4

온라인 게임 개발과정의 코드검증 적용사례

NHN ▌박종채 · 이춘희

1. 서 론

온라인 게임사업은 90년대 말 이후 지속적으로 성장하여 최근에는 국가 경쟁력을 갖춘 국내 대표 성장 산업으로 발전하고 있다. 온라인 게임의 확산에 따라 게임개발업계에서는 기획적 성공요인에 대한 탐색 못지않게 대상게임의 품질향상 측면에도 많은 관심을 기울이고 있다. 이를 위해 대형 온라인 게임개발업체를 중심으로 다양한 테스트 방법과 테스트 프레임워크 마련 등의 검증방안이 함께 고려되고 있는데 그 대표적인 방법 중 한가지가 소프트웨어 품질향상 검증기법 중 가장 효율적이라고 알려져 있는 코드리뷰라 하겠다.

기존의 연구들을 살펴보면 코드리뷰의 성공요인이나 효과에 대한 논문들은 많이 있지만, 방법론적 접근이나 실무적인 적용방법을 검증한 경우는 많지 않았고 특히 게임개발 분야에 특화된 적용사례는 거의 없었다.

따라서 본 연구에서는 온라인 게임개발에서의 결함발생을 줄이기 위한 방법으로의 코드리뷰를 고찰하고 온라인 게임개발의 특성을 고려한 개발 밀착형 코드리뷰 방안을 제안한다. 이 방법에는 정적분석 도구를 통한 코드검증 자동화 방안을 함께 제안하여 실제 개발에 적용하고 여기서 검출된 온라인 게임의 주요 결함 유형을 고찰한다.

본 논문은 총 6절로 구성되어 있다. 2절에서는 일반적인 소프트웨어 개발과 게임개발의 차이점을 통해 게임개발이 가진 특징을 살펴보고, 3절에서는 국내/외 코드리뷰에 대한 연구사례를 통해 코드리뷰의 적용방법 및 효과에 대해 기술한다. 4절에서는 게임개발의 특성과 코드리뷰의 연구사례를 토대로 게임개발에 적합한 코드검증 방안을 제안하고 5절에서는 제안한 방법론을 실제 게임프로젝트에 적용한 사례 및 검증결과를 소개하며 6절에는 결론을 맺고 향후 연구과제에 대해 검토한다.

2. 온라인 게임개발의 특성

2.1 서비스 사업

온라인 게임의 경우 일반적인 패키지 게임처럼 프로그램화 된 게임이 하드웨어에서 동작하는 형태가 아니라 온라인 게임 서비스 업체에서 제공하는 서버에 접속하여 서비스를 이용하는 방식으로 구성 되어 있다. 따라서 게임 개발의 입장에서 보면 프로젝트 종료 후, 모든 개발 활동이 완료되는 것이 아니라 서비스 이후의 요구사항, 패턴, 불편사항 등을 찾아 지속적으로 발전시키면서 해결해야 한다는 특징이 있다. 즉 성공적 온라인 게임을 위해서는 게임 사용자가 상상하고 요구하는 게임에 보다 가까워질 수 있도록 필요한 시스템이나, 기능, 공간, 아이템 등의 요소를 지속적이고도 신속하게 업데이트 해야 한다[1].

2.2 지속적인 통합

온라인 게임은 서비스 산업이면서도 개발과정에서 다양한 내부 컴포넌트의 조합으로 이루어진다는 특징이 있다. 뿐만 아니라 평균적인 제작기간이 길고 투입되는 인원도 많아서 다양한 형태의 커뮤니케이션과 지속적인 통합이 필요한 분야이다. 하위 개발 컴포넌트를 봐도 게임엔진 분야와 기반 플랫폼, 게임 개발, 웹 개발, 사운드 등 다양한 구성요소들이 존재하며 이들을 효과적으로 통합해 가는 과정을 통해 원하는 게임으로 구현된다. 따라서 개발의 측면에서 보자면 온라인 게임은 오랜 기간 지속적인 프로젝트들의 결합 활동이라는 의미로 볼 수 있다. 때문에 통합과정에서 각 컴포넌트 자체 및 인터페이스의 잠재적 결함은 치명적인 서비스 장애로 도출될 가능성이 있고, 이러한 서비스 장애가 많아질 경우, 사용자의 신뢰를 잃어 서비스를 중단하는 사례로 이어질 수도 있다.

