ETRI

2017
Electronics and
Telecommunications
Trends

SW 개발 R&D 프로젝트에서 소스 코드 품질을 위한 정적분석

Static Analysis for Code Quality on R&D Projects of SW Development

박정현 (J.-H. Park) 품질보증연구실 책임연구원
박영식 (Y.-S. Park) 품질보증연구실 책임연구원/실장
정효택 (H.-T. Jung) 품질혁신실 책임연구원/실장

* 본 연구는 '경영품질성과 향상을 위한 프로세스 개선 및 혁신전략 연구 과제(16OL1220)' 수행 결과임/본 연구 결과는 ETRI 공식 견해와 다를 수 있음.

본고에서는 SW 개발 R&D 프로젝트에 개발 소스 품질 검증을 위한 정적분석 실행 방안을 기술한다. 정적분석 실행 방안으로 소스 코드 품질 검증 대상 SW 개발 R&D 프로젝트 범위와 정적분석 검증 기준을 기술하고, 정적분석 검증을 위한 구체적 실행 방안과 검증 프로세스 그리고 정적분석 검증을 지원 환경 체계를 제시했다. 또 제시된 정적분석 검증 실행 방안에 근간하여 수행 중인 프로젝트에서 개발한 소스 코드에 대해 시범적으로 정적분석 검증을 수행하고 이를 바탕으로 정적분석 검증 실행의 안정적인 도입 및 운영 방향을 시사점으로 기술하였다. 제시된 정적분석 검증 실행 방안은 SW 개발 R&D 프로젝트에서 개발 소스 코드 품질 관리 및 유지를 위한 방안으로 고려해 볼 수 있을 것으로 기대한다.

Ⅰ. 서론

SW 개발 프로젝트 성공을 위해서는 개발 프로세스 수립 및 이행과 적절한 인력 참여 그리고 개발 방법과 도구를 포함한 개발 환경 구축이 중요하다. SW 개발 프로젝트의 성공 여부를 판단하는 기준은 SW 개발 주기 측면에서만 보면 주어진 기간과 주어진 예산에 목표하는 품질과 요구사항을 만족하도록 개발하는 것으로 해석할 수 있다. SW 개발 프로젝트에서 품질은 개발 시스템에 대한 요구사항의 달성 여부로 평가될 수 있는데, 품질은 SW 개발 주기에서 예산과 기간에 영향을 주는 핵심 요소이기도 하다. 품질은 요구사항 정의, 설계, 구현, 시험 등 SW 개발 주기의 각 단계에서 발생하는 결함 발생 수준에 영향을 받는다. 개발 과정의 각 단계에서 발생하는 결함을 감소하기 위해서는 각 단계 마다 리뷰 및 테스트와 같은 결함 식별 활동과 함께 각 단계에서 기존에 발생된 결함에 대한 분석을 통해 발생 가능한 결함을 사전에 차단하는 것이 필요하다.

ICT 분야에서 기초, 원천, 표준화, 정책, GCS, 그리고 SW Bank 등록 대상 과제 등을 포함해 연간 500~600개의 다양한 R&D 과제를 수행하는 ETRI에서도 SW를 개발하는 R&D 프로젝트 수행 비중은 크다. 더욱이 임베디드, 빅데이터, 콘텐츠, 클라우드, 모바일, 음성 인식, 융합, 보안 등을 포함한 다양한 프로젝트와 TRL 1~9 형태의 기술 개발 과제 그리고 과제별 서로 다른 개발 언어를 고려한 다양한 개발 환경을 갖는다. 또 과거 TDX나 CDMA를 개발할 때는 연구 사업에 국한된 시스템 개발 체계와 방법론을 통해 R&D 품질 관리를 위해 이행해왔으나 2008년부터는 R&D 프로젝트 표준프로세스 수립과 Q-mark 모델 개발 및 운영[1][2]을 통해 R&D 프로젝트에 대한 품질 관리를 이행해 오고 있다. Q-mark 모델에는 요구사항정의 및 시험 등 5개 프로세스를 포함하고 있으며 R&D 프로젝트 수행 결과에 대한 품질 인증을 받기 위해서는 기본적으로 이들 5개 표준프로세스를 기반하여 과제가 수행되어야 한다. 그러나, 다양한 R&D 프로젝트 형태와 기술 개발 수준 그리고 과제별 서로 다른 개발 환경으로 인해 구현 단계에서 발생하는 결함을 차단하고 검증해야 하는 코드 품질 기준에 대해서는 아직까지 구체적 방안이나 기준을 정하지 못한 실정이다. 이렇다 보니 SW 개발 R&D 프로젝트에서 코드 품질 수준의 지나친 편차가 발생하고, 나아가 개발한 소스 코드에 보안 취약점이나 실행 오류와 같은 치명적인 결함을 갖는 경우도 발생한다. 또한 개발한 기술의 소스 코드에 치명적인 결함이 있는 상태에서 업체에 기술이전하는 경우 소스 코드 품질 저하로 인한 추가적인 문제를 유발할 수 있게 된다. 따라서 SW 개발 R&D 프로젝트에서 개발 기술의 소스 코드 결함으로 발생하는 문제를 기본적으로 제거하기 위해서는 개발된 소스 코드에 대한 품질 검증이 필요하다.

이에 본 연구에서는 SW 개발 R&D 프로젝트에서 코드 품질 검증을 위한 정적분석 실행 방안을 제시한다. 이를 위해 Ⅰ장 서론에서 SW 개발 R&D 프로젝트에서 개발 소스 코드 품질 관리의 필요성을 기술하고, Ⅱ장에서는 코드 품질과 정적분석을 기술하며, Ⅲ장에서 SW 개발 R&D 프로젝트에 정적분석 실행 방안을 제시한다. Ⅳ장에서는 현재 수행 중인 SW 개발 R&D 프로젝트에서 개발한 소스 코드에 대해 제시된 정적분석 검증 실행 방안을 시범적으로 적용하고 그 결과를 바탕으로 R&D 프로젝트에서 소스 코드 품질 검증을 위한 정적분석 실행 방안의 안정적 도입 및 운영 전략을 시사점으로 기술한다. 그리고 Ⅴ장에서 결론으로 본 연구 결과를 요약한다.

Ⅱ. 코드품질과 정적분석

본 장에서는 코드 품질과 정적분석에 대해 기술한다.

1. 코드 품질

최근의 SW는 융·복합화가 이루어지면서 점차 대형화,

복잡화되고 있는 추세이며, 자동차, 비행기, 원자력 등 모든 산업 제품에서 SW가 차지하는 비중은 점차 커지고 있는 실정이다. 이에 따라 SW 품질은 그 중요성이 점차 증대되고 있으며, 이러한 SW 품질의 중요성은 앞으로 더욱 커질 전망이다.

