Pusan National University Computer

Science and Engineering Technical Report

2023-10

소스코드 기반 소프트웨어 품질 측정 방법 및 도구 개발

201624567 전민기

201724492 성민우

201524593 천동혁

지도교수 채흥석

목차

1. 서론

1.1. 연구 배경

갈수록 자동차, 군사, 우주항공, 의료와 같은 다양한 분야에서 소프트웨어 사용이 증가하고 있다. 이런 분야에서는 소프트웨어의 작은 결함이 엄청난 피해를 야기시킬 수 있다. 실제사례로는 2009년 도요타의 렉서스 자동차가 전자제어장치(ECU)에 내장된 SW의 결함 때문에 급 발진하여 탑승자 전원이 사망하였다. 이 사고로 인해 900만 대의 차량이 리콜 되며 5조 원 이상의 경제적 손실을 입은 것으로 알려져 있다[1]. 또 받음각(AOA) 센서의 SW 오류로 보잉 737 MAX가 추락하여 총 346명이 사망하였다[2].

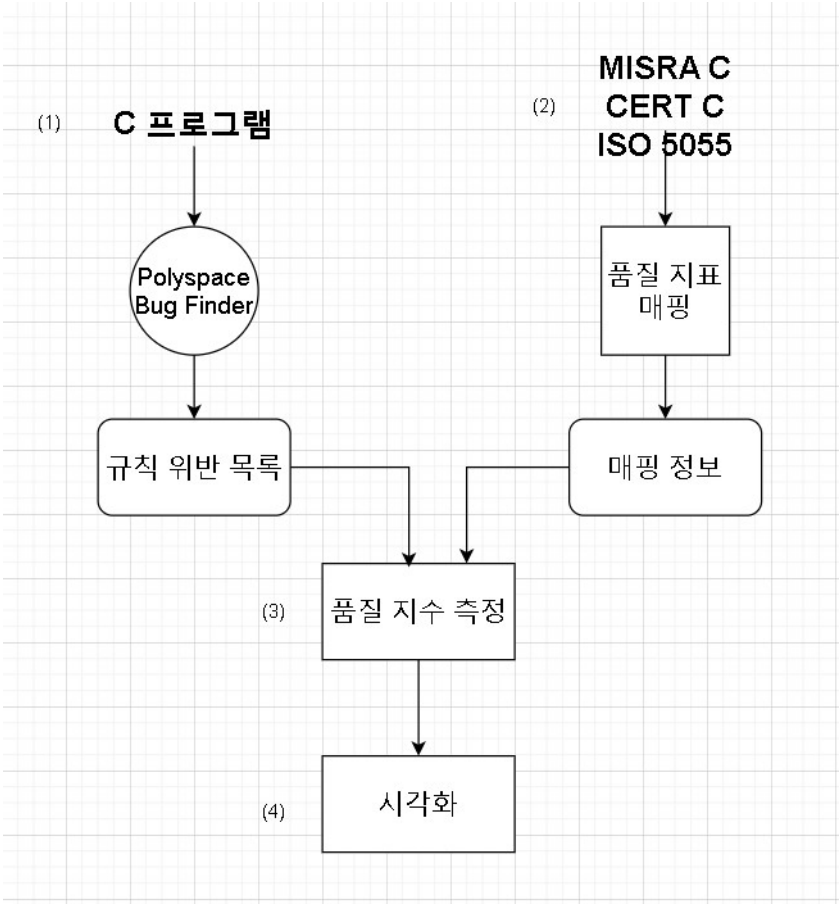
이처럼 소스 코드의 작은 결함이 엄청난 피해를 발생시킬 수 있고, 그렇기 때문에 소스 코드에 대한 철저한 분석이 필요하다. 대표적으로 정적 분석 기법이 있다. 정적 분석 기법이란 소스 코드의 실행 없이, 코드의 의미를 분석해 결함을 찾아내는 코드 분석 기법이다. 대표적인 정적 분석 도구인 Polyspace를 이용해 과제를 수행하고자 한다.

1.2. 기존 문제점

소프트웨어 품질 향상을 위해 표준화된 정량적인 소프트웨어 품질 측정 방법이 필요하다. 소프트웨어의 품질을 정량적으로 측정함으로써, 관리자에게 개발 중인 소프트웨어의 품질 수준을 알려주고, 개발자에게 품질 수준이 낮은 모듈을 개선하도록 요구할 수 있다. 그러나 기존의 소프트웨어 품질 측정 방법은 다양한 품질 속성에 대해 적용되지 않거나 표준에 기반하지 않고 인스펙션과 같은 개발자의 역량에 의존적인 방법을 사용하는 경우들이 있었다.

1.3. 연구 목표

본 졸업 과제에서는 C 소프트웨어의 소스코드를 기반으로 소프트웨어의 품질을 측정하는 소프트웨어를 개발하는 것을 목표로 한다. 소프트웨어의 품질을 자동으로 측정하기 위해 코딩 표준을 분석하여 표준에 정의된 코딩 규칙과 소프트웨어 품질 속성 간의 관계를 분석한다. 그리고 기존의 정적 분석 도구를 이용해 코딩 표준 위반 정보를 수집하고 수집된 위반 정보를 기반으로 자동으로 품질 속성을 측정한다. 정리하면 다음과 같다.



연구 흐름도

(1) Polyspace 프로그램을 이용하여 C 소스코드의 정적 분석을 수행한다.

(2) MISRA-C, CERT C, ISO 5055를 활용하여 품질 요구사항과 코딩규칙을 매핑한다.

(3) 위의 두 결과를 이용하여 품질 지수 측정 계산법을 개발한다.

