통신 시스템을 위한 코딩 표준 기반 품질 보증

강선영*, 이은석**
성균관대학교 전자전기컴퓨터공학과
e-mail : sy01.kang@samsung.com, leees@skku.edu

A Quality Assurance based on Coding Standards for Communication System

Sun-Young Kang*, Eun-Seok Lee**
Dept. of Electrical and Computer Engineering, Sungkyunkwan University

요 약

임베디드 소프트웨어의 소스 코드에 대한 품질 검증은 매우 중요한 사항이다. 이에 다양한 시험을 통해 요구되는 품질 속성을 만족하기 위해 노력한다. 본 연구에서는 이러한 품질 확보 활동의 일환으로 구현 단계에서 진행하는 코드리뷰에 코딩 표준을 적용한 사례를 소개한다. 코딩 표준에 대한 다양한 연구와 도메인 내의 결함 분석을 통해 몇 가지의 코딩 표준을 제안하였으며, 이를 조직에 적용하기 위한 고려사항을 언급하였다. 마지막으로 적용 결과에 대해 분석하여 향후 진행될 방향으로 마무리 지었다.

1. 서론

소프트웨어 규모가 대형화 됨에 따라 개발에 소요되는 시간과 인력도 대형화 되고 있으며, 또한 원하는 수준의 소프트웨어 품질을 얻기 위해 필요한 테스트 비용도 점점 더 증가하고 있다. 과거 SW 산업에서는 품질을 확보하는 차원에서 테스트를 중요시하였다. 이에 따른 많은 테스트 절차와 기법들이 소개되고 적용되어 왔다. 특히 임베디드 소프트웨어 개발에 있어서 하드웨어 플랫폼에 탑재하기 전, 소스 코드에 대한 품질 검증하는 작업이 매우 중요한 이슈로 부각되고 있다.[1]

임베디드 소프트웨어의 소스 코드 품질이란, 기능적 요구 사항의 제공과 함께 비기능적 요구사항인 메모리 요구사항, 성능 요구사항, 전력소비 요구사항, 그리고 다양한 플랫폼으로의 이식성 및 유지보수성 등을 의미한다.[2] 본 연구에서는 이러한 임베디드 소프트웨어 소스 코드 품질을 향상시키기 위한 방법으로 정적 분석 도구를 활용한 코딩 패턴 개선 사례를 소개한다. 2 장에서는 소스코드 품질 향상을 위한 코딩 표준에 관한 기존 연구를 조사 분석하고, 3 장에서는 주요 결함 예방 및 품질 속성 확보를 위한 주요 코딩 표준을 제안한다. 4 장에서는 이러한 코딩 표준을 조직에 적용할 때의 고려 사항에 대해서 언급하였으며 5 장에서는 결론을 기술한다.

2. 관련연구

코딩 표준을 정립하는 이유는 프로그래머가 자신의 취향이나 선호가 아닌, 프로젝트나 조직의 요구사항에 의해 결정된 가이드라인을 따르도록 하기 위함이다. 이러한 가이드라인은 불안전한 코딩이나 취약점이 될 만한 정의되지 않은 행동을 줄이는 데 있다. 이 코딩 표준을 통해 신뢰성 있고 공격에 강한 높은 품질의 시스템을 만들 수 있기 때문이다.[3] 이러한 취지로 소프트웨어 소스코드 수준의 보안성 강화를 통한 근원적인 접근법에 관심이 증대되면서 이에 대한 연구도 활발하게 진행되고 있다. 국내에서도 행정안전부를 중심으로 전자정부 소프트웨어 개발 시에 보안 취약점에 대한 검사를 의무화 하는 정책을 추진하고 있다.

이러한 모든 활동은 소프트웨어의 취약성을 파악하는 것에서부터 시작할 수 있다. CWE(Common Weakness Enumeration)는 미국 국토 보안부내 국가 사이버 보안국 의 지원으로 MITRE 에서 소프트웨어 취약점을 다양한 관점에서 분류하여 모아 놓았다. CWSS(Common Weakness Scoring System)은 아키텍처, 설계, 코드 또는 구현상의 취약점들과 보안상 중요 사항에 대한 취약점에 대한 점수 체계를 수립하였다. CVE(Common Vulnerabilities and Exposures)는 발견된 보안 취약점의 히스토리를 기록해 두었다. 현재 CVE 조정위원회를 통해 관리된 목록은 1999 년부터 시작해 매년 1,000 여개 목록이 추가되고 있다. MITRE 와 SANS 협회에서는 매년 ‘CWE/SANS top25’ 라고 불리는 리스트를 발표하고 있다. 이 리스트에는 가장 위험한 프로그래밍 오류라는 부제를 가지며 소프트웨어 취약점을 야기하는 가장 중요한 프로그래밍 오류들을 명시화 하고 있다.