가. 코드품질

스티브 맥코넬[3]은 코드 품질을 좌우하는 SW 구현 단계를 상세설계, 코딩, 디버깅, 단위 테스팅, 기술적 검토, 성능최적화 단계로 이루어진다고 하고 있다. 여기서 상세설계는 아키텍처 설계를 확장하여 클래스, 루틴 차원까지 처리하는 단계를 의미하고, 디버깅은 결함을 찾고 수정하는 단계, 단위테스트는 개발자 스스로 함수 단위 형태로 시행하는 단계, 그리고 기술적 검토는 리팩토링, 코드 리뷰, 설계 리뷰를 진행하는 단계, 성능최적화는 메모리 등 자원 활용 효율성 및 처리 속도 개선을 감안하여 진행하는 단계를 의미한다. 또 실제 코드를 개발하는 코딩 단계는 다시 구현설계, 추상화, 의사코딩, 루틴작성, 방어코딩 과정으로 이루어진다고 하고 있다. 여기서 구현설계라 함은 설계도 구현의 일부로 SW 기능 구현 혹은 문제 해결 방법에는 다양한 설계 방법이 존재하며 한번의 설계로 완벽한 설계를 할 수 없어 프로그래머들은 공식적이든 아니든 설계를 하면서 구현을 하고 다시 설계를 보완하고 구현을 반복한다는 의미이다. 구현설계 단계에서는 복잡성을 최소화하면서 느슨한 결합도, 그리고 확장성과 재사용성 및 이식성 등을 고려하고 SW 구현 4가지 주요 원칙을 고려하는 것이 SW 품질에 중요한 영향을 준다고 하고 있다. 첫 번째는 코딩의 복잡성을 최소화시키는 것이다. SW 구현의 복잡성을 줄이는 문제는 복잡한 정보를 유지하는 인간 능력의 한계와 관련이 있다. 이러한 한계로 인해 SW 디버깅과 같은 복잡한 문제를 해결하는 능력은 제한적일 수밖에 없으며 이는 결국 SW 코드는 단순하고, 가독성이 있어야 한다는 것을 의미한다. 둘째로 향후에 일어날 변화를 고려해야 한다. 시간이 지남에 따라 대부분의 SW 시스템은 변하기 때문에 이런 변화를 예상하고 SW를 구현해야 한다. 대부분의 경우 향후 일어날 일정부분은 예상이 가능하며, 코드는 이런 변화요소들에 대응할 수 있어야 한다. 셋째는 SW 구현은 검증이 가능해야 한다는 것이다. 이는 구현되는 SW가 테스팅이나 다른 검증 기술을 통해 에러의 발견이 용이한 형식으로 구현되어야 한다는 것을 의미한다. 마지막으로 표준을 활용하는 것으로, 앞서 언급한 세 가지의 원칙들을 표준 프로그램 언어 및 외부 프로그램과의 인터페이스를 제공하는 표준 API를 선택함으로써 용이하게 적용될 수 있다.

나. 코딩규칙 및 표준의 중요성

규모가 큰 SW 개발 프로젝트에서 프로그래머들이 하나의 형식을 따른다면 앞서 언급한 세 가지의 원칙들을 쉽게 달성할 수 있게 된다. 이런 경우, 적용하는 표준은 모범 사례를 제시할뿐 아니라 임의의 가능한 선택으로부터 프로그래머가 일관된 선택을 하도록 요구하게 된다. 이 경우 외부 표준화 기구의 표준보다는 일반적으로 기업이나 조직 내부에서 자체적으로 만들어진 표준이 활용된다. 그리고 다수의 개발자가 참여하여 공동으로 SW를 구현하는 경우 개발자의 코딩 스타일은 프로그램의 논리에 영향을 주는 것은 아니지만 일의 효율에 큰 영향을 미칠 수 있기 때문에 코딩 규칙이나 표준을 준수하여 작성해야 한다. 그렇게 완성된 소스 코드는 마치 한 명의 개발자가 하나의 세션에서 코드를 작성한 것처럼 일관적인 스타일을 유지한다. 이것이 많은 SW 개발 회사들이 SW 소스 코드 품질을 위해 자체적으로 코딩 규칙이나 표준을 갖추고 있어야 하는 중요한 이유이기도 하다. 이렇듯 SW 개발 조직이나 업체들은 SW 개발 프로젝트를 성공적으로 수행하고 관리할 수 있도록 코딩 규칙이나 표준을 수립하여 모든 개발자가 프로젝트

에서 조화롭게 작업할 수 있도록 한다. 이렇게 해서 좋아진 소스 코드의 가독성은 개발자가 SW 시스템을 이해하는 데 영향을 미치게 되고, 가독성이 향상되면 이해가 용이해지고 유지보수가 수월해지며 일반적으로 소스 코드의 품질도 좋아진다. 또한, 코딩 규칙이나 표준을 지키게 되면 새로운 기능을 추가하고 기존 기능을 수정하기 위해 SW 시스템을 쉽게 변경할 수 있는 코드 유지관리성도 좋아지게 된다. 가독성과 유지 관리성은 여러 가지 요인에 의해 좌우되지만 모든 SW 개발자에게 영향을 미치는 SW 개발의 한 측면은 코딩 기술이다. 개별 개발자들의 코딩 기술은 개발자별로 상이하기 때문에 개발 팀에서 우수한 코드를 작성할 수 있도록 유도하는 가장 쉬운 방법은 코딩 표준을 설정한 다음 이 표준을 루틴 코드 검사에서 적용하는 것이다. 고급 소스 코드를 작성하기 위해 완벽한 코딩 기술과 뛰어난 프로그래밍 기술을 사용하면 SW 품질과 성능을 향상시키는 데 중요한 역할을 할 수 있다. 또한, 명확하게 정의된 코딩 표준과 올바른 코딩 기술을 일관적으로 적용하고 루틴 코드 검사를 적용하면 보다 쉽게 SW 시스템을 이해하고 관리할 수 있으며 SW 소스 코드 품질 수준을 일정하게 유지할 수 있게 된다. 그러나 결함이 전혀 없는 100% 완벽한 SW는 존재할 수 없으며 모든 SW는 잠재적으로 어느 정도의 결함을 내재하고 있고, SW 개발 R&D 프로젝트에서도 예외가 아니다. 따라서 SW 개발 R&D 프로젝트에서 소스 코드를 완료 혹은 릴리지 할 수 있는 품질지표와 기준이 필요하며 NIPA[10][11][12]에서는 SW 개발 R&D 프로젝트에서 코드품질 5대 기준 지표로 구문 및 분기 실행 여부를 나타내는 코드 커버리지, 코딩 표준 준수율, 정적분석 결함 제거율, 함수 및 코드 복잡도를 표기하는 사이클로매틱 복잡도, 메소드(클래스) 사용 중복성을 들고 있다. 이들 코드 품질지표는 항공이나 국방 등 고신뢰성 SW 개발과 SI형 SW 개발 그리고 R&D형 SW 개발로 나누어 제시하고 있다. 따라서 ETRI에서 진행하는 SW 개발 R&D 프로젝트에서도 이들 품질 지표를 응용하여 소스 코드 품질 기준을 고려해 볼 필요가 있다. 이를 통해 ETRI 내 SW 개발 R&D 프로젝트의 소스 코드 품질 기준을 정하고, 소스 코드 품질 검증 환경과 체계를 구축하여 기본적으로 ETRI에서 개발되어 외부로 기술 이전하는 SW의 소스 코드 품질은 어느 정도 보증하는 환경을 확보할 필요가 있다.

2. 정적분석

SW 개발 주기에서 결함은 요구사항 정의 및 설계 등 문서를 작성하는 단계에서나 SW 구현의 세부 단계인 구현설계, 추상화, 의사코딩, 루틴작성, 방어코딩 등 코드를 작성하는 단계에서 인간의 오류로 발생하나, 근본적인 원인은 정해진 개발 시간의 압박, 복잡한 코드, 기반 환경의 복잡성, 기술이나 시스템의 변경, 그리고 수많은 시스템간 상호 연동 등의 이유로 발생한다. 그 외 온도 및 습도, 방사, 자기, 전자기장, 물리적 오염과 같은 환경적 요인도 하드웨어조건에 영향을 주어 소프트웨어 실행 오류 및 결함으로 나타날 수 있다. 이와 같은 결함은 SW 개발 주기의 각 단계에서 기술적 검토와 시험을 통해 검증하고 제거할 수 있으나 그렇다고 개발된 SW에서 결함이 전혀 없다고 100% 보증할 수는 없다. 또한, SW 개발의 구현 단계의 세부 과정인 코딩 작성 단계에서 발생하는 결함은 디버깅과 시험 그리고 코드 리뷰를 통해 제거할 수 있다. 이외에 결함을 줄이고 소스 코드 품질을 높이기 위해서 개발자는 코드 작성 시점에 함수 복잡도 수준, 함수 간 호출 및 연관관계 수준, 전역변수 수, 함수 및 모듈 수, 사용되지 않는 변수 및 함수 수준, 주석 수준, 그리고 코딩 표준 준수 수준 등 소스 코드 품질에 영향을 주는 요인 등을 고려하여 코드를 작성할 필요가 있다. 또 개발자가 작성한 소스 코드의 결함을 제거하기 위해 수행하는 코드 리뷰는 개발 SW가 크거나 복잡하지 않은 경우 수동적으로 동료 간

에 직접 검토도 가능하겠으나 대부분은 정적분석이라는 자동화 도구를 이용하여 진행한다.