(4) 점수를 시각화하기 위한 프로그램을 만든다.

2. 연구 배경

2.1. 정적 분석

정적 분석이란 코드를 실행하지 않고 수행되는 품질, 안정성, 보안을 위해 소스코드를 분석하는 소프트웨어 검증이다. 이는 동적분석을 보완할 수 있는 몇 가지 이점을 가지고 있는데 다음과 같다.

1. 오류 감지

동시성, 오염된 데이터, 데이터 흐름, 보안, 정적 및 동적 메모리와 관련된 여러 결함을 검출할 수 있다. 일부 결함은 동적 테스트로 검출하는 것보다 정적 분석을 통해 검출하는 것이 용이하다.

2. 낮은 비용

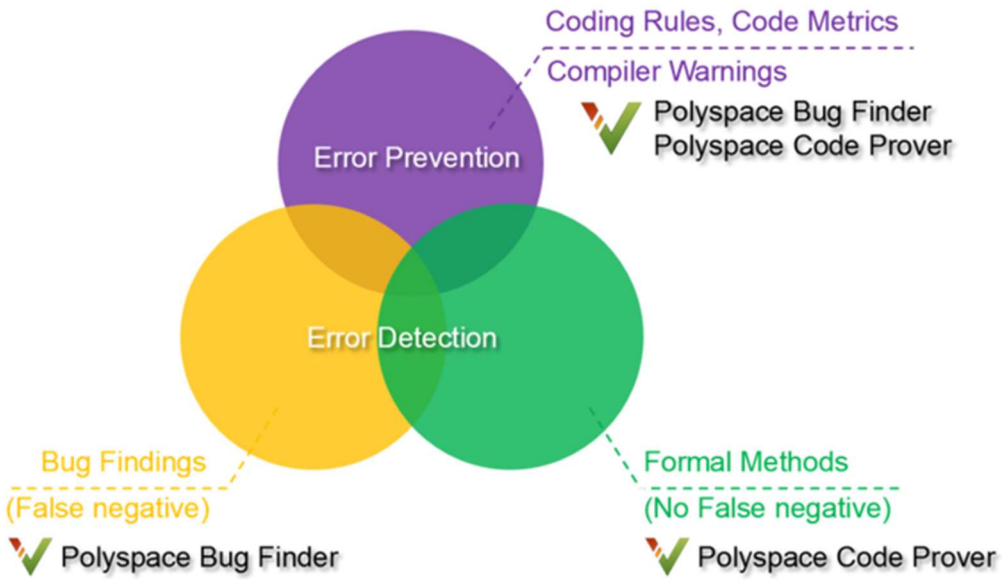
정적 분석에는 동적 분석에 필요한 테스트 케이스 설계, 테스트 실행, 테스트 결과 분석에 의한 비용이 발생하지 않기 때문에 상대적으로 저렴한 비용에 행할 수 있다.

2.2. Coding Rules

코딩 규칙은 특정 언어로 작성된 프로그램에 대한 프로그래밍 스타일, 사례, 방법 등으로서 이는 파일 구성, 들여쓰기, 주석, 선언, 명령문, 공백, 명명 규칙, 프로그래밍 관행, 아키텍처 모범 사례 등을 다루어 소스코드의 가독성을 높이고 유지관리를 용이하게 하여 소프트웨어의 품질을 높일 수 있다. 이러한 규칙은 공식적으로 문서화되거나 회사, 팀, 개인 단위로 이루어질 수도 있다. 그리고 고품질 코드를 생산하기 위해 디자인된 Rules의 모음을 코딩 표준이라고 부르며 우리는 이러한 코딩 표준 중 MISRA-C 와 SEI CERT, ISO 5055를 이용할 것이다.

3. 연구 내용

3.1. Polyspace



Polyspace 프로그램이 제공하는 기능[3]

Polyspace는 Error prevention, Error Detection 기능을 제공한다. Error prevention은 코딩 규칙 검사를 말한다. Error Detection은 에러를 탐지하는 능력을 말한다.

Polyspace가 분석하는 규칙은 크게 보라색 영역인 Error Prevention, 노란색 부분인 Bug Findings, 초록색 부분인 Formal Methods로 구분할 수 있다.

Error Prevention에서는 영역 에러를 미연에 방지하고자 하는 목적이 강하다. 이 분야에서 검사 가능한 항목들은 MISRA C와 같은 코딩 가이드라인에 대한 위반 사항을 찾아주거나 소프트웨어의 품질 지표로 삼을 수 있는 코드 메트릭이 있다. 위반 사항 혹은 목표 수치를 맞추지 못한 사항을 찾아 주고, 이를 수정함으로써 개발자가 코딩 규칙이나 코드 메트릭을 맞추기 위한 코딩 방식을 배우는데 매우 큰 도움을 줄 수 있다. 즉, 올바른 코딩 방식을 배움으로써 코딩 시 적용하여 에러를 미연에 방지할 수 있다. 그런데 사람들이 100개가 넘는 코딩 규칙 같은 것들을 외워서 위반 사항이 있는지 체크하기가 매우 어렵기 때문에 도구의 도움을 받아 위반 사항 혹은 목표 수치를 달성하지 못한 부분의 코드를 확인하는 것이다. 노란색 원 부분인 bug findings 부분은 코딩 규칙 위반 만으로는 찾아낼 수 없었던 버그들을 탐지할 수 있는 부분이다.

Bug Findings 부분은 소프트웨어 버그 발견을 의미한다. 위의 코딩 규칙 검사만으로 탐지할 수 없었던 버그들도 찾아낼 수 있다. Polyspace Bug Finder가 이 부분에 특화되어 있다.