3. 코드리뷰 선행연구

3.1 코드리뷰 효과

소프트웨어 제품이나 서비스를 개발하는데 있어 품질 향상을 고민하는 이유는 그것이 개발 비용의 감소로 직결되기 때문이다. 일반적으로 소프트웨어의 개발 비용에 가장 큰 영향을 주는 것은 코드의 재작성 비용으로 알려져 있다. 창조적인 개발활동이 아닌 비정상이거나 동작하지 않는 코드를 수정하고 디버깅하는데 드는 활동이 많게는 전체 개발활동의 약50% 정도를 차지한다고 주장한다. 따라서 오류를 예방하여 결함을 제거하는데 드는 시간을 줄이면 생산성이 향상되고 낭비되는 시간을 절약할 수 있다. 수정비용을 줄이는 대표적이고 직접적인 방법으로는 테스트가 있다. 하지만 테스트 시점에서 발견된 오류를 수정하려면 각 모듈별 연관관계를 이해하여 모듈간의 영향도를 파악해야 한다. 만약 상호간의 연관관계가 발견된다면 오류가 발생한 모듈 외에 연관 모듈까지 재작업 대상이 되어 전체적인 수정비용이 커지게 된다.

테스트 외에도 다양한 품질향상 기법은 있겠지만 가장 쉽고 빠른 효과를 볼 수 있는 방법 중 하나가 코드리뷰(Code Review)이다. 코드리뷰는 수행비용이 저렴하고 결함을 식별하는 비율도 우수할 뿐 아니라 식별된 결함을 제거하는 비용도 상대적으로 저렴하기 때문이다. Steve McConnell은 자신의 저서 'Code Complete'에서 코드리뷰의 효과에 대한 여러 연구들을 들어 이런 사실을 증명하고 있는데 시간당 결함발견 비율이 테스트에 비해 80%정도 우수하다는 연구나(Basili, Selby 1987) 동일한 결함을 발견하는데 있어 코드리뷰에 비해 테스트에 들어가는 비용이6배 이상 높다는 연구(Ackerman, Buchwald, Lewski 1989), 그리고 코드리뷰 활동의 결함검출에 소요되는 자원이 테스트에 비해 약 5배 정도 덜 든다는 연구 등이 여기에 해당된다(Kaplan 1995)[2]. 표 1에는 코드리뷰 효과에 대한 연구 사례를 정리하였다.

3.2 코드리뷰 기법

한편, 코드리뷰에 사용되는 기법 또한 여러 가지가 있는데 인스펙션, 워크스루, 코드리딩, 자동코드리뷰 등이 가장 대표적인 기법이며 각각에 대한 소개는 표 2에 나타내었다[3].

표 1 코드리뷰 효과에 대한 연구사례

연구자	주요효과
Basili, Selby (1987)	시간당 결함 발견 비율이 테스트의 1.8배
Ackerman, Buchwald, Lewski (1989)	동일한 결함의 식별에 있어 테스트의 비용이 코드리뷰 보다 6배 이상 소요
Kaplan (1995)	코드리뷰의 결함검출에 소요되는 자원이 테스트에 비해 1/5정도 임

표 2 코드리뷰 기법

기법	상세설명
인스펙션 (Inspection)	코드 레벨에서 코드산출물을 교육받은 참가자에 의해 상세하게 검토하는 공식적인 평가 기법으로 잘 정의된 프로세스를 기반으로 사전교육과 문서준비가 필요
워크스루 (Walkthrough)	인스펙션과 동일한 목적의 코드리뷰 활동이나 사전준비 및 공식모임 등을 간소화여 작성자와 검토자의 상호활동을 중심으로 진행
코드리딩 (Code Reading)	한 두명이 코드를 직접 읽어가는 방식으로, 검토자의 준비가 끝나면 미팅/메일링을 통해 코멘트와 질문을 주고 받아 정리
자동코드리뷰 (Automated Code Review)	코드리뷰에서 검출하고자 하는 항목을 Rule로 정의한 후, 자동화 도구를 통해 코드를 다양한 레벨로 분석.

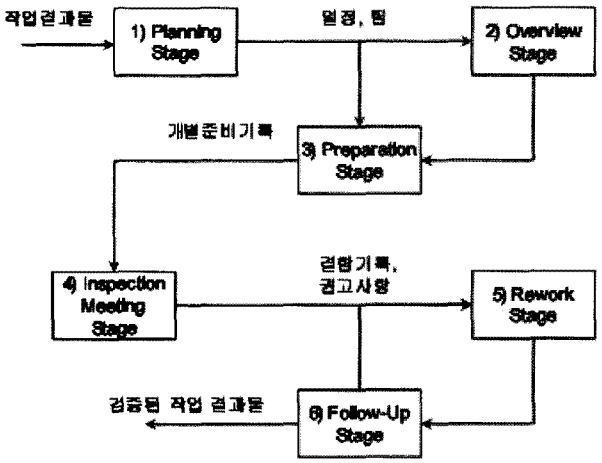


그림 1 Fagan의 Inspection 프로세스

현재 인스펙션(Inspection) 기법으로 사용되는 대부분의 경우는 M.E.Fagan이 제안한 코드 인스펙션 방법에 기초한다[4]. M.E.Fagan의 인스펙션 프로세스는 그림 1과 같이 6개의 주요 단계로 정의되어 있다.