가. 정적분석

정적분석(static analysis)은 프로그램 비실행을 전제로 자동화 도구를 활용하여 소스 코드를 Parsing하여 문법, 코딩규칙, 보안 취약점, 실행 오류 등 프로그램 특성을 분석하여 발생 가능한 소스 코드의 잠재적인 취약점과 결함을 찾아내는 방법[4]–[9]이다. 정적분석은 SW 개발 주기 상 코딩을 완료하여 개발된 프로그램을 실행하면서 결함을 찾아내는 동적 테스트를 수행하기에 앞서 상세설계 및 코딩을 진행하면서 프로그램을 실행하지 않은 상태에서 소스 코드와 모델에서 결함을 찾아내는 방법으로 비교적 개발 초기 결함을 찾아내어 전체 개발 수명 주기의 효율을 높여 개발 비용을 낮추는데 도움을 준다. 정적분석을 수행하는 도구는 기 정의된 코딩 규칙이나 표준을 준수하는지 확인하는 용도로 컴포넌트 혹은 SW 모듈 및 컴포넌트 통합 구현 단계에서 주로 개발자에 의해 사용되고, 모델링하는 동안에는 설계자에 의해 사용되며, SW 개발 완료 시점에는 개발 SW의 정해진 품질 목표에 대해 품질관리자나 테스터에 의해 개발된 소스 코드의 품질 검증을 확인하는 차원에서 사용된다. 정적분석의 가치는 동적테스트에서 발견하기 힘든 결함을 조기에 찾아내고, 개발된 소스 코드에서 코딩 복잡도, 모델 의존성, 불일치성 등을 분석하고 보안 취약성이나 버퍼 오버플로우(Buffer Overflow) 및 메모리 릭(Memory Leak)과 같은 실행 오류를 찾아내어 SW 개발자에게 경고 메시지를 알려주어 개발 소스 코드에서 잠재적 결함을 사전에 조치할 수 있도록 한다. 따라서 SW 개발 주기에서 자동화 도구를 이용하는 정적분석을 수행하므로 개발 효율성과 비용 절감을 유도할 수 있게 한다.

나. 정적분석 도구

현재 국내에서 대표적으로 사용하는 정적분석 상용 자동화 도구는 〈표 1〉와 같으며 대부분의 정적분석 상용 제품은 기본적으로 개발 소스의 코딩 스타일과 보완 취약점 점검 기능을 포함하고 있으며 프로그램 실행 오

〈표 1〉 정적분석 자동화 도구 I (상용)

도구명	제조사/라이선스	지원언어	특징	비고
Sparrow	파수닷컴	C/C++/C#/Java/JS/JSP 등	코딩 규칙, 보안 약점, 런타임 에러 등	CC 인증 제품
CodeInspector	슈어소프트	C/C++/Java 등	코딩 규칙 등	
Sofos Coding	한컴시큐어	C/C++/Java 등	보안 약점	CC 인증 제품
CodePrisom/Security Prisom	지티원	C/Java/JSP 등	보안 약점, 코딩 규칙	CC 인증 제품
Code Ray	트리니티소프트	C/Java 등	보안 약점, 형상 관리, 감사 기록 등	CC 인증 제품
BigLook	이븐스타	C/C++/Java/JSP/.net 등	코딩 규칙, 보안 약점 등	CC 인증 제품
HealingSecu ScanJ	사이버텍	Java/JSP 등	보안 약점, 실행 오류 등	CC 인증 제품
Resort	소프트4소프트	C/C++/Java	코딩 규칙 등	
Coverity	Coverity	C/C++	코딩 규칙, 실행 오류 등	

〈표 2〉 정적분석 자동화 도구 II (오픈소스)

도구명	제조사/라이선스	지원언어	특징	비고
PMD	BSD License	Java	코딩 규칙	http://pmd.sourceforge.net, https://pmd.github.io/
FindBugs	GNU Public License	Java	코딩 규칙	http://FindBugs.sourceforge.net
CheckStyle	GNU GPL	Java	코딩 규칙	http://CheckStyle.sourceforge.net
CppCheck	GNU Public License	C/C++	코딩 규칙	http://sourceforge.net/projects/cppcheck
Splint	Univ. of Virginia	C	보안 약점 코딩 규칙	https://sourceforge.net/projects/splint/, http://www.splint.org
Yasca	GNU Public License	C/C++/JS/PHP/ ASP 등	코딩 규칙 보안 약점 등	https://sourceforge.net/projects/yasca/

류 점검 기능과 나아가 false 오류 비율을 낮추는 노력을 기울이고 있다.

〈표 2〉는 오픈소스 기반 대표적인 정적분석 제품이다. 오픈 소스 기반 정적분석 제품이다 보니 상용 정적분석 제품에 비해 지원 언어나 정적분석 룰 셋이 다양하지 못한 어려움을 갖고 있으나 개발자가 정적분석 제품 도입 비용에 대한 제한을 받지 않고 원하는 개발 환경이나 룰 셋을 자유롭게 구축하여 활용할 수 있는 장점을 갖는다. 〈표 2〉에서 보는 바와 같이 오픈소스 기반 Java 소스에 대한 대표적인 정적분석 도구는 PMD나 FindBugs가 있고, C/C++를 지원하는 오픈소스 기반 대표적인 정적분석 도구는 Cppcheck가 있다.

이들 오픈소스 기반 정적분석 도구는 지원언어나 언어별 코딩 규칙 도구를 활용하는 과정에서 개발자나 QA가 직접 추가할 수 있는 편리함이 있다.

〈표 3〉 국내 주요 산업별 소스코드 코딩 규칙[13]

임베디드 코딩 표준

C Standards			
MISRA-C:2004	Automotive	141 Rules	ISO 26262 DO-178B/DO-278 IEC 62304/IEC 61508
MISRA-C:2012	Automotive	159 Rules	
JPL	Aerospace	23 Rules	
C++ Standards			
MISRA-C++:2008	Automotive	228 Rules	ISO 26262 DO-178B/DO-278 IEC 61508
JSF++ AV	Aerospace	231 Rules	
BSS C/C++	Aerospace	124 Rules	
Java Standards			
BSS Java	Aerospace	223 Rules	DO-178B/DO-278 IEC 61508
JPL	Aerospace	53 Rules	
무기체계 소프트웨어 코딩규칙 (C/C++)	방위사업청	65 Rules	C: 50 규칙 C++: 60 규칙

보안 코딩 표준

Secure Coding Standards			
CWE	Security	943 Rules	v 2.9
Juliet Test Suite for C/C++	Security	118 Rules	Vulnerability
Juliet Test Suite for Java	Security	112 Rules	Vulnerability
시큐어 코딩 가이드 (C/Java)	행정자치부	47 Rules	

실행 오류 검증 기준/CWE

Run-time Error		
무기체계 소프트웨어 실행 오류 (C/C++)	방위사업청	81 Rules (CWE-658, in C) 85 Rules (CWE-658, in C++) 74 Rules (CWE-658, in Java)

〈표 4〉 행정자치부 47개 보안 코딩 세부규칙[14]