Formal Methods 부분은 런타임 에러가 없음을 증명할 수 있는 것을 의미한다. 이 부분에서는 Polyspace Code Prover가 특화되어 있다.

다음 표로 Polyspace의 세부 도구인 Polyspace Bug Finder Polyspace Code Prover에 대해 자세히 비교해보았다.

Table 1. Bug Finder와 Code Prover 비교

	Polyspace Bug Finder	**Polyspace Code Prover**
목적	버그 및 오류 탐지	정적 코드 검증 및 증명
특징	프로그램의 버그와 오류를 찾고 수정하는 데 중점을 둔다. 코드를 정적으로 분석하여 잠재적인 문제를 식별하며, 배열 범위 초과, 널 포인터 참조, 정의되지 않은 변수 사용 등을 포함한 일반적인 프로그래밍 오류를 탐지한다.	프로그램의 정확성을 증명하고, 코드에 대한 formal methods를 수행한다. 코드의 모든 실행 경로를 검사하여 특정한 속성이 항상 참임을 증명하고, 버그와 오류 없음을 검증한다.

3.2. Coding Standard

3.2.1. MISRA C

MISRA C: 2012[4]는 MISRA에서 만든 C 언어용 개발 지침으로서 실수를 줄이거나 제거하기 위한 규칙을 정의한다. 본 과제에서는 2012년도 판인 Third Edition을 이용한다. Third Edition에서 개발 지침은 크게 Directives와 Rules로 분류되며 이는 코드만을 가지고 지침의 준수 여부를 확인할 수 있는지 여부로 나뉜다. Directives는 코드만으로 불가능하여 설계문서 등이 필요하고 Rules는 코드만으로 준수하였는지 확인할 수 있는 지침이다. 우리는 별도의 문서 없이 C 소스코드의 정적 분석 도구를 통한 분석 결과를 활용할 예정이므로 둘 중 Rules에 집중할 것이다.

또한 각 Rules는 어떠한 문제에 대한 지침인지에 따라 분류되어 목차에 따른 식별자를

가지며 이하와 같은 항목을 가진다. Table 2는 MISRA C 문서의 규칙 별 세부항목을 표현한다.

Table 2. MISRA 규칙의 세부항목

항목	세부항목	상세
Category	Mandatory	무조건 준수해야 함
	Required	준수해야 함. 하지 않는다면 'deviation'에 대한 공식적인 기술이 필요함.
	Advisory	준수할 것을 권장. 공식적인 'deviation'이 필요하지는 않으나 비준수에 대한 문서화를 위해 대체 조치 필요.
Decidability	Decidable	모든 경우에 예, 아니오 로 대답할 수 있음.
	Undecidable	모든 경우에 예, 아니오 로 대답할 수 없음. (ex 런타임 속성에 의존하는 경우)
Scope	System	모든 소스코드를 분석하여야 정확하게 확인 가능.
	Single Translation Unit	Translation Unit 단위로 개별적인 확인 가능.

항목의 Category는 지침을 준수해야 하는 정도를, Decidability는 정적 분석 도구가 코드가 지침을 준수하는지에 답할 수 있는지 여부를, Scope는 준수 여부를 파악하기 위해 확인해야 하는 범위를 뜻한다.

3.2.2. CERT C

SEI CERT C Coding Standard[5]는 C 언어의 보안 코딩을 위한 규칙을 제공한다. SERT C는 악용 가능한 취약점으로 이어지는 정의되지 않은 동작을 제거하는 등 안전하고 신뢰할 수 있는 시스템을 개발하는 것을 목표로 한다.

각각의 지침은 제목, 설명, 비 준수 코드 예제 및 준수 솔루션으로 이루어져 있다. 이 지침들은 규칙과 권고사항으로 나누어지는데 규칙은 세 가지 조건을 만족하여야 한다. 각각은 다음과 같다.

1. 위반 시 시스템의 안전성, 신뢰성, 보안성에 악영향을 줄 수 있는 결함이 발생할 수 있을 것

2. 소스 코드의 annotation이나 assumption에 의존하지 않을 것

3. 적합성을 자동 분석 도구나 수동 검사 등으로 결정할 수 있을 것

권고사항의 경우 충족시키면 시스템의 안정성, 신뢰성, 보안성을 향상시킬 수 있지만 위의 세 가지 중에서 만족하지 못하는 것이 있을 때 해당된다.

3.2.3. ISO 5055

ISO/IEC 5055:2021[6]은 소프트웨어 품질 평가 및 소스코드 평가를 위한 표준으로서 우리가 품질 평가에 이용할 ISO/IEC 25010:2011 Quality Model의 8가지 품질 속성 중 4가지인 Maintainability, Performance Efficiency, Reliability, Security에 대한 취약점 목록을 제공한다. 해당 취약점들은 CWE의 취약점울 사용하고 있다.

CWE란 Common Weakness Enumeration으로 보안에 영향을 줄 수 있는 일반적인 소프트웨어 및 하드웨어의 취약점 유형에 대한 목록이다. Polyspace에서 ISO 5055 표준의 준수 여부를 직접 검사할 수는 없지만 CWE의 준수 여부는 검사할 수 있었기 때문에 CWE의 준수 여부를 통해 간접적으로 확인할 수 있다고 판단하여 Coding Standards와 Quality Model의 품질속성의 매핑에 이용하고자 한다.

3.2.4. ISO 25010

ISO/IEC Quality Model[7]은 ISO에서 제정한 제품 품질에 대한 모델로서 본 과제에서 측정하고자 하는 품질 속성을 포함하는 8가지 품질 속성을 정의한 것이다. ISO Quality Model의 품질 속성은 다음과 같다.