<표 1> CWE/SANS top 25 목록

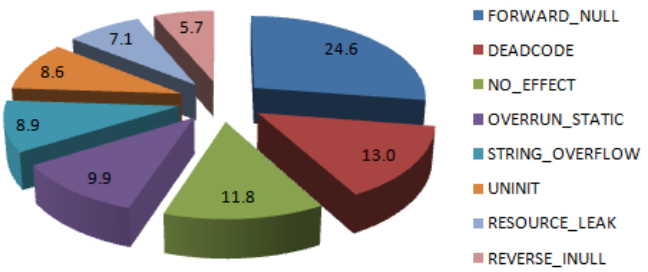
분류	설명
컴포넌트 사이의 안전	분리된 컴포넌트, 모듈, 프로그램, 프로세스, 쓰레드 간 혹은 시스템간에 주고 받

하지 못한 상호작용	는 데이터들의 안전하지 못한 상호 작용과 관련된 9 개 CWE 목록
위험성이 높은 자원 관리	소프트웨어가 시스템의 중요 리소스들에 대한 생성, 사용, 이동 혹은 사용 후 반납 등의 동작을 적절히 관리하지 못하는 문제와 관련된 9 개 CWE 목록
허점 많은 방어	잘못 사용하거나, 남용 되거나 혹은 일상적으로 무시되는 매우 고질적인 습관들과 관련된 7 개 CWE 목록

CERT C 코딩 표준은 안전한 코딩을 위해서는 취약성을 만들어내는 공통적인 프로그래밍 오류를 찾아 이를 기반으로 안전한 코딩 표준을 만드는 것을 가이드하고 있다. 이러한 연구를 바탕으로 89 개의 규칙과 132 개의 권고로 구성된 표준안[4]을 제시하였다.
MISRA-C 표준은 자동차 산업과 관련된 소프트웨어 개발 과정에서 사용하는 것으로 1998 년에 127 개의 규칙으로 시작해서 2004 년 21 개의 영역(ex.환경, 런타임)에 걸쳐 121 개의 필수 규칙과 20 개의 권장 규칙으로 구성되어 있다.[5] 해당표준은 다양한 정적 분석 도구에 의해서 체크 될 수 있는 가이드라인을 제시했으며 일부 항목의 체크를 위해서는 동적 분석 도구를 필요로 한다.

3. 통신 시스템에서의 주요 결함 및 코딩 표준

1) 통신 시스템 주요 결함 분석
코딩 표준을 제안하는데 있어, 도메인 내의 결함 유형을 분석하는 것도 중요한 일 중에 하나이다. 언어 자체의 취약성과 함께 도메인 내에서 자주 발생되는 결함 유형을 예방하는 효과를 기대할 수 있기 때문이다. 본 논문에서는 통신 시스템에서 진행되는 약 20 개의 프로젝트에 대한 결함 유형에 대한 자료를 정리하였다. 결함 유형 자체가 분석하는 관점에 따라 다양한 해석이 나올 수 있기 때문에 결함 검출 도구[6]에서 나온 데이터를 기반으로 분석을 진행하였다. 이에 통신 시스템 특성과 더불어 결함을 도출하는 도구의 특성이 반영되었음을 고려해야 한다. 또한, 해당 결과가 조직 내에서 인정될 수 있도록 결함 검출 도구에 대한 신뢰를 사전 확보하고 있어야 한다.
분석 결과 널 포인터를 참조하는 항목이 24.6%로 가장 많은 항목을 차지했으며 소스코드상에 집행되지 않는 코드나 의미 없는 코드에 대한 부분이 10% 이상을 차지하였다. (이외 항목에 대해서는 (표 2)와 (그림 1)을 참조)