구분	#	행자부2013 취약점 명칭	Java CWE ID	C, C++ CWE ID
입력 데이터 검증 및 표현	1	SQL 삽입	89	89
	2	경로 조작과 자원 삽입	99	99
	3	크로스 사이트 스크립트 (XSS)	79	79
	4	운영체제 명령어 삽입	78	78
	5	위험한 형식의 파일 업로드	434	(x)
	6	신뢰되지 않는 URL 주소로 자동 접속 연결	601	(x)
	7	XQuery 삽입	652	(x)
	8	XPath 삽입	643	(x)
	9	LDAP 삽입	90	90
	10	크로스사이트 요청 위조	352	(x)
	11	HTTP 응답 분할	113	(x)
	12	정수형 오버플로우	190	190
	13	보안 기능 결정에 사용되는 부적절한 입력값	807	(x)
	14	메모리 버퍼 오버플로우	(x)	119
	15	포맷 스트링 삽입	134	134
시간 및 상태	1	경쟁 조건: 검사시점과 사용시점 (TOCTOU)	367	367
	2	종료되지 않는 반복문 또는 재귀함수 (recursion)	674	674
에러 처리	1	오류 메시지 통한 정보 노출	209	209
	2	오류 상황 대응 부재	390	390
	3	부적절한 예외처리	754	754
코드 오류	1	널(Null) 포인터 역참조	476	476
	2	부적절한 자원 해제	404	404
	3	해제된 자원 사용	(x)	416
	4	초기화되지 않은 변수 사용	457	457
보안 기능	1	적절한 인증없 중요기능 허용	306	(x)
	2	부적절한 인가	285	285
	3	중요한 자원에 대한 잘못된 권한 허용	732	732
	4	취약한 암호화 알고리즘의 사용	327	327
	5	중요정보 평문 저장	312	312
	6	중요정보 평문 전송	319	319
	7	하드코드된 패스워드	259	259
	8	충분하지 않은 키 길이 사용	326	326
	9	적절하지 않은 난수값의 사용	330	330
	10	하드코드된 암호화키	321	(x)
	11	취약한 패스워드 허용	521	(x)
	12	사용자 하드디스크에 저장되는 쿠키를 통한 정보 노출	539	539
	13	주석문 안에 포함된 시스템 주요정보	615	615
	14	솔트 없이 일방향 해쉬 함수 사용	759	759
	15	무결성 검사없는 코드 다운로드	494	(x)
	16	반복된 인증시도 제한 기능 부재	307	307
캡슐화	1	잘못된 세션에 의한 데이터 정보 노출	488	488
	2	제거되지 않고 남은 디버거 코드	489	489
	3	시스템 데이터 정보 노출	497	497
	4	Public 메소드부터 반환된 Private 배열	495	495
	5	Private 배열 public 데이터 할당	496	496
API 오용	1	DNS lookup에 의존한 보안결정	247	247
	2	취약한 API 사용	676	676

〈표 1〉과 〈표 2〉에 제시된 정적분석 도구는 Jenkins와 연계하여 CI(Continuios Integration) 환경을 구축하여 활용할 수 있으며, 그 외 Junit와 같은 동적 시험 도구나 SVN 혹은 Git과 같은 형상관리 도구도 CI와 통합한 환경을 구축하여 활용할 수 있는 특징을 갖는다.

그 외 〈표 3〉은 국내 주요 산업별 소스 코드에 대한 코딩 규칙이고 〈표 4〉는 행정자치부에서 권장하고 있는 SW 개발 시 소스 코드에 대한 47개 보안 코딩 세부 규칙 내용이다.

〈표 3〉에서 보는 바와 같이 항공 우주, 자동차, 방위산업, 그리고 보안 제품과 같은 SW 개발에서는 보다 강력한 품질 기준과 보다 넓은 코딩 규칙을 요구하고 있다. 개발하는 모든 SW에서 보안 취약점 검증을 위해 행정자치부에서 제시한 47가지 보안 코딩 세부 규칙은 〈표 4〉에서 보는 바와 같이 SQL(Structured Query Language) 삽입, 경로조작 및 자원 삽입, 크로스사이트 스크립트 등 입력데이터 표현 및 검증 관련 15개, 부적절한 인가, 중요정보 평문저장, 하드코드된 비밀번호 등 보안 기능 관련 16개, 경쟁조건 즉, 검사 시점과 사용 시점 혹은 종료되지 않는 반복문 또는 재귀 함수 등 시간 및 상태 관련 2개, 오류상황 대응 부재, 오류메시지를 통한 정보노출 등 에러처리 관련 3개, 제거되지 않고 남은 디버거 코드, 시스템 데이터 정보노출 등 캡슐화 관련 5개, DNS(Domain Network Service) lookup에 의존한 보안결정 등 API 오용 관련 2개, 널 포인터 역참조, 부적절한 자원 해제 등 코드 오류 관련 2개 등이 있다.

Ⅲ. SW 개발 R&D 프로젝트에서 정적분석 실행 방안

현재 수행하고 있는 SW 개발 모든 R&D 프로젝트를 대상으로 정적분석 검증을 당장 실행하는 것은 제도적 및 환경적으로 준비되지 않은 연구개발 현장에 여러 가지 복잡한 상황을 가중할 수 있다. 그렇다고 SW 개발 R&D 프로젝트에서 개발 SW의 소스 코드에 대해 정적분석을 실행하지 않는 것도 개발 SW의 내재적 위험을 가중할 뿐 아니라 개발 기술 이전에 따른 고객 만족도에도 상당한 영향을 가져올 수 있다. 이에 본 장에서는 SW 개발 R&D 프로젝트에서 개발 기술의 소스 코드 품질 관리를 위한 정적분석 검증 대상 과제와 적용 기준 및 실행 방안에 대해 기술한다.

1. 적용 기준

현재 수행 중이거나 앞으로 수행 예정인 SW 개발 R&D 프로젝트에서 정적분석 검증을 실행하는 과제 대상과 정적분석 검증 기준은 다음과 같이 고려할 수 있다.

가. 검증 대상

먼저 개발 SW의 소스 코드 품질보다는 기술 개발 가능성 및 타당성 검증 차원(TRL 4이하)에서 수행하는 SW 개발 R&D 프로젝트에서 개발 소스 코드에 대해 정적분석을 수행하는 것은 불필요할 수 있을 것이다. 다만 개발 기술 결과를 업체에 기술 이전하는 경우나 TRL 5 이상의 SW 개발 R&D 프로젝트에서는 개발 SW의 소스 코드에 대한 정적분석을 수행하는 것은 필요하다. 이에 SW 개발 R&D 프로젝트에 다음과 같은 분류에 포함되는 경우는 개발 SW 소스 코드에 대해 정적분석을 수행할 것을 권장한다.

- TRL(Technology Readiness Level) 5 이상의 SW 개발 R&D 프로젝트
- 기술이전 대상 SW 개발 R&D 프로젝트
- SW Bank 등록 대상 과제
- GCS(Global Challenge SW)/WBS(World Best SW) 과제 등
- 기타 SW 개발 R&D 프로젝트 중 PM이 정적분석 검증이 필요하다고 인정하는 과제 등

나. 검증 수행 지원 언어

현재 연구원 차원에서 모든 SW 개발 언어에 대한 정적분석 검증 지원 환경이 갖추어져 있는 상태가 아니기 때문에 현재 일부 준비된 환경을 고려하여 SW 개발 R&D 프로젝트에서 정적분석 검증 지원 언어는 C/C++/C#, 그리고 Java/JS/JSP/ASP를 기본으로 하고, 그 외 언어로 개발한 소스 코드는 정적분석 검증 지원 환경 구축 보완을 통해 점진적으로 포함하도록 한다.

다. 검증 수행 시기

개발 SW 소스 코드에 대한 정적분석 수행은 구현 및 시험 단계에서는 개발자에 의해 진행하며 기술 개발이 완료되는 기술 이전 및 사업 Q-mark 단계에서는 개발 경험이 풍부한 시험자나 품질담당자에 의해 정적분석 검증을 수행하므로 SW 개발 R&D 프로젝트의 코드 품질에 대한 제2자 검증 체계가 가능하도록 한다.