Table 3 ISO 25010의 세부 사항

품질 속성	설명
Functional Suitability	제품이나 시스템이 특정 조건에서 사용될 때 명시적, 암시적 요구사항을 충족시키는 기능을 제공하는 정도.
Perfomance Efficiency	명시된 조건에서 사용된 자원의 양에 대한 성능.
Compatibility	동일한 하드웨어, 소프트웨어 환경을 공유하며 제품, 시스템 또는 구성요소가 다른 제품, 시스템 또는 구성요소와 정보를

	교환하거나 필요한 기능을 수행할 수 있는 정도
Usability	제품이나 시스템이 특정 사용자가 목적을 달성하기 위해 효과, 효율성, 만족을 얻으며 사용할 수 있는 정도
Reliability	지정된 기간 동안 지정된 조건에서 지정된 기능을 수행하는 정도
Security	정보 및 데이터를 보호하여 사람이나 다른 제품, 시스템이 권한에 적합한 데이터 액세스 수준을 갖도록 하는 정도.
Maintainability	관리자가 제품 또는 시스템을 수정할 수 있는 효과 및 효율성의 정도.
Portability	하나의 하드웨어, 소프트웨어 혹은 다른 사용 환경에서 다른 환경으로 이전될 수 있는 효과 및 효율성의 정도.

3.3. 품질 속성 매핑

본 과제에서는 MISRA C, SEI CERT C, ISO 5055의 3가지 코딩 표준을 ISO 25010의 품질 속성과 매핑하여 품질 점수를 측정한다. 품질 속성 매핑을 위해 각 코딩 표준의 코딩 규칙을 관련된 품질 속성과 매핑하는 방법을 정의한다.

3.3.1. MISRA C

MISRA C:2012 표준 문서에서 일부 가이드라인은 Portability Issue로서 Undefined, Unspecified, Implementation를 표기한다. Portability Issue는 ISO 9899:1990 및 ISO 9899:1999 표준의 내용을 기반으로 작성되었다. Table 3은 Portability Issue의 각 항목에 대한 설명이다.

Table 4 MISRA C에서의 Portability Issue

Portability Issue	설명	연관 품질속성
Unspecified	컴파일은 성공적으로 이루어지지만 컴파일러에 따라 동작이 달라질 수 있으며 한 컴파일러에서도 모든	Portability

	구성에서 일관적으로 동작한다고 할 수 없는 경우.	
Undefined	이는 컴파일러에 의해 오류가 표시되지 않을 수 있지만 함수에 대한 유효하지 않은 매개변수 등 본질적으로 프로그래밍 오류에 속한다.	Reliability
Implementation	Unspecified와 유사하게 컴파일러마다 동작이 다를 수 있다. 하지만 한 컴파일러 내에서는 일관된 접근을 취하고 문서화해야 한다.	Portability

일부 MISRA C Rule들은 Portability Issue가 정의되어 있지 않다. 이러한 경우 MISRA C 표준 문서의 vision에서는 높은 신뢰성이 요구되는 개발에 사용할 수 있다고 한다. 즉, 따로 표시되지 않았다면 MISRA C 표준의 가이드라인은 기본적으로 신뢰성(Reliability) 품질 속성에 연관이 있다고 볼 수 있다.

따라서 정리하면 Unspecified와 Implementation은 공통적으로 프로그래밍 오류는 아니지만 컴파일러에 따라 의도하지 않은 동작을 취할 수 있는 문제를 가지고 있다. 즉, 이식성(Portability) 품질속성과 연관이 있다. Undefined는 검출되지 않았을 수 있을 뿐 프로그래밍 오류에 해당한다. 의도하지 않은 동작을 하거나 아예 동작하지 않을 수 있으므로 신뢰성(Reliability) 품질 속성과 연관이 있다. 아무런 표시가 없는 경우 신뢰성(Reliability) 품질 속성과 연관이 있는 것으로 본다.

이러한 표시는 각 가이드라인에 대한 내용에서도 확인할 수 있지만 Appendix G와 Appendix H에서도 표시하고 있으므로 양 쪽 모두를 확인하여 각 가이드라인에 대해 연관되는 품질 속성을 체크할 수 있었다.

ID	Description	Reliability	Portability	Portability Issue	Severity
Rule 17.1	The features of <stdarg.h> sha	O	O	Undefined, Unspecified	Required
Rule 17.2	Functions shall not call themse	O		-	Required
Rule 17.3	A function shall not be declare	O		Undefined	Mandatory
Rule 17.4	All exit paths from a function w	O		Undefined	Mandatory
Rule 17.5	The function argument corresp	O		-	Advisory
Rule 17.6	The declaration of an array par	O		Undefined	Mandatory
Rule 17.7	The value returned by a functic	O		-	Required
Rule 17.8	A function parameter should n	O		-	Advisory
Rule 18.1	A pointer resulting from arithm	O	O	Undefined, Unspecified	Required
Rule 18.2	Subtraction between pointers s	O	O	Undefined, Unspecified	Required
Rule 18.3	The relational operators >, >=,	O	O	Undefined, Unspecified	Required
Rule 18.4	The +, -, += and -= operators	O		-	Advisory
Rule 18.5	Declarations should contain no	O		-	Advisory

3.3.2. CERT C

CERT C는 SEI(Software Engineering Institute)에서 개발한 The CERT C Secure Coding Standard 문서를 뜻한다. 또한 CERT C의 목표는 정의되지 않은 동작을 제거하여 안전한 보안 시스템을 개발하는 규칙을 제공하는 것이다. 따라서 별도의 표기가 없는 가이드라인은 기본적으로 Security에 매핑을 하였다.