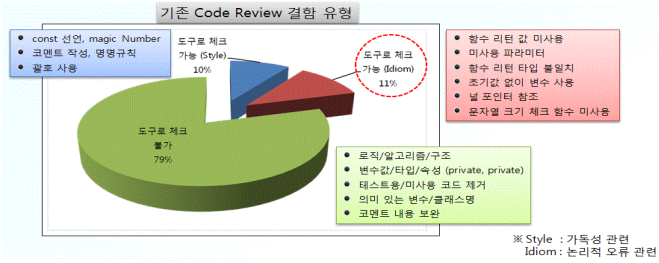


(그림 1) Checker 별 결함 비율

<표 2> Checker 별 설명

Checker	Description
FORWARD_NULL	널 포인터를 체크하지 않고 포인터를 참조
DEADCODE	소스 코드 상에서 정적으로 직접 수행될 수 없는 코드 검출
NO_EFFECT	아무 역할도 하지 않는 문장이나 표현식 검출
OVERRUN_STATIC	할당된 정적 메모리(array)의 경계 밖을 참조하는 오류 체크
STRING_OVERFLOW	C 스타일의 문자열에 오버 런이 발생하는 오류 체크
UNINIT	초기화 되지 않은 변수들의 사용 체크
RESOURCE_LEAK	자원(파일,메모리 등)을 소유하고 있는 범위를 넘어서는 오류 체크
REVERSE_INULL	Dereference 후에 널 체크하는 오류 검출
PASS_BY_VALUE	함수의 파라미터가 잠재적으로 너무 큰 데이터 타입이나 이를 참조하는 경우 검출

2) 코드 리뷰 기반의 결함 분석
본 절에서는 코드 리뷰를 통해 발견된 결함을 분석하였다. 아이 체크를 통해 발견된 코드 리뷰의 결함 분석을 통해 정적 분석 도구를 통해 검출 가능한 항목을 도출하였다.
약 21%가 도구 적용을 통해 검출할 수 있는 항목으로 나타났으며 79%는 로직, 알고리즘에 관련된 사항으로 도구로의 검출은 불가능한 사항 이였다.



(그림 2) Code Review 결함 유형

3) 코딩 표준 사례
이미 다양한 분야에서 Coding Rule 을 제안하였다. 앞에서 언급한 MISRA 는 자동차 분야에서 Ellemtel 은 통신 시스템에서 폭넓게 활용되는 C++ 기반의 Rule 을 제안하였다. Joint Strike Fighter 에서도 코딩 표준을 제안하여 안정적인 코딩 개발을 위한 지침을 마련하였다. 본 논문에서는 위의 분석 내용을 바탕으로 통신 시스템 내에서 적용 가능한 코딩 룰을 몇 가지 제안하였다. 해당 항목은 이미 제안된 항목과 겹칠 수 있으며 표준을 정의하는 사례로 제시한 것이다. 제안되는 Rule 은 모호하지 않고 명확해야 하며 예외 사례가 유한해야 한다. 하나의 항목에 대해 여러 개의 Rule 이 제시 될 수 있다. 본 사례에서는 6 개의 Checker 와 코드 리뷰에서 나온 2 개 분류에 대한 코딩 표준을 제안하였다.

<표 3. 제안 코딩 표준>

분류	제안 코딩 표준
FORWARD_NULL	-. 함수가 포인터를 리턴 하는 경우, 널 체크
DEADCODE	-. Return, break, continue, goto 구문 뒤에 비 실행 코드가 없어야 함 -. Switch 구문 뒤에 비 실행 코드가 없어야 함
NO_EFFECT	-. 모든 문장은 적어도 하나 이상의 역할을 수행해야 함
OVERRUN_STATIC	-. 배열의 인덱스 사이즈를 체크해야 함
STRING_OVERFLOW	-. 버퍼 오버플로우를 발생 시킬 수 있는 불안전한 함수(strcpy, strcat, sprint, gets)을 사용하지 말아야 함
UNINIT	-. Unsigned integer 변수에 signed 상수를 할당하지 말아야 함 -. 모든 변수를 초기화
함수 리턴값	-. 리턴 타입 일치 -. 리턴 타입이 없는 경우 void 로 선언
함수 파라미터	-. 파라미터의 수는 선언된 것과 일치 -. 호출.된 파라미터는 모든 활용되어야 함

4. 조직 내 적용 시 고려 사항

품질에 대한 관심이 높아질수록, 테스팅과 관련한 비용 투자에 너그러워지는 것이 사실이다. 다양한 테스팅 기법과 절차를 구체화 하고 테스트 케이스를 확보하는 등의 활동을 통해 보다 완벽한 테스팅을 위해 힘쓴다. 과거 시스템 시험, 통합 시험 등에 많은 비용을 투자했다면 현재의 관심은 앞 단계에서의 품질 확보를 위한 방안에 관심이 기울어지고 있다. 이에 본 절에서는 통신 사업을 수행하는 도메인에서 코딩 표준에 대한 접근 시 고려 사항에 대해 간략히 기술하였다.