Planning Stage는 개발작업의 완료시점에서 인스펙션을 위한 팀 구성, 역할부여, 체크리스트 및 사전준비 활동을 수행하는 단계이다. Overview Stage는 리뷰를 수행하는 검토담당자에게 리뷰를 위한 사전 설명을 하는 단계이다. Preparation Stage는 검토자가 작업산출물과 결과에 대해 이해하고 검토항목들을 숙지해야 한다. Inspection Meeting Stage는 검토자가 대상산출물에 대해 실질적인 검토를 진행하고 결함을 찾아내는 활동을 하는 단계이다. 이 단계를 통해 찾은 결함은 Rework Stage에서 수정하고 수정한 결과는 Follow-up Stage에서 인스펙션의 진행주체가 최종 확인함으로써 완료된다.

그 외에 Tom Gilb의 인스펙션 프로세스도 널리 활용되고 있다[5]. 그림 2에 나타낸 Tom Gilb의 프로세

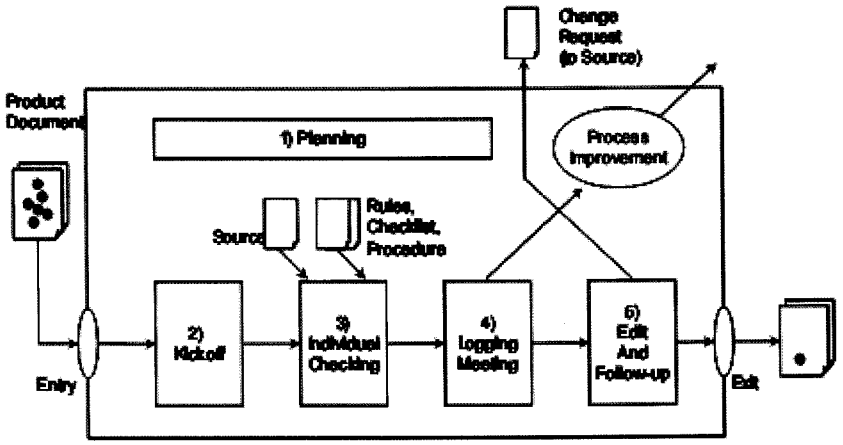


그림 2 Gilb의 Inspection 프로세스

스는 M.E.Fagan의 프로세스를 확장한 형태로 시작기준(entry criteria)과 완료기준(exit criteria)을 두고 그 내부에 계획, 개별검토, 로깅(logging)회의, 수정 및 추적의 단계를 두어 인스펙션을 수행하고 있다.

이처럼 인스펙션 유형의 검토활동은 참석자의 역할이 분명하게 정의되어야 하고 검토자가 사전에 산출물을 검토할 수 있는 시간적 기간이 필요하며 정례화된 회의가 필요하기 때문에 효율적인 활동을 위해서는 3~6명 정도의 인원으로 팀을 구성해야 하는 사전 조건이 있다.

워크스루(Walkthrough)의 경우 인스펙션의 사전준비과정 및 공식미팅을 간소화 하고 작성자와 검토자 중심의 미팅으로 리뷰를 진행하는 활동이란 점을 제외하면 기본적인 프로세스는 인스펙션과 동일하다. 둘 간의 가장 큰 차이는 공식화 정도라고 할 수 있다.

코드리딩(Code Reading)은 공식적인 인스펙션보다 좀더 효율적인 방법이라 할 수 있다. 코드리딩을 사용할 때는 함께 모이는 회의는 불필요하다. 작성자가 작성한 코드를 프린트하여 검토자에게 제공한다. 각 검토자는 체크리스트를 이용하여 소스코드를 라인단위로 읽어 가다가 결함이 식별되면 펜으로 해당 내역을 표시 한다. 작업이 완료되면 검토자는 자신의 리뷰결과를 작성자에게 돌려주고 작성자는 내용을 확인 후, 표시된 결함을 고친다. 식별된 결함에 대한 논의나 검토결과를 공유하고자 한다면 별도 회의를 진행할 수 있다.

효율적인 코드리뷰 수행을 위해서는 체크리스트가 중요하다. 이는 검토자가 과거에 수행했던 결함 유형을 찾기 쉬워진다.