라. 검증 기준

SW 개발 R&D 프로젝트에서 개발 소스의 정적분석검증 기준은 국방 SW나 안전성이 요구되는 SW 소스 코드 품질 수준과는 다를 것이다. 일반적으로 소스 코드 복잡도를 표현할 때 간단함 함수만을 사용해 작성한 코드(복잡도: 1~4), 잘 구조화되고 안정적인 함수를 사용해 작성한 코드(복잡도: 5~10), 비교적 복잡한 함수를 사용해 작성한 코드(복잡도: 11~20), 복잡한 함수를 사용해 작성한 코드(복잡도: 21~50), 그리고 극단적으로 복잡하고 문제된 함수를 사용해 작성한 코드(복잡도: 50 이상)로 분리하는데 개발 소스 코드의 복잡도에 따라 개발 SW의 품질 관리는 다르다. 또한, 소스 코드에서 사용되지 않은 함수 혹은 변수나 이런 함수 및 변수의 정의 및 사용 개수에 따라 소스 코드 품질 수준은 다르다. 이에 SW 개발 R&D 프로젝트에서 코드 품질을 고려해 시행하는 정적분석 검증 수준을 다음과 같이 고려해 볼 수 있다.

- 중요 코딩 규칙 준수율(치명적 코딩 규칙 위반율, 코딩 준수율에 따라 등급 적용)
- 중요 보안 규칙 준수율(치명적 보안 취약점 위반율)
- 실행 오류 발생율(치명적 실행 오류 발생율)
- 코드 중복도 준수율(코드 복사 및 반복 사용율)
- 함수 복잡도 수준
- 사용되지 않는 함수/변수 수준(사용하지 않는 함수 혹은 변수에 따라 등급 적용)
- 그 외 주석율 수준

이와 같은 SW 코드 품질 검증 기준을 SW 개발 R&D 조직에 도입 및 적용하기 위해서는 먼저 조직 환경에 맞는 적절한 코딩 규칙 준비와 코드품질 교육 프로그램 그리고 코딩 규칙을 기반한 세부적인 정적분석 검증 기준과 실행 방안 준비를 통해 사전에 개발자를 포함한 조직 구성원들의 의견 수렴과 시범 적용을 통해 점진적으로 확대해 나가도록 해야 할 것이다.

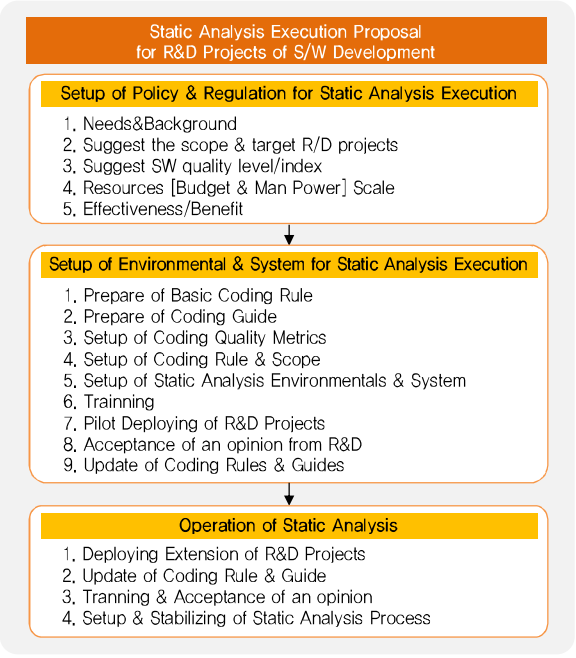
2. 적용 모델

솔루션을 개발하는 프로젝트의 경우 그리고 국방이나 안전 및 신뢰성이 강조되는 SW 제품을 개발하는 조직에서는 개발 SW의 소스 코드 품질이 조직이나 회사의 매출과 수입에 직접 영향을 주기 때문에 정적분석 검증 수준이 높게 수행되고 있다. 그러나 제품 개발이 아닌 원천 기술 개발 혹은 시제품 개발 위주의 R&D 프로젝트를 수행하는 조직에서 정적분석 검증 필요성은 실효성 측면이나 수행 효과 측면에서 크지 않을 수 있다. 그럼에도 SW Bank 등록 대상 과제나 GSC 과제와 같은 R&D 과제는 과제 계획 수립 단계부터 과제 수행에 필요한 프로젝트관리 계획 혹은 품질관리 계획을 요구하

고 있고, 이 품질관리계획에는 소스 코드 품질 관리를 위한 정적분석 검증 수행을 요구하고 있다. 더욱이 SW 개발 R&D 프로젝트 결과를 기업에 기술 이전하는 경우 개발 소스의 코드 품질 유지는 R&D 수행 결과에 대한 고객 만족 증진은 물론 고객과의 기본적인 신뢰 관계 유지를 위해서도 필요하다. 그리고 개발 SW의 소스 코드 품질 유지를 위해 필요한 정적분석 검증 수행을 조직과 프로젝트에 적용하기 위해서는 먼저 조직 내 공감대 형성과 필요한 정보 공유가 필요하고 동시에 정적분석 검증 수행을 위한 제도적 지원 체계와 환경적 구축 체계가 필요하다. 따라서 SW 개발 R&D 프로젝트의 소스 코드 품질 관리를 위해 요구되는 정적분석 검증 실행 모델에는 정적분석 검증을 위한 정책 등이 포함된 실행 방안, 검증 수행 프로세스 그리고 정적분석 검증 지원 환경 구축 체계를 포함한다.

가. 정적분석 검증 실행 방안

(그림 1)은 SW 개발 R&D 프로젝트 수행 조직에서 정적분석 검증 수행 필요한 정책 등을 고려한 정적분석 검증 실행 방안이다.

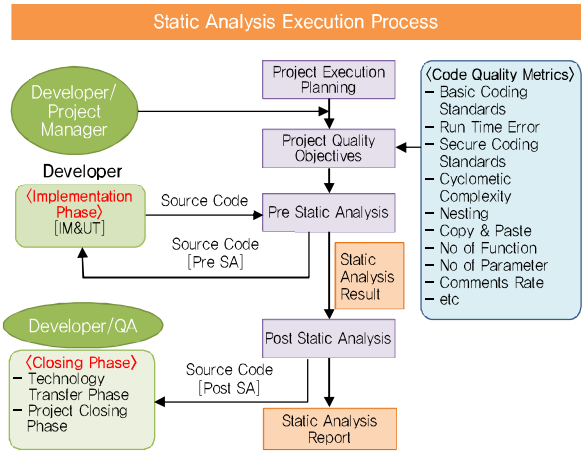


(그림 1) 정적분석 검증 실행 방안

(그림 1)과 같은 정적분석 검증을 이행하기 위해서는 먼저 개발 SW 소스 코드에 대한 정적분석 검증 필요성과 목적에 대한 조직 내 공감대 형성이 필요하며 동시에 SW 개발 R&D 프로젝트에서의 코드 품질 검증 항목과 코드 품질 수준, 그리고 이를 고려한 소스 코드 품질 정책을 수립할 필요가 있다. 또한, 조직 내 코딩 규칙 가이드를 준비하여 제공하고, 코딩 규칙 교육과 공유를 통해 코딩 룰 적용 범위와 수준에 대한 의견 수렴을 진행하여 조직과 환경에 적절한 소스코딩 규칙을 커스터마이징하고 이에 맞는 정적분석 지원 환경 구축이 필요하다. 또한, 정적분석 도구를 준비하고 도입하는 과정에서도 조직 내 의견 수렴과 환경을 고려하여 정적분석 도구를 선정 및 도입하고, 도구 도입 후 정적분석 검증을 실행하는 과정에서도 당장 강행하지 않는 코딩 규칙은 비활성화를 유도하고, 너무 많은 코딩 규칙의 적용보다는 작지만 의미있는 코딩 규칙부터 적용할 수 있도록 환경적 설정을 통해 운영 및 지원할 필요가 있다. 또 정적분석 도구가 갖는 자체 결함이나 Critical 결함의 미해결도 존재함을 인식시키고, 반복 및 중복 발생하는 결함에 대한 지속적인 교육과 공지를 통해 SW 개발에 대한 정적분석 검증 실행이 조직 내 자연스런 프로세스화로 정착될 수 있도록 해야 한다. 다만, 앞서 언급한 바와 같이 SW 개발 R&D 프로젝트의 특성을 감안하여 개발 소스 코드에 대한 정적분석 검증 항목이나 수준을 신뢰성 높은 SW 개발 제품이나 국방 품질 수준 보다는 기술 개발의 검증 차원과 개발 기술에 대한 업체 이전에 필요한 수준으로 진행하는 것이 보다 합리적일 것으로 고려된다.