1) 표준 정의

초기 적용 시에는, 코딩 표준은 직접적인 결함이 아니고 결함을 양산할 수 있는 잠재 결함의 성격을 가지다 보니 개발자 및 관리자의 호응을 얻는 것이 쉬운 것이 아니다. 이에 관리자에게는 코딩 표준에 대한 충분한 이해와 개발자와는 표준을 제정하는데 있어 충분한 역할을 부여하여 진행함으로써 표준에 대한 확신을 가질 수 있도록 해야 한다. 위에서 진행한 결함 분석 및 코드 리뷰 기반 정보는 표준을 정의하고 관련자의 호응을 얻는데 중요한 기반 정보로 활용될 수 있다.

2) 도구 선정

코딩 표준에 대한 적용을 위해 중요한 것 중의 하나로 수행 노력을 최소화 해야 한다. 과거 체크 리스트 기반의 리뷰에서 코딩 표준에 대한 준수 여부는 도구를 활용하여 쉽고 명확하게 점검할 수 있어야 한다. 도구 선정 작업에서는 도구 자체에서 제공하는 다양한 코딩 표준과 함께 조직의 특성에 맞는 표준을 생성할 수 있는 기능도 내제되어 있어야 한다. 다양한 정적 분석 도구가 상용화되어 있으며 4.1 에서 선정한 표준을 잘 수용할 수 있고 사용자 편의성을 고려하여 도구를 선정해야 한다. 필요에 따라서는 룰을 적용함에 따라 표준을 더 상세화 하는 작업이 진행 될 수 있다.

3) 정책 수립

코딩 표준을 조직 및 과제에 적용할 때는 다음과 같은 정책이 정의되어야 한다. 개발 어느 시점부터 언제까지 코딩 표준을 점검할지를 정의해야 하며 점검 단위를 레거시 코드를 포함할지에 대해서도 판단해야 한다. 통신 시스템의 경우 현장에서 운영하는 코드를 기반으로 신규 개발이 이루어지기 때문에 레거시 코드에 대한 수정 여부가 큰 이슈가 될 수 있다. 이는 조직 및 사업자의 특성에 따라 선택 운영 할 수 있는 정책을 수립할 수 있다. 정책 수립 시에는 수립된 표준의 운영 주기에 대해서도 정의해야 한다. 적용의 효율성을 고려하여 조직에서 표준을 적용할 때는 일반적으로 50 개 이하로 할 것을 권장한다. 한정된 범위에서 표준을 운영하다 보면 의미 있는 표준들로 개정하는 절차가 필요하게 된다. 조직의 성숙도에 따라 이미 체질화 되어 더 이상 발생되지 않는 표준에 대해서는 제외하고 현장에서 발생한 문제점이나 결함/코드 리뷰를 추가 분석하여 의미 있는 표준을 추가하는 활동 등이 지속적으로 진행되어야 한다. 일반적으로 일년 단위로 표준에 대해 재점검 할 것을 권장한다. 이러한 활동을 통해 가장 필요한 코딩 표준이 현재 조직에 적용되고 있음을 인식시켜야 한다.

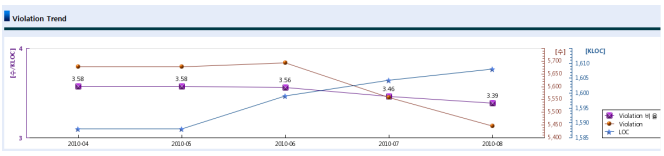
4) 모니터링

마지막으로 코딩 표준에 대한 준수 여부를 모니터링 하는 방법 및 절차에 대해 고려되어야 한다. 이러한 정보는 개발자에게는 도전 정신을 발휘하게 도와주면 관리자에게는 코드 품질 확보에 대한 노력을 직접 확인 할 수 있게 하는 계기가 될 수 있다. 또한, 향후 정량적인 평가에도 활용될 수 있다. 모니터링 방법은 위에서 정의된 사항들을 포함하여 개발자/관리자/시스템 담당자의 기능으로 보여 줄 수 있어야 하며 표준이 변화되는 것에 대해서도 유연하게 반영되어야 한다. 상용 도구에 따라 부분적으로는 이러한 기능을 포함할 수도 있다.