검토활동 중 유사한 결함을 발견했다면 체크리스트에 추가하고 기존에 등록된 결함 중 개발자 레벨에서 잘 통제되고 있는 결함은 리스트에서 제거한다.

자동코드리뷰(Automated Code Review)는 보다 효과적인 코드리뷰 방안 연구의 일환으로 발전해 왔다. 통합 개발환경(IDE)과 연계된 시스템을 활용하여 다양한 레벨에서 코드를 분석하여 결함을 추출하고 결함이 발생한 부분이나 잠재적인 결함요소를 알려주기 때문에 기존의 방식보다 훨씬 더 많은 양의 코드리뷰가 가능하다. 자동코드리뷰는 정적분석(Static Analysis) 도구를 활용하며 사전에 정의한 정보를 이용하여 정적분석 도구에 다양한 룰(rule)을 만든 후, 룰에 위배된 사항들을 찾아내는 방식을 취하고 있다. 따라서 얼마나 개발환경에 적합한 검증 룰을 만들어 낼 수 있는가 하는 것이 자동코드리뷰의 효율성에 있어 가장 중요한 부분이다.

자동코드리뷰를 효과적으로 사용하면 초기 환경설정 이후, 실행하는데 드는 시간과 비용을 줄일 수 있어서 여기서 남는 시간을 보다 높은 수준의 코드검증에 활용할 수 있다는 장점이 있다.

표 3에서는 앞서 설명한 코드리뷰 기법의 특징을 비교하였다[3]. 인스펙션과 워크스루가 고비용의 전통적 리뷰 기법이라면 코드리딩과 자동코드리뷰는 개인별 리뷰가 가능하고 사전 준비가 없기 때문에 보다 비용효율적인 리뷰기법이라고 할 수 있다.

4. 게임개발 코드검증방안 제안

온라인 게임 개발의 경우, 독립적인 소프트웨어 제품의 특징보다는 서비스적 특성이 크며 각종 요구사

표 3 코드리뷰 기법의 특징비교

기법	사전준비	도구	회의여부	상대적 비용
인스펙션	필요	없음	전체회의	높음
워크스루	불필요	없음	부분회의	중간
코드리딩	불필요	체크리스트	개별리뷰	낮음
자동코드리뷰	불필요	소프트웨어	개별리뷰	매우 낮음

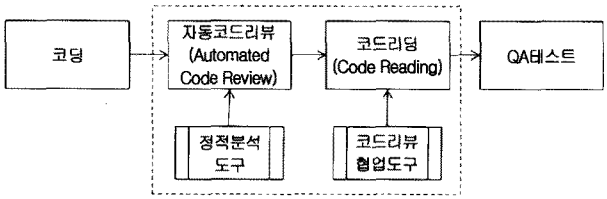


그림 3 게임개발 코드검증 프로세스

항 및 개선사항의 신속한 반영을 통해 사용 연속성을 보장하는 것이 중요함을 확인하였다. 따라서 코드의 품질개선을 위한 검증활동도 형식에 치우친 방법보다는 효율성을 고려한 방법으로 고민하였으며 그 결과 자동코드리뷰와 코드리딩 방식을 채택하였다.

그림 3은 게임개발에 적용하려는 코드검증 프로세스를 나타낸다. 개발이 완료되면 QA테스트로 전환하기 전 단계에서 코드검증을 위한 리뷰를 실시하였으며, 리뷰는 크게 2단계에 걸쳐 진행하였다. 우선, 개발이 완료된 전체 코드를 대상으로 정적분석 도구를 활용한 자동코드리뷰를 실시하고 여기서 도출된 결함을 1차 수정한 후, 온라인 코드리뷰 협업도구를 이용한 코드리딩을 수행하였다.

한편, 편의성과 효율성을 높이기 위한 방법으로 자동코드리뷰의 경우, 별도의 리뷰 기간을 두지 않고 정적분석 도구를 CI(Continuous Integration)와 연동하여 개발과정에서 지속적인 점검과 수정이 가능하게 구성하였으며, 코드리딩의 경우 온라인 협업도구를 활용하여 별도의 미팅이나 문서 없이 자동코드리뷰가 완료된 소스에 대하여 조직 내 리뷰 담당자에게 온라인요청 및 검토를 할 수 있게 구성하였다.

5. 사례적용 및 결과분석

5.1 대상 및 적용방법

본 장에서는 앞서 제안한 게임개발의 코드검증 방안을 국내 대표 게임사인 N사에서 서비스 중이거나 개발중인 게임 프로젝트 및 유지보수 과제 등에 적용하였다. 코드검증방안의 적용은 게임개발 특성 별 차이를 고려하여 게임개발, 웹 개발, 플랫폼개발의 세 가지 유형으로 구분하여 수행하였다. 또한 새로운 리뷰방안을 기존의 전통적 리뷰기법과의 비교 함으로써 제시한 방안의 효율성도 검증할 수 있도록 하였다. 마지막으로 검증과정에서 새로운 코드검증 방안을 통해 도출되는 주요 결함을 게임개발유형 별로 분석하는 것도 연구의 범위로 설정하였다.