Undefined Behavior, Unspecified Behavior라고 표시된 가이드라인은 MISRA에서 분류한 것과 같이 각각 Reliability, Portability로 매핑하였다.

그리고 ISO 5055, MISRA C와 관련 있는 경우를 모두 표시하여 관련 있는 규칙을 모두 매핑하였고, 완전히 동일한 규칙은 제거하여 중복을 방지했다.

Table 5. CERT C 매핑 테이블의 일부

ID	Reliability	Security	Portability	Related ISO 5055	Related MISRA	Exact ISO 5055	Portability Issue
MEM03-C		O		CWE-226 CWE-244			
MEM04-C	O	O	O	CWE-687			Undefined, Unspecified
MEM05-C	O	O			Rule 17.2		

위 표를 보면 MEM03-C는 CERT C이므로 기본적으로 Security 속성에 매핑이 되고 관련된 규칙인 CWE-226과 CWE-224는 Polyspace에서 검출할 수 없는 규칙이므로 Security 속성에만 매핑이 된다.

MEM04-C는 기본적으로 Security 속성에 더해 CWE-687 속성은 Polyspace에서 검출할 수 없어 관련이 없지만 Undefined, Unspecified로 Portability Issue가 있으므로 추가적으로 Reliability와 Portability에 속성이 매핑이 된다.

MEM05-C는 기본적으로 Security 속성에 더해 MISRA의 Rule 17.2 규칙과 관련이 있고, Rule17.2는 Reliability와 관련이 있으므로 추가적으로 매핑해준다.

같은 방식으로 모든 규칙에 대해 매핑을 완료하여 엑셀로 정리하였다.

3.3.3. ISO 5055

앞서 말했듯이 ISO 5055는 Portability를 제외한 4개의 품질속성에 관여하는 취약점에 대한규칙을 제공하고 있다. 해당 품질속성을 매핑하는 것에 더해서 각 규칙이 CWE를 이용하고 있는 특징상 CWE의 다른 표준의 규칙과의 관계를 제공하는 Taxonomy Mapping을 이용할 수 있다. CERT C의 규칙과의 관계 역시 제공되기 때문에 유사한 규칙이라면 같은 품질 속성에 영향을 줄 수 있다고 보고 추가로 매핑하는 것이다.

ID	Description	Maintainability	Reliability	Security	Related CERT C	Related CERT C(Exact)
CWE-407	Algorithmic Complexity	O				
CWE-478	Missing Default Case in Switch Statement	O				
CWE-480	Use of Incorrect Operator	O	O	O		
CWE-484	Omitted Break Statement in Switch	O	O			
CWE-561	Dead Code	O			MSC07-C	
CWE-570	Expression is Always False	O		O	MSC00-C	
CWE-571	Expression is Always True	O		O	MSC00-C	
CWE-783	Operator Precedence Logic Error	O		O		EXP00-C

3.3.4. 품질 속성 확장

각각의 표준에 대한 매핑이 완료되면 셋을 통합한 테이블을 만든다. 이 때 ISO 5055와 CERT C에서 표기한 Related guideline에 대한 정보를 활용할 수 있다. 서로 유사한 가이드라인이라면 영향을 주는 품질 속성 역시 공유하고 있다고 판단할 수 있는 것이다. 따라서 연관된 가이드라인에서 체크된 품질 속성을 추가로 체크할 수 있다. Figure1과 Figure2 는 품질 속성 확장 방법을 설명하기 위한 예제이다.

ID	Reliability	Security	Portability	Related MISRA C
ARR39-C	O	O	O	Rule 18.1 Rule 18.2 Rule 18.3 Rule 18.4

Figure 1

ID	Reliability	Portability	Portability Issue
Rule 18.1	O	O	Undefined, Unspecified
Rule 18.2	O	O	Undefined, Unspecified
Rule 18.3	O	O	Undefined, Unspecified
Rule 18.4	O		-

Figure 2

Figure 1은 CERT C의 규칙 중 하나인데 MISRA C 연관 규칙이 있는 것을 확인할 수 있

다. 그리고 Figure 2는 해당 MISRA C 규칙들로 매핑 규칙에 따라 Reliability과 Portability 품질 속성에 연관되어 있다. 따라서 본래 Security 품질 속성에만 매핑 되어 있었던 ARR39-C 규칙이 Reliability 및 Portability 품질 속성에도 매핑 되는 것이다.

단 이때 ISO 5055와 SEI CERT C 사이에는 Relate가 Exact인 일부 가이드라인이 존재한다. 이는 두 가이드라인이 완전히 일치한다는 뜻으로써 품질 속성의 통합 체크는 진행하되 CERT C 쪽의 가이드라인만 남기도록 한다. 같은 규칙의 위반이 양 쪽에서 검출되면 실제보다 위반 횟수가 높게 계산되어 점수에 왜곡이 생길 수 있기 때문이다.

3.4. 품질 점수 측정

전체 규칙 위반횟수를 5로 나누어 상위 10%는 규칙 위반 점수를 5로, 상위 35%는 4로, 상위 65%는 3으로, 상위 90%는 2로, 나머지는 1로 설정했다.

심각도는 각 표준 문서를 참고하였고, 심각도를 낮은 것부터 1,2,3순서로 매핑하였다.