나. 정적분석 검증 프로세스

(그림 2)는 SW 개발 시 소스 코드에 대해 정적분석 검증을 위한 수행 프로세스이다. 개발자는 구현 및 시험

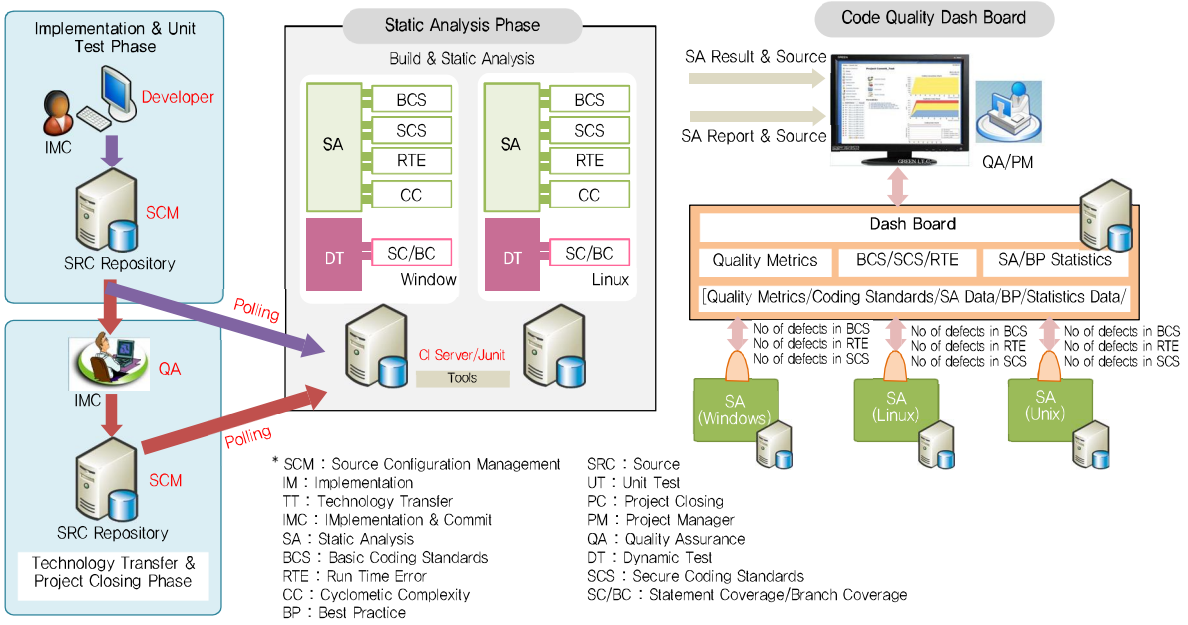


(그림 2) 정적분석 검증 프로세스

단계에서 개발하는 소스 코드에 대해 정적분석을 수행하고 개발 완료 시 수행한 정적분석 검증 결과서를 준비한다. 이어 개발한 기술을 기업에 이전하는 시점에 품질 담당자나 시험자는 개발 완료한 소스 코드에 대해 다시 한번 정적분석 검증을 수행한 후 정적분석 검증보고서를 준비하여 기술 이전에 필요한 절차를 진행한다. 또 정적분석 검증을 실행한 소스는 시스템에 등록하고 정적분석 검증 결과는 CI 환경을 통해 개발자와 품질담당자가 모두 모니터링할 수 있도록 하고 나아가 정적분석 검증에 대한 통계 분석 기초 자료로 활용할 수 있도록 한다.

다. 정적분석 검증 지원 환경 구축 체계

(그림 2)와 같은 정적분석 검증 수행 프로세스를 지원하기 위한 정적분석 검증 환경 구축 체계가 (그림 3)이며, 정적분석 검증 프로세스 세부 활동을 실행하기 위한 도구와 구축 내용을 포함한다. 따라서 SW 개발 R&D 프로젝트에 대한 정적분석 검증 실행을 위해 조직에서는 점진적으로 (그림 3)과 같은 정적분석 검증 수행 지원 환경을 구축해 나가야 한다. 결론적으로 SW 개발 R&D 프로젝트에서 정적분석 검증 실행 모델은 앞서 기술한 정적분석 검증 대상과 지원 언어, 정적분석 검증 시기와 정적분석 검증 수행자, 정적분석 검증 지표를 포함한 검증 기준, 조직 내 품질 정책을 고려한 정적분석 검증 실행 방안과 검증 프로세스, 그리고 이와 같은 정적분석 검증을 실행할 수 있는 환경적 지원 체계 구축을 모두 포함하는 것이다.



(그림 3) 정적분석 검증 지원 환경 구축 체계

Ⅳ. 적용 사례 분석

1. 시범 적용 대상

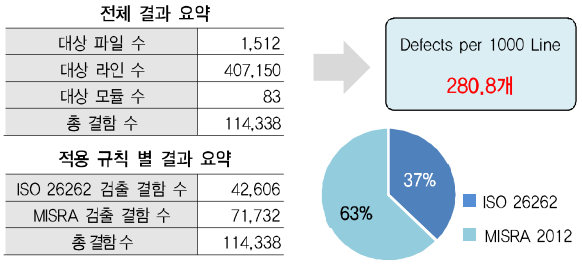
본 연구에서 제시한 소스 코드 품질 관리를 위한 정적분석 검증 실행 모델을 시범적으로 적용하기 위해 준비한 것은 C 언어를 이용해 개발한 윈도우용 OS이다. 개발한 OS 소스 크기는 약 40만 라인 정도며 개발한 소스 코드의 정적분석 검증을 실행하기 위한 사용한 도구는 CodeInspector(상용 도구)이고, 적용한 코딩 규칙은 MISRA2012의 121개 규칙과 ISO26262의 10개 규칙이다[15][16].

2. 시범 적용 환경 및 절차

시범적으로 진행한 정적분석 검증 실행 환경과 절차는 (그림 4)와 같으며 이는 앞서 정적분석 검증 실행 지원 구축 환경 체계로 제시한 (그림 3)의 정적분석 검증 지원 환경 구축 체계의 일부로 이해할 수 있고, 정적분석 검증 수행은 (그림 2)의 정적분석 검증 프로세스 절차를 따르고 있다.

3. 시범 적용 검증 결과

본 연구에서 SW 개발 R&D 프로젝트에서 코드 품질 관리를 위해 제시한 정적분석 검증 실행 방안에 대한 시범 적용 차원에서 사용한 정적분석 검증 도구는 Code-Inspector이고, 코딩 규칙으로는 MISRA2012와 ISO26262을 적용했다. 시범 검증대상으로 C 언어로 개발한 40만 라인 정도의 개발 소스에 대해 정적분석 검증을 수행한 결과 (그림 5)와 같이 코딩 스타일과 같은 사소한 결함



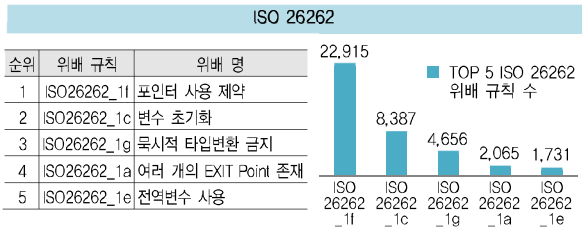
전체 결과 요약

대상 파일 수	1,512
대상 라인 수	407,150
대상 모듈 수	83
총 결함 수	114,338

적용 규칙 별 결과 요약

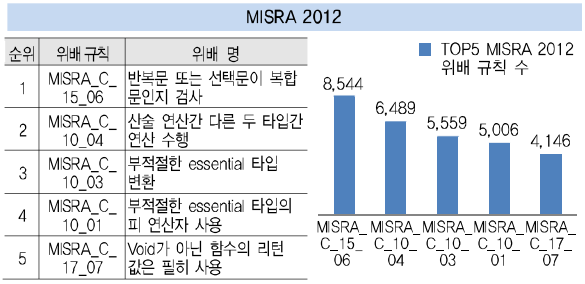
ISO 26262 검출 결함 수	42,606
MISRA 검출 결함 수	71,732
총 결함 수	114,338