시중에는 자동코드리뷰를 지원하는 다양한 오픈 소스 기반의 정적분석 도구들이 나와 있으나, 이번 연구에서는 사내에서 라이선스를 보유 중인 상용제품을 활용하였다. 자동코드리뷰를 위한 정적분석 도구의 경우 자체에서 정의한 10단계 결함 중요도 레벨 중, 명백한 결함에 해당하는 4단계 수준까지의 결함 데이터만 추출하여 분석하였다. 코드리딩을 위한 도구는 온라인 협업기능이 있는 도구를 통해 별도의 미팅이나 문서 없이 개발자의 개별 리뷰가 가능하게 구성하였고 조직 내에서 사용하는 체크리스트는 별도로 정의하여 사전 공유하였다.

각 게임개발 유형별로 1차 정적분석 도구를 통한 자동코드리뷰 후, 2차 코드리뷰 협업도구를 이용한 코드리딩을 수행하여 단계별 결함을 추출하도록 유도하였다. 개발팀 CI와 연계하여 비교적 쉽게 활용 가능한 자동코드리뷰의 경우, 3가지 개발유형에 해당되는 14개 팀에 적용하였고, 코드리딩의 경우, 기존에 조직 내부에서 자율적 코드리뷰 활동을 수행해 오던 게임개발조직 내 3개 팀에 적용하였다.

5.2 적용결과

본 검증방안 중 자동코드리뷰 적용을 통해 도출된 각 게임개발 유형별 결함도출 결과는 표 4에 나타내었다.

온라인 게임 개발과정에서 발생하는 결함 밀도는 게임 개발 유형에 따른 큰 차이는 없었으며 평균적으로 1000라인당 3개 정도의 결함이 식별되는 수준임을 알 수 있다.

표 4 게임 유형별 결함밀도(결함 수/KLOC)

개발유형	팀 구분	결함밀도
게임 개발	A팀	2.64
	B팀	1.19
	C팀	2.96
	D팀	3.47
	E팀	1.95
	평균	2.44
게임 웹 개발	F팀	4.88
	G팀	5.18
	H팀	3.84
	I팀	2.83
	J팀	2.32
	평균	3.81
게임 플랫폼 개발	K팀	2.21
	L팀	2.84
	M팀	9.33
	N팀	0
	평균	3.60

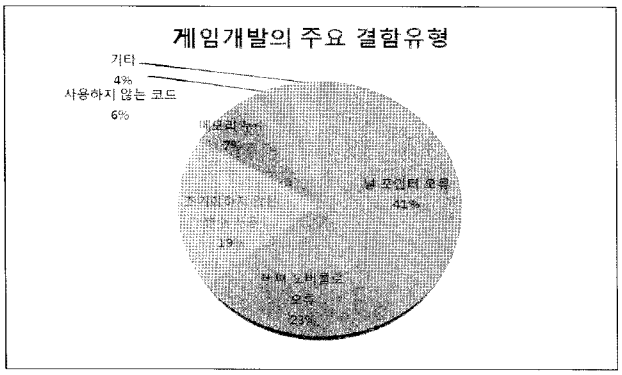


그림 4 게임개발의 주요결함 유형

조직 별 결함유형을 보다 상세하게 살펴보면 게임 개발조직의 경우, 그림 4에서 보는 것처럼 Null Pointer 처리 미흡에 따른 결함유형이 전체의 41%로 가장 많았으며 그 뒤를 Buffer Overflow(23%)와 변수 초기화 미흡(19%) 유형이 차지했다.

게임의 웹 영역 개발의 경우도 그림 5에서 나타낸 것처럼 Null여부에 대한 체크를 하지 않고 변수를 사용한 Null Pointer 오류 유형이 전체의 절반 정도인 46%로 가장 많았으며 그 다음으로 큰 비중을 차지한 결함유형은 객체에 선언된 변수에 값 할당 시, 할당된 값의 유효성을 체크하지 않아서 발생하는 변수처리 오류로 나타났다. 이 유형의 경우, 결함의 중요도는 상대적으로 낮지만, 다른 패키지의 악의적 코드에 의해 값이 변경될 가능성이 있어 보안취약점을 함유한 결함코드라 할 수 있다.