Table 6. 각 표준에서의 심각도 표시 방법

	심각도 수준 1	**심각도 수준 2**	**심각도 수준 3**
MISRA C	Advisory	Required	Mandatory
CERT C	L3	L2	L1
ISO 5055	Low	Middle	high

MISRA C의 경우 MISRA C:2012 표준 문서에서 심각도를 Advisory, Required, Mandatory 순으로 분류해 그것을 참고하였다. CERT C의 경우 카네기 멜런 대학교에서 분류한 기준을 살펴보니 L3,L2,L1으로 분류하여서 심각도가 가장 낮은 L3를 심각도 수준 1로 설정하였다. 마찬가지로 ISO 5055도 표준 문서를 참고하여 심각도가 가장 낮은 low를 심각도 수준 1로 설정하였다.

구한 규칙 위반 점수에 각 Quality Model별로 구분된 규칙의 심각도를 곱한다. 그러면 각 Quality Model별 규칙의 심각도가 반영된 품질 점수를 구할 수 있다. 이를 QSscore라고 가정하면, 함수 별 점수를 구하는 식은 (1-(QSscore/TOTAL_SEVERITY))*100으로 함수 별 Quality Model로 구분된 규칙의 점수를 구할 수 있다.

파일 별 점수는 함수 별 점수를 모두 구한다음 합하고 그 결과를 파일에서 위반한 규칙의 수로 나눈다. 프로젝트의 점수는 파일 별 점수를 모두 구한다음 합하고 그 결과를 프로젝트에서 위반한 규칙의 수로 나눈다.