(a) 정적분석 검증 실행 결과 요약



ISO 26262

순위	위배 규칙	위배 명
1	ISO26262_1f	포인터 사용 제약
2	ISO26262_1c	변수 초기화
3	ISO26262_1g	묵시적 타입변환 금지
4	ISO26262_1a	여러 개의 EXIT Point 존재
5	ISO26262_1e	전역변수 사용

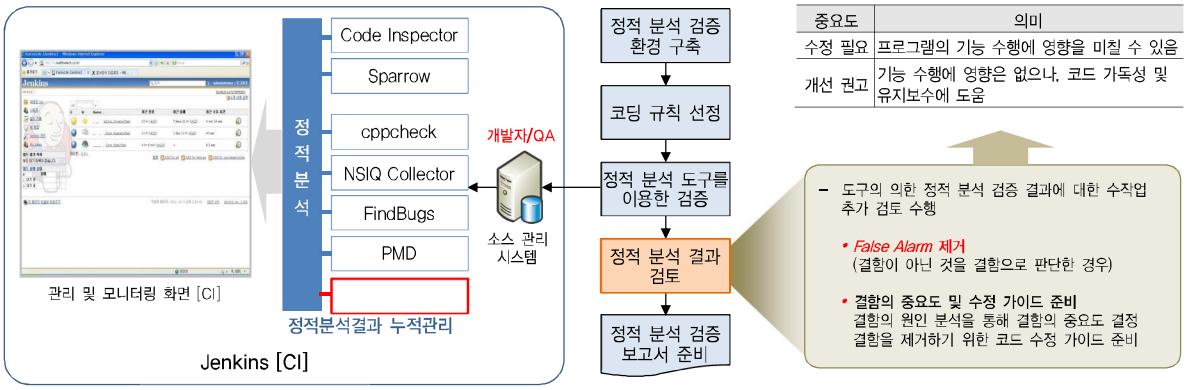


MISRA 2012

순위	위배 규칙	위배 명
1	MISRA_C_15_06	반복문 또는 선택문이 복합문인지 검사
2	MISRA_C_10_04	산술 연산간 다른 두 타입간 연산 수행
3	MISRA_C_10_03	부적절한 essential 타입 변환
4	MISRA_C_10_01	부적절한 essential 타입의 피 연산자 사용
5	MISRA_C_17_07	Void가 아닌 함수의 리턴값은 필히 사용

(b) 적용코딩 규칙별 위배 요약

(그림 5) 정적분석 검증 시범 적용 결과



(그림 4) 정적분석 검증 시범 환경과 절차

제목	여러 개의 exit point [ISO26262_1a]
설명	• 함수는 하나의 exit point를 가져야 함 • 두개 이상의 RETURN 구문을 사용 할 경우 코드의 흐름추적이 용이하지 않아 결함 발생 가능성이 높아질 수 있음.

파일명	함수명	라인
rtems-4.10.99-src₩c₩src₩lib ₩libbsp₩shared₩console.c	console_open	114

위배 코드

```
rtems_device_driver console_open
 ( rtems_device_major_number major,
 rtems_device_minor_number minor,
 void                      * arg
 )
{
  ...
  If(minor 〉 Console_Port_Count) {
   return   RTEMS_INVALID_NUMBER;  /*
   Not compliant */
  }

  ...

  return status; /* Not compliant */
  }
```

권고 코드

```
rtems_device_driver console_open
 ( rtems_device_major_number major,
 rtems_device_minor_number minor,
 void                      * arg
 )
{
  ...
  int status;

  If(minor 〉 Console_Port_Count) { st
    atus = RTEMS_INVALID_NUMBER
  }

  ...

  return status; /* Compliant */
}
```

하나의 함수 내에서 두 개 이상의 리턴 구문을 가지고 있음.
코드의 흐름추적이 용이하지 않아 결함 발생 가능성이 높아짐

(a) ISO26262 규칙의 대표 결함

제목	반복문이나 선택문이 복합문인지 검사 MISRA_C2012_15_06
설명	• 반복문이나 선택문의 몸체를 괄호 없이 문장만을 나열하는 경우 표현식 다음에 ";"이 포함되면 null 문장으로 연결되므로 위험함 • 엉뚱한 if문과 else문을 연결해서 가독 할 수 있음

파일명	함수명	라인
rtems-4.10.99-src₩tool₩cpu₩nios2₩ptf.c	prf-free	156

위배 코드

```
Void ptf_free(struct ptf *ptf)
{
  ...
  If(this-〉value != NULL) free(this-〉
  value); /* Not complant*/
  ...
  }
}
```

권고 코드

```
Void ptf_free(struct ptf *ptf)
{
  ...
  If(this-〉value != NULL)
  {
    free(this-〉 value); /* Complant*/
  }
}
```

(b) MISRA 2012 규칙의 대표 결함

(그림 6) 정적분석 검증 시범 적용 대표 결함 내용

부터 변수 타입 선언과 변수 사용 불일치 그리고 포인터 사용에 따른 제약 내용 등 중요한 결함 등이 추출되었다. 이중 MISRA 2012 코딩 규칙의 위배 결함은 모두 7만개, 그리고 ISO 26262 코딩 규칙의 위배 결함이 4만개 등으로 개발한 소스에서 총 11만개의 크고 작은 코딩 결함을 확인할 수 있었다. 시범적용 결과에서 알 수 있듯이 소스 코드 품질 관리를 위해 준비하는 정적분석 검증 도구와 코딩 규칙은 도입 및 운영에 앞서 조직 환경에 맞는 코딩 규칙과 도구 선정을 위한 사전 검토와 조직 내 의견 수렴 과정이 필요하다. 또한, 실제 정적분석 검증 실행에 앞서 의미있는 결함만을 찾아낼 수 있는 코딩 규칙 단순화 및 세분화 과정이 필요하고 정적분석 검증 지원 환경 구축 체계의 안정적 도입과 운영을 위한 조직 내 도움과 협조도 요구된다.

(그림 6)은 정적분석을 시범적으로 적용하여 찾아낸 약 11만개 결함 중 MISRA 2012와 ISO26262의 구체적인 대표 결함 내용이다. ISO26262 규칙 위배로 찾아낸 결함은 함수에서 하나 이상의 exit point가 있어 함수가 정상적인 동작을 수행하지 못하는 결함이고, MISRA 2012 규칙 위배로 찾아낸 결함은 괄호 사용 없이 반복문 혹은 선택문을 사용한 경우 사용한 문장이 다른 문장과 결합해서 동작할 수 있는 결함이다.