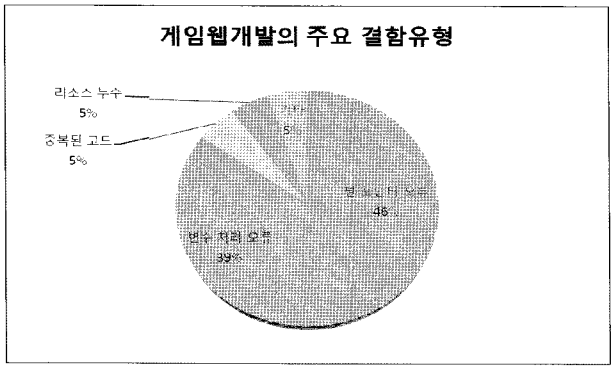


그림 5 게임 웹 개발의 주요결함 유형

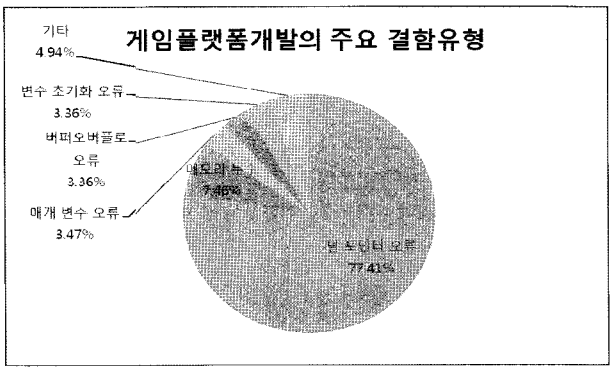
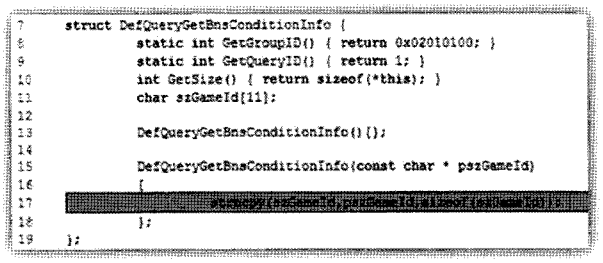


그림 6 게임 플랫폼 개발의 주요 결함유형

게임 플랫폼 개발의 결함유형은 그림 6에 나타내었다. 플랫폼 개발의 경우 역시, Null Pointer 처리 미흡과 관련된 결함이 전체결함의 80%에 육박할 정도로 상대적 빈도가 높았으며 메모리 사용에러와 매개변수 오류 유형이 그 다음으로 많은 비중을 차지하고 있었다.

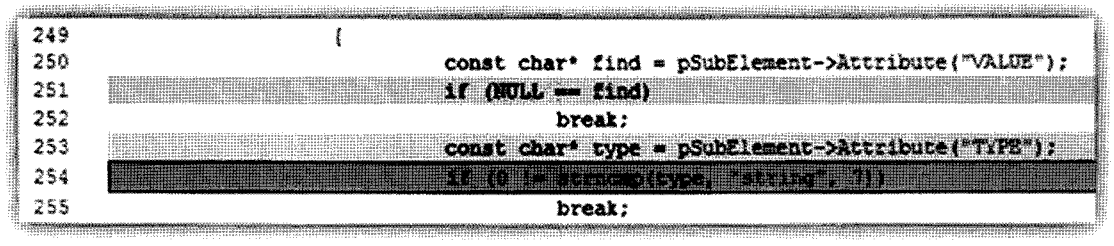
전체적으로 가장 많은 결함유형으로 나타난 Null Pointer 처리오류의 대표적인 사례를 그림 7에 나타내었다. 이 결함은 Null 값에 대한 예외처리를 누락하여 리턴 값에 Null이 반환될 경우, Null값을 참조하게 되는 문제가 발생하는 유형이다. 자동코드리뷰에서 식별된 결함 유형들은 소스코드에 if문을 추가하는 방법으로 해결할 수 있었다.

그림 8은 또 다른 대표적 결함 유형인 Buffer Overflow가 발생 가능한 소스코드 식별사례를 제시하였다. 이 결함은 입력 버퍼의 크기가 출력버퍼보다 큰 경우 Overflow가 발생하는 사례로 주로 게임서버 개발



```
7   struct DefQueryGetBnsConditionInfo {
8           static int GetGroupID() { return 0x02010100; }
9           static int GetQueryID() { return 1; }
10          int GetSize() { return sizeof(*this); }
11          char szGameId[11];
12
13          DefQueryGetBnsConditionInfo(){};
14
15          DefQueryGetBnsConditionInfo(const char * pszGameId)
16          {
17                  [redacted]
18          };
19  };
```

그림 8 대표적인 결함유형 예시 – Buffer Overflow



```
249                 {
250                         const char* find = pSubElement->Attribute("VALUE");
251                         if (NULL == find)
252                                 break;
253                         const char* type = pSubElement->Attribute("TYPE");
254                         [redacted]
255                                 break;
```

그림 7 대표적인 결함유형 예시 – Null Pointer Dereference

에서 많이 식별되었다. 결함의 수정은 소스코드에 버퍼 사이즈를 제어하는 옵션을 추가하여 해결 할 수 있었다.