3.5. 품질 측정 도구 개발

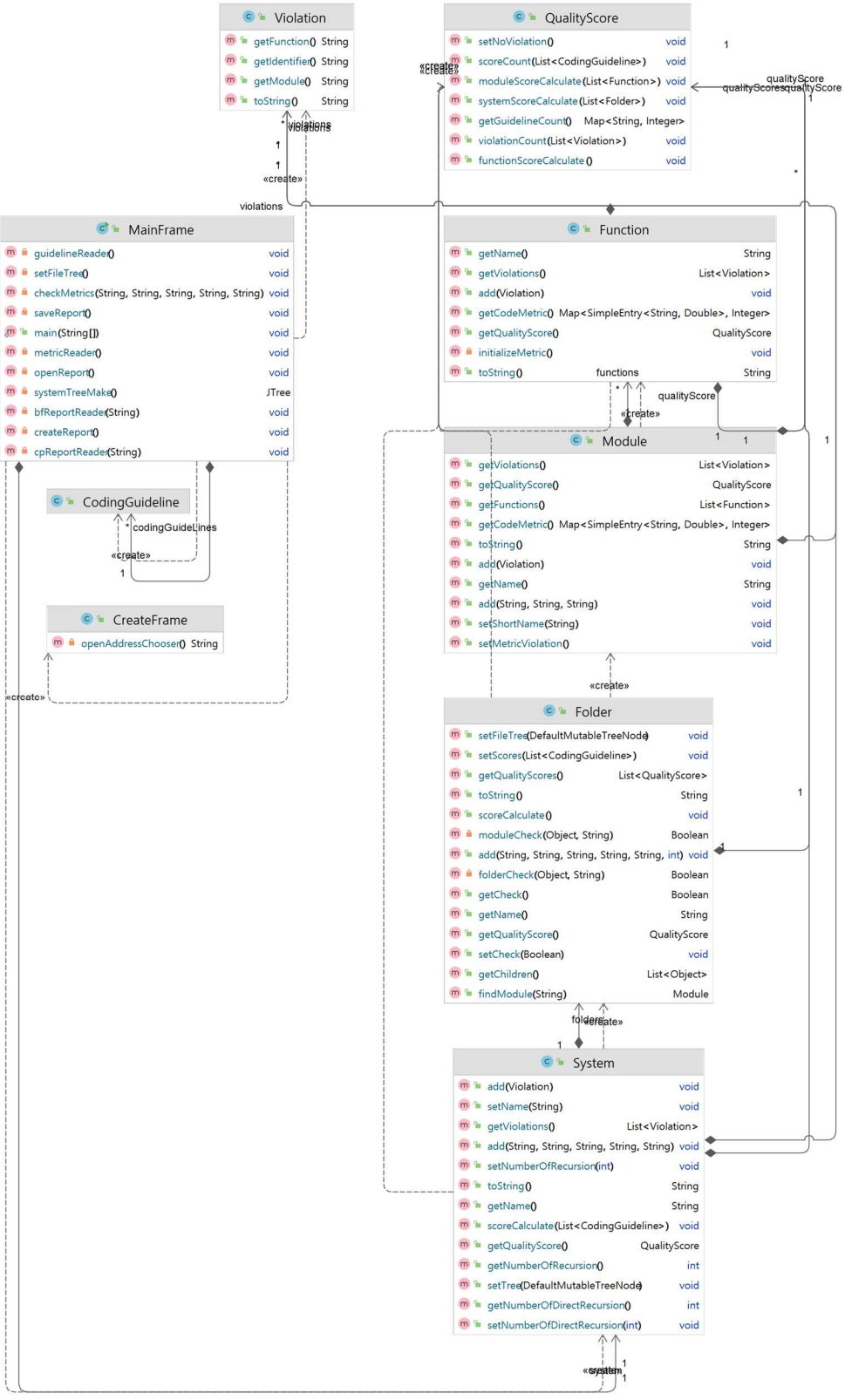

Violation
getFunction() String
getIdentifier() String
getModule() String
toString() String
QualityScore
setNoViolation() void
scoreCount(List<CodingGuideline>) void
moduleScoreCalculate(List<Function>) void
systemScoreCalculate(List<Folder>) void
getGuidelineCount() Map<String, Integer>
violationCount(List<Violation>) void
functionScoreCalculate() void
«create»
violations
qualityScore
MainFrame
guidelineReader() void
setFileTree() void
checkMetrics(String, String, String, String, String) void
saveReport() void
main(String[]) void
metricReader() void
openReport() void
systemTreeMake() JTree
bfReportReader(String) void
createReport() void
cpReportReader(String) void
Function
getName() String
getViolations() List<Violation>
add(Violation) void
getCodeMetric() Map<SimpleEntry<String, Double>, Integer>
getQualityScore() QualityScore
initializeMetric() void
toString() String
functions
Module
getViolations() List<Violation>
getQualityScore() QualityScore
getFunctions() List<Function>
getCodeMetric() Map<SimpleEntry<String, Double>, Integer>
toString() String
add(Violation) void
getName() String
add(String, String, String) void
setShortName(String) void
setMetricViolation() void
CodingGuideline
codingGuideLines
CreateFrame
openAddressChooser() String
Folder
setFileTree(DefaultMutableTreeNode) void
setScores(List<CodingGuideline>) void
getQualityScores() List<QualityScore>
toString() String
scoreCalculate() void
moduleCheck(Object, String) Boolean
add(String, String, String, String, String, int) void
folderCheck(Object, String) Boolean
getCheck() Boolean
getName() String
getQualityScore() QualityScore
setCheck(Boolean) void
getChildren() List<Object>
findModule(String) Module
System
add(Violation) void
setName(String) void
getViolations() List<Violation>
add(String, String, String, String, String) void
setNumberOfRecursion(int) void
toString() String
getName() String
scoreCalculate(List<CodingGuideline>) void
getQualityScore() QualityScore
getNumberOfRecursion() int
setTree(DefaultMutableTreeNode) void
getNumberOfDirectRecursion() int
setNumberOfDirectRecursion(int) void

Figure 3

Figure 3은 규칙 위반 정보 및 품질 매핑 정보를 입력 받고 품질 점수를 산출하는 일에 관여하는 클래스들의 다이어그램이다.

3.5.1. 규칙 위반 및 품질 매핑 정보 분석

실제 개발한 도구에서 입력에 관여하는 클래스의 다이어그램이다. MainFrame이 프로그램의 기본 GUI를 담당한다. 프로그램이 호출되어 MainFrame이 생성되면 guidelineReader, metricReader 메소드를 통해 사전에 우리가 작성한 매핑 데이터 및 Code Metric에 대한 파일을 읽어 CodingGuideline 클래스의 형태로 저장한다. 이후 MainFrame 창에서의 조작에 따라 CreateFrame 클래스가 호출되면 필요한 데이터 중 Polyspace의 분석 결과를 입력받기 위한 창을 생성한다. 해당 창을 통해 파일의 주소를 입력받고 나면 cpReportReader 및 bfReportReader 메소드를 통해 해당 파일을 읽어 들인다.

Table 7. Polyspace의 Result list에서 사용되는 부분의 일부

ID	Family	Group	Check	Function	File
32768	MISRA C:2012	Dir 4 Code design	- 생략 -	File Scope	₩sqlite-master₩test₩tt3_checkpoint.c
32806	MISRA C:2012	11 Pointer type conversions	- 생략 -	mergeWorkerFinishHierarchy()	₩sqlite-master₩ext₩lsm1₩lsm_sorted.c
2219	SEI CERT C	Integers (INT)	- 생략 -	uintCollFunc()	₩sqlite-master₩ext₩misc₩uint.c
35132	CWE	Numeric Errors	- 생략 -	mergeWorkerLoadHierarchy()	₩sqlite-master₩ext₩lsm1₩lsm_sorted.c

32771	Code Metric	Function Metrics	- 생략 -	treeShmkey()	₩sqlite-master₩ext₩lsm1₩lsm_tree.c

Table 7은 Polyspace의 Result list에서 사용되는 부분을 일부 가져온 것이다. Family의 MISRA, CERT, CWE는 어떤 Coding Guideline에 대한 위반인지를 나타낸다. Code Metric은 Metric 측정값임을 의미한다. Guideline에 대한 위반은 없을 수도 있으므로 먼저 Code Metric의 File에 표시된 경로를 통해 Folder, Module을 생성한다. 이후 Function이 File Scope라면 넘어가고 아니라면 Function을 생성한다. 이렇게 각 metric값이 기록된 Folder, Module, Function이 얻어지고 나면 Guideline에 대한 위반을 읽어 해당하는 경로에 Violation을 추가하는 식이다.

