트웨어의 개발이 많아지면서 재사용성이 요구되고 있다. 이 점을 만족시키는 CBD 패러다임에서도 소프트웨어를 100% 에러 없이 만들 수 있는 것은 아니다. 이런 방법으로 개발할 때에 역시 결함을 잡기 위한 많은 비용과 시간이 들며 이런 문제를 해결해 줄 코드 인스펙션 방법도 기존의 방법은 맞지 않거나 부족한 면들이 있다. 본 논문에서의 CIP는 EJB 컴포넌트에 대한 효율적이고 적합한 코드 인스펙션 방법을 제시하고 코드 결함으로 인한 비용과 시간을 줄일 수 있을 것이다. 향후에는 EJB 컴포넌트에 대한 코드 인스펙션의 결과를 리포팅 해주는 도구, 즉 기능자동화 도구를 연구할 계획이다. 현재 코드 인스펙션을 자동화 해주는 도구는 많이 제공되지 않고 있다. 특히 EJB 컴포넌트에 관련된 시나리오 상의 결함이나 논리적인 흐름에 대한 검사 도구는 더더욱 그렇다. 이런 상황에서 이 자동화 도구는 많은 도움과 지원을 제공하게 될 것이다.

참고 문헌

[1] M. E. Fagan, "Design and Code Inspections to Reduce Errors in Program Development," IBM Systems Journal, vol. 15, pp. 182-211, 1976.

[2] A. Davis, Software Requirements: Analysis and Specification(Englewood Cliffs, NJ:Prentice-Hall, 1990), p.20.

[3] M. E. Fagan, Advances in Software Inspections, IEEE Transactions in Software Engineering, 12(7), pp. 744-751,1986.

[4] G. W. Russel, Experience with Inspection in Ultralarge-Scale Developments, IEEE Software, 8(1), pp. 25-31, 1991.

[5] A. Dunsmore, M. Roper, M. Wood, Practical Code Inspection for Object-Oriented Systems, Paris, France, pp. 49-57, July 2001.