두 사례는 일반적인 코드리뷰 과정이라면 검토자가 직접 눈으로 확인해야 하는 결함을 정적분석 도구를 이용한 자동코드리뷰 단계를 통해 손쉽게 식별 및 수정할 수 있었다는 점에서 의미가 있었으며 이로 인해 개별 검토활동 시간을 코드리딩 단계에 집중할 수 있었다.

코드리딩을 통한 게임개발단계 결함도출 활동은 오프라인 성격의 코드리딩 활동 대신 온라인 협업도구를 사용하여 효율성을 높이기 위한 시도를 하였다. 그 결과 협업도구 도입 전/후 4개월 동안의 결함 식별 건 수를 비교했을 때, 기존 인스펙션에 비해 결함 식별 건 수가 그림 9에서 보는 것처럼 약 2.7배 증가하였음을 알 수 있다. 뿐만 아니라 코드리딩 과정에서 식별된 결함 유형도 로직 오류와 관련된 부분이 상대적으로 많이 식별되었다.

이러한 현상은 자동코드리뷰를 통해 반복적인 에러나 문법적 오류들이 1차적으로 걸러진 상태에서 코드리딩을 수행하기 때문에 효율성은 높이면서 자동코드리뷰에서 식별하기 힘든 커스텀(custom) 룰이나 로직 위주의 결함 검출에 집중할 수 있기 때문으로 유추할 수 있다. 코드리딩을 통해 식별된 결함 유형 중 로직 오류에 해당되는 사례는 그림 10에 나타내었다. 이러한 로직 오류는 평상시에는 정상적으로 처리되나 오류 시 문제가 될 수 있는 부분으로 상용 혹은 오픈소스 기반의 정적분석 도구를 이용한 자동코드리뷰에서는 식별하기 힘든 결함 유형에 해당된다.

이와 같이 자동코드리뷰와 온라인 기반의 코드리딩 활동을 병행함으로써 기존의 전통적 인스펙션 활동에 비해 개발단계의 코드리뷰 부담은 덜면서 더 많은 결함을 찾아낼 수 있었다.

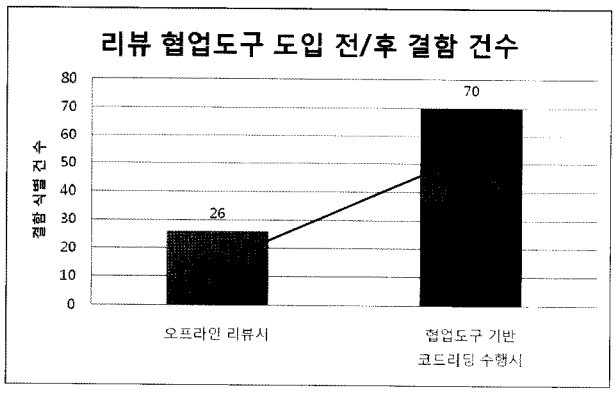


그림 9 협업도구 도입 전/후 결함 건 수 비교

```
●사례1. DB 트랜잭션 후, Rollback누락
GDBTransaction dbTxn;
for (CMemberResultVector::iterator itr
        = vecMemberResult.begin(); itr !=
        vecMemberResult.end(); ++itr)
  { // check ptr
    if (pPlayer == NULL)
    {
  // dbTxn.Rollback(); ◀이 부분 누락
    return
E_GAMERESULT_DBUPDATE_FAIL;
                }
                ...
        }
        dbTxn.commit();
●사례 2. Lock처리 미흡으로 Dead Lock 발생가능
m_pRoomInfo->Lock();
If (NULL == pPlayer)
{
// m_pRoomInfo->Unlock(); ◀이 부분 누락
 return;
}
...
m_pRoomInfo->Unlock();
```