5. 평가

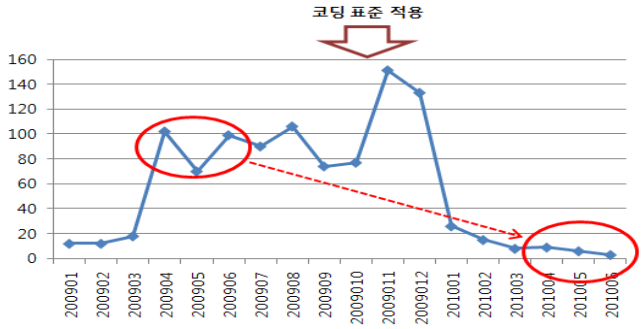
해당 사례는 코드 품질 확보 차원에서 코딩 표준을 제안하고 이를 적용한 사례를 간략히 소개하였다. 위와 같은 절차를 걸쳐 약 10 개월 정도의 적용한 효과를 간략히 분석하였다.

(그림 3)를 통해 레거시 코드를 인정하는 160 만 라 정도의 과제 규모에서 코딩을 시작할 당시 대비, 코딩이 완료되었을 때 코드 내에 잠재된 Violation 비율이 0.19 정도 개선된 것을 확인할 수 있다.



(그림 3) Violation Trend

(그림 4)은 결함 관리 시스템을 통한 간접적인 분석 결과이다. 해당 결과는 조직의 다양한 품질 확보 활동과 패키지 배포 시점과 연계되어 있어 코딩 표준에 대한 직접적인 결과라고 말 하기에는 어려움이 있으나 코딩 표준이 적용된 시점 이후에 초기 상당한 결함이 발생되었으며 이후 4~6 개월 결함 수치가 전년도 대비 66.4%가 감소된 결과를 확인 할 수 있었다



(그림 4) 월별 결함 유입

이러한 접근을 통해 코딩 표준에 대한 어느 정도의 인식을 맞출 수 있는 계기가 되었으며, 형식적으로 진행되던 리뷰에 실질적인 정보가 도출되는 계기가 되었다. 부수적으로는 시험 단계 이전에 품질 속성을 확인할 수 없었으나 이러한 접근을 통해 코딩 시작부터 품질 속성 및 진행 정도 및 라인 수 증가 등의 추가 정보를 확인 할 수 있는 환경이 제공되었다.

6. 결론

지금까지 다양한 분과에서 진행되는 코딩 표준에 대한 연구를 바탕으로, 통신 시스템에서의 코딩 표준을 일부 제안하고 조직에 적용하기 위한 절차를 정의하였다. 해당 적용을 통해 직접적으로 Violation 밀도의 개선과 간접적으로 결함 감소 및 코딩 기간부터 품질에 대한 마인드를 제고하는 계기가 되었다. 약 10 개월 간의 적용을 통해 다음과 같은 미진 사항도 도출되었다. 조직/과제 전체에 표준 코딩을 접근하다 보니 과제 특성 및 요구되는 품질 속성에 대해서는 감안하지 못 하는 사항이 발생하였다. 코딩 표준을 세분화하여 이를 반영할 수 있는 분류 체계 수립이 필요함이 인지되었다. 현재는 코딩 표준 준수를 통한 정성적 평가 및 비용 측면의 효과 분석도 미진한 상태다. 이러한 효과 분석은 코딩 표준을 계속적으로 나아가기 위해 중요한 요소로서 향후 연구되어야 할 과제로 남았다.

코딩 표준에 대한 접근은 다양한 품질 보증 활동의 일부이다. 기존의 어떤 부분을 대체할 수는 없으며 코드 리뷰의 일부로 활용될 수 있는 한계가 있다. 코드 리뷰는 알고리즘/구조에 대한 부분을 점검하는 주요한 활동이다. 짧은 개발 일정 동안 이러한 주요 활동에 더 많은 시간을 투여하기 위해 도구를 통한 코딩 표준 준수로 접근되어야 할 것이다.

참고문헌

[1] David E. Simon “Embedded Software Primer” Addison-Wesley
[2] P,Koopman, “Embedded System Design Issues(the Rest of Story), Proceedings of the International Conference of Computer Design (ICCD 96)” Austin
[3] Robert C Seacord “The CERT C Secure Coding Standard” Addison-Wesley
[4] Jason A Rafai ”CERT secure Coding Initiative” Addison-Wesley
[5] MISRA, Guidelines for the Use of the C Language in Vehicle Based Software, April, 1998
[6] http://www.coverity.com/products/static-analysis.html