4. 시사점

현재 수행 중인 SW 개발 R&D 프로젝트에서 소스 코드의 품질 수준을 확인하기 위해 제안된 정적분석 검증 실행 모델을 시범적으로 적용하였다. 시범적으로 진행한 개발 소스 코드의 정적분석 검증 결과를 보면 SW 개발 R&D 프로젝트에서 개발 기술의 소스 코드 품질 수준을 대내외적으로 수용할 수 있는 좀더 객관적인 정적분석 검증 지침이 범위가 필요하다고 판단된다. 또한, SW 개발 R&D 프로젝트에서 개발 기술의 소스 코드에 대해 정적분석 검증을 적용하기에 앞서 먼저 R&D 품질 목표 및 소스 코드 품질 수준을 수립하고, 수행하는 R&D 프로젝트 특성에 따라 소스 코드 품질 지표와 적용 범위를 정의하는 것이 필요하다. 아울러 앞서 기술한 바와 같이 SW 개발 R&D 프로젝트에서 소스 코드 품질 관리를 위해서는 먼저 의미 있고 유용한 코딩 기본 규칙을 수립하고 개발 현장과 공유하며, 이를 기반한 조직 내 SW 개발 코딩 준수 가이드를 준비하여 필요한 교육을 진행해야 한다. 처음부터 너무 많은 코딩 규칙은 지양하고 당장 강행하지 않은 코딩 규칙은 비 활성화하며 가능한 핵심적이고 꼭 필요한 몇 가지 의미 있는 규칙부터 적용할 수 있도록 유도해야 한다. 이와 같이 작지만 의미 있는 규칙을 시작으로 연구개발 현장에 적용을 통

해 의견 수렴하면서 점진적으로 연구현장에 맞게 코딩 규칙을 조정해 나가야 한다. 또 다양한 개발 언어와 개발 기술의 수준에 따라 개발 환경에 맞도록 적절한 코딩 규칙과 적용 기준 및 범위를 점진적으로 조정 및 보완해 가야 한다. 정적분석 도구 도입시에도 가능한 개발자와 연구현장 의견을 수렴하여 R&D 조직 환경을 고려한 도구가 도입될 수 있도록 하고, 도구 도입 후 적절한 교육과 운영을 통해 점진적으로 확산 적용 가능하도록 해야 한다. 실제 개발 소스 코드 품질을 검증하기 위해 코딩 규칙과 툴을 운영하는 과정에서도 도구 자체의 결함이나 Critical 결함의 미해결도 개발 소스에 존재함을 인식하도록 지속적으로 교육하고, 나아가 반복 및 중복 발생하는 결함에 대해서도 시스템적으로 DB화하고 이를 연구개발자와 연구현장에 지속적으로 공유하도록 해야 본 연구에서 제시된 SW 개발 R&D 프로젝트의 소스 코드 품질 유지를 위한 정적분석 검증 모델이 성공적으로 도입 및 운영 가능할 것이다. 또 제시된 정적분석 검증 모델이 현재 수행 중인 다양한 SW 개발 R&D 프로젝트를 대상으로 소스 코드 정적분석 검증을 진행하지는 않았으나 SW 개발 R&D 프로젝트에서 소스 코드 품질을 위한 정적분석 검증 모델로 운영하여 R&D 프로젝트의 소스 코드 품질 수준을 유지하기 위한 하나의 대안으로 고려할 수 있을 것이다.

V. 결론

SW 개발 R&D 프로젝트에서 개발 기술 및 소스 코드의 품질은 어느 정도 수준이 필요한가? 단순히 기술 개발의 검증 차원에서 수행하는 R&D 프로젝트에서 개발 기술과 소스 코드의 품질은 제품화 혹은 상용화를 기대하는 개발 기술과 같은 소스 코드 품질 수준을 요구할 필요는 없을 것이다. 그렇다고 개발한 기술을 기업의 요구에 의해 기술이전 하는 R&D 프로젝트에서 개발 기술 및 소스 코드 품질은 적어도 기술 이전 업체에서 문제없이 활용할 수 있는 정도는 유지되어야 할 것이다. 또 R&D 과제 규모나 과제 타입 혹은 개발 기술의 수준 등 아주 다양한 R&D 프로젝트에서 개발 기술이나 소스 코드의 품질을 일률적으로 정해서 관리하는 것도 R&D 프로젝트를 수행하는 개발자에게 불필요한 부담을 줄 수 있다. 이에 본 연구에서는 SW 개발 R&D 프로젝트에 개발 소스 품질 검증을 위한 정적분석 실행 모델을 제시하였다. 제시된 모델은 SW 개발 R&D 프로젝트에서 개발 소스 코드의 품질을 관리할 수 있는 사항을 포함하도록 먼저, 소스 코드 품질을 유지해야 하는 R&D 프로젝트 대상과 범위, 소스 코드 품질 지표와 기준 포함하며, 조직에서 SW 개발 R&D 프로젝트에서 소스 코드 품질 관리를 수행할 수 있는 정적분석 검증 실행 방안과 검증 프로세스 그리고 정정분석 검증 지원 환경 구축 체계를 제시했다.

제시된 정적분석 실행 모델을 근간하여 현재 수행 중인 R&D 프로젝트에서 개발한 소스 코드에 대해 시범적으로 정적분석 검증을 수행했고 그 결과를 통해 SW 개발 R&D 프로젝트에서 개발 소스 코드 품질 검증을 위한 정적분석 실행 모델의 안정적인 도입 및 운영 방안을 시사점으로 기술하였다.

따라서 본 연구에서 제시된 개발 소스 코드 품질 검증을 위한 정적분석 실행 모델은 SW 개발 R&D 프로젝트의 개발 기술과 소스 코드 품질 관리 및 유지를 위해 활용할 수 있을 것으로 기대한다.

약어 정리

ASP	Active Server Page
BCS	Basic Coding Standards
BP	Best Practice
CC	Common Criteria
CC	Cyclometic Complexity
CDMA	Code Division Multiple Access
CMMI	Capability Maturity Model Integration (Software Engineering Institute, Carnegie

	Mellon University, Pittsburgh)
DT	Dynamic Test
ETRI	Electronics & Telecommunications Research Institute
GCS	Global Challenge SW
IM	Implementation
IMC	IMplementation & Commit
ISO	the International Organization for Standardization
IT	Information Technology
JS	Java Script
JSP	Java Server Page
NIPA	National IT Industry Promotion Agency
OSS	Open Source Software
PC	Project Closing
PMBoK	A Guide to the Project Management Body of Knowledge
QA	Quality Assurance
R&D	Research & Development
RTE	Run Time Error
SA	Static Analysis
SC/BC	Statement Coverage/Branch Coverage
SCM	Source Configuration Management
SCS	Secure Coding Standards
SoC	System on Chip
SP	Software Process
SPICE	Software Process Improvement and Capability dEtermination (ISO15504)
SRC	Source Repository
SVN	Subversion
SW	Software
TDX	Time Division Exchange
TRL	Technology Readiness Level
TT	Technology Transfer
UT	Unit Test
WBS	World Best SW

참고문헌

[1] 박정현 외 3인, "R&D 품질관리를 위한 Q-mark 개선 방향," 한국품질경영학회 추계학술대회, 2014.

[2] J.-H. Park et al., Q-mark Model for Quality of R&D Projects, FGCN2014, SERSC Proceeding. 2014.

[3] 서우역 번역(스티브맥코넬), "Code Complete," 정보문화사, 2005. 4.

[4] 테스트마이다스, "품질관리 개선," ETRI Seminar, 2015. 2.

[5] TTA, "14차 SW 테스트전문가 양성교육-일반과정," 2015. 8.

[6] 여지영(석사논문), "소스 코드의 지속적 통합을 통한 검색엔진 코드 품질과 결함율 개선을 위한 연구," 숭실대학교, 2012.

[7] 정영범, "코딩기술과 정적분석," 파수닷컴, 2015.

[8] 테스트마이다스, "품질보증개요," KAIST 전문 교육, 2014. 6.

[9] 테스트마이다스, "CI 서버를 이용한 품질 관리 방안," KAIST 전문 교육, 2014. 6.

[10] NIPA, "SW 개발 품질관리 매뉴얼," 2013. 12.

[11] NIPA, "공개소프트웨어를 활용한 정적분석," GCS School, 2015. 5.

[12] NIPA, "CI 구축 및 적용," GCS School, 2015. 5.

[13] ㈜소프트4소프트, "정적 분석 & 코딩 표준, 코드 점검 (RESORT) 프로세스 소개," 2016.

[14] OpenEG (김영숙), "소프트웨어 개발 보안," 파수닷컴 원포인트 보안세미나, 2015.

[15] 박정현, "CodeInspector 기반 정적분석 구축 환경과 적용 사례," ETRI, 2016. 9.

[16] 박정현, "코드 기반 정적 시험 및 품질 검증 시스템 구축," ETRI, 2015.