그림 10 코드리딩을 통한 로직오류 식별 사례

5.3 결과분석

대상 소스코드에 대한 결함 검증을 위해 게임 유형별 정적 분석을 수행한 결과 게임개발 유형이나 개발 언어와 무관하게 Null Pointer 처리 미흡에 따른 결함 유형의 빈도가 가장 높게 나타났다. 그 외에 C/C++을 사용하는 게임개발은 Buffer Overflow결함이 플랫폼 개발의 경우는 메모리 처리 미흡으로 인한 결함이 높은 빈도를 차지하였고 Java언어를 사용하는 웹 개발의 경우 변수 선언 시 변수 값 확인 누락으로 인한 결함유형이 높게 나타났다. 조직 별 결함 밀도나 발견되는 결함의 종류가 비슷한 것은 게임 개발의 특성상 함께 사용하는 공통 모듈이 많은 것을 한 원인으로 볼 수 있다. 결과에서 제시하지 못했지만 실제 도출된 결함의 빈도와 결과패턴이 공통모듈에 대한 결함패턴과 유사하게 나타난 점도 특이하다고 하겠다. 또한 결함 분석결과를 통해 조직 별 발생 결함의 유형이 비슷하고 결함의 종류가 다양하기 보다는 특정 결함이 높게 나타나는 특징이 있음을 볼 수 있었는데, 이는 동일한 소스코드 사용 패턴에 따

른 결과로 볼 수 있다. 따라서 결함 빈도가 높은 유형부터 우선순위를 높여 수정하는 전략을 통해 전체 소스코드의 품질을 빠르게 향상하는 방법을 모색해 볼 수 있다. 온라인 코드리딩의 경우 오프라인 코드리뷰 방식에 비해 두 배 이상의 결함 검출향상 효과를 나타내었다.

6. 결론 및 향후 연구

본 논문에서는 정적 분석 도구와 코드리뷰 협업도구를 이용하여 게임개발에 있어서의 효과적인 코드검증 방안을 제안하였으며, 타당성 검증을 위해 실제 게임 개발사의 다수 프로젝트에 이를 적용해 보았다. 특히 전통적 코드리뷰와는 달리 정적분석 도구를 활용한 자동코드리뷰와 온라인 코드리딩를 통한 검증단계로 구분하여 진행함으로써 제안한 코드검증 활동의 효율성을 측정해 보았다. 실제 적용결과, 기존에 비해 코드검증 시간은 단축하고 결함검출은 높이는 효과를 가져왔으며 이러한 결과를 통하여 자동코드리뷰와 코드리딩에 기반한 코드리뷰 방법이 게임 개발 유형에 관계없이 결함검출에 유용함을 확인하였다. 또한 자동코드리뷰를 통해 전체 코드에 대한 결과를 한꺼번에 살펴봄으로써 단편적 리뷰를 통해서 얻기 어려운 게임유형 별로 가장 빈번한 결함 유형을 분석할 수 있었다.

향후에는 게임 개발의 주요 결함유형 정보를 활용한 결함 예방안도 함께 모색되어야 할 것으로 본다. 조직 내 코딩 가이드라인에 결함빈도가 높은 유형의 예방 방안을 추가하여 전파하고 이후 결함의 변화추이를 지속적으로 연구해 보는 것도 좋은 과제가 될 것이다. 이 외에도 자동코드리뷰에 필요한 룰(rule)의 최적화 활동을 통해 특성이 다른 다양한 개발 조직에도 동일한 코드리뷰 방안을 적용하기 위한 연구가 함께 이루어질 필요가 있겠다.

참고문헌

[1] 장경호, 온라인 게임 사이트 성공요인에 관한 연구, 2009, 계명대학교 석사학위 논문

[2] Steve McConnell, Code Complete 2nd Edition, p. 657~p678, Microsoft Press, 2004.

[3] J. Oh, "A Study on Evolution Model of Code Review by Practicing Automated and Manual Code Review", 2005, Information and Communication University

[4] M.Fagan, "Design and code inspection to reduce errors in program development", IBM System Journal vol. 15, No 3, p. 183-211, 1976.

[5] Tom Glib, Dorothy Graham, Susannah Finazi, Software Inspection, Addison-Wesley, 1993.

[6] 김윤정, 리스크 기반의 Software Code Inspection방안 연구, 2006, 서강대학교 석사학위 논문

[7] S/W 개발 Project에서 Inspection 적용사례 연구, 2005, 연세대학교 석사학위 논문

박 종 채

2000 경북대학교 경영학과(MIS) 석사
2000~2004 쌍용정보통신㈜ 품질경영팀
2004~2007 CJ시스템즈㈜ 품질경영팀
2007~현재 NHN㈜ 품질인프라팀
관심분야 : SPI, SQC, PSP, TSP
E-mail : jongchae.park@nhn.com

이 춘 희

2001 성신여자대학교 전산학과 학사
2001~2005 어울림정보기술㈜ 기술연구소 연구원
2005~2008 지니네트웍스㈜ 기술연구소 연구원
2008~현재 NHN㈜ 품질인프라팀
E-mail : choonhee.lee@nhn.com