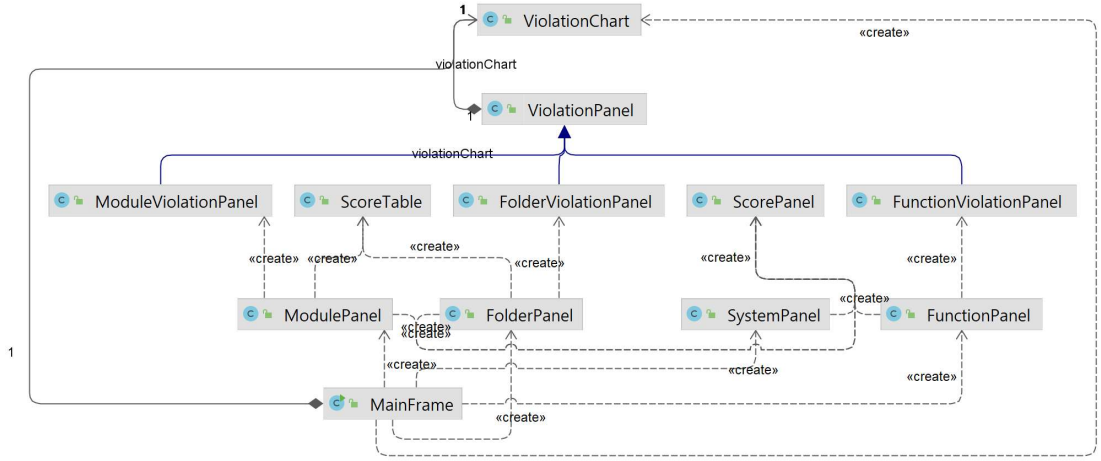
3.5.2. 품질 속성 측정

점수 산출 방법은 3.5에서 다룬 내용을 따른다. 3.6.1에서 필요한 파일을 읽는 과정이 끝나면 먼저 System의 scoreCalculate 메소드가 호출된다. 이 메소드는 다시 System의 하위 항목인 Folder에 대해 setScores 메소드를 호출한다. setScores 메소드는 Folder의 하위 항목이 Module이라면 Module의 QualityScore가 가지는 moduleScoreCalculate를 호출하고 Folder라면 다시 해당 Folder에서 setScores를 호출한다.

만약 moduleScoreCalculat가 호출되었다면 다시 Module의 하위 Function들의 QualityScore가 가지는 functionScoreCalculate 메소드를 호출하여 Function의 품질 속성을 측정하고 function의 점수 산출이 완료되었다면 이를 이용하여 Module의 점수 산출이 완료된다.

위와 같은 과정을 거쳐 Folder의 하위 항목의 점수가 모두 산출되었다면 이를 이용하여 Folder의 점수를 산출하고 다시 보다 상위 Folder의 점수를 산출하는 식으로 거슬러 올라가 최종적으로 System의 점수를 산출하게 된다.

3.5.3. 품질 측정 결과 시각화



출력에 관여하는 클래스의 다이어그램이다. Main Frame은 기본 GUI를 담당하는 클래스이므로 마찬가지로 사용된다. MainFrame의 좌측은 입력단계에서 저장한 구조를 트리로 만들어 표시한다. 우측은 트리의 항목을 선택하는 것에 대응해 System, Module, Folder, Function 각각의 Panel을 만들며 선택된 항목의 하위 항목 중 가장 Guideline 위반이 많은 항목 10개에 대한 차트를 ViolationChart를 통해 별도의 창으로 표시한다.

각각의 Panel은 선택된 대상의 품질 점수를 ScorePanel에서 표시하며 하위 항목들의 품질점수를 ScoreTable, 세부 위반사항에 대한 표를 ViolationPanel을 통해 표시한다.

4. 연구 결과 분석 및 평가

4.1. 품질 측정 대상 소프트웨어

품질 측정을 위해 대표적인 오픈소스 C 프로그램인 FFmpeg[8], OpenSSL[9], SQLite[10]를 사용하였다. FFmpeg는 오디오, 비디오, 자막 및 관련 메타데이터 등 멀티미디어 데이터를 처리하기 위한 라이브러리 및 도구 모음이다. OpenSSL은 SSL 프로토콜로 알려진 TLS 프로토콜을 위한 강력하고, 상용 등급의 모든 기능을 갖춘 오픈소스 툴킷이다. 프로토콜 구현은 독립적으로 실행할 수 있는 강력한 범용 암호화 라이브러리를 기반으로 한다. SQLite는 독립형 트랜잭션 SQL 데이터베이스 엔진을 구현하는 in-process 라이브러리

로서 별도의 서버 프로세스를 요구하지 않는 임베디드 SQL 데이터베이스 엔진이다. Table 4는 품질 측정 대상 소프트웨어의 규모 메트릭 값을 표현한다.

Table 8. 품질 측정 대상 소프트웨어의 규모 메트릭

대상 소프트웨어	메트릭	값
FFmpeg	LOC	1,590,554
	# of Files	4,131
	# of Classes	4,472
	# of Functions	29,278
OpenSSL	LOC	721,606
	# of Files	1,985
	# of Classes	1,588
	# of Functions	17,706
SQLite	LOC	394,515
	# of Files	391
	# of Classes	865
	# of Functions	7,438

4.2. 품질 측정 결과

다음은 예시 프로젝트 중 SQLite 코드의 일부분이다.

```
u64 lsmGetU64(u8 *aOut){
  return ((u64)aOut[0] << 56)
       + ((u64)aOut[1] << 48)
       + ((u64)aOut[2] << 40)
       + ((u64)aOut[3] << 32)
       + ((u64)aOut[4] << 24)
       + ((u32)aOut[5] << 16)
       + ((u32)aOut[6] << 8)
       + ((u32)aOut[7]);
}
```

이 함수에서 Polyspace로 검출된 규칙 위반 리스트는 다음과 같다.

ID	Family	Check	Information	Function
335	MISRA C:2012	10.7 If a composite expression is used..	Category: Required	IsmGetU64()
16168	MISRA C:2012	10.7 If a composite expression is used..	Category: Required	IsmGetU64()
20663	MISRA C:2012	8.13 A pointer should point to a const..	Category: Advisory	IsmGetU64()
15039	SEI CERT C	DCL13-C Declare function parameters..	Category: Recommendation	IsmGetU64()
20194	SEI CERT C	DCL00-C Const-qualify immutable objects..	Category: Recommendation	IsmGetU64()

총 5개의 규칙들이 검출되었으며 품질 측정도구로 측정한 결과는 다음과 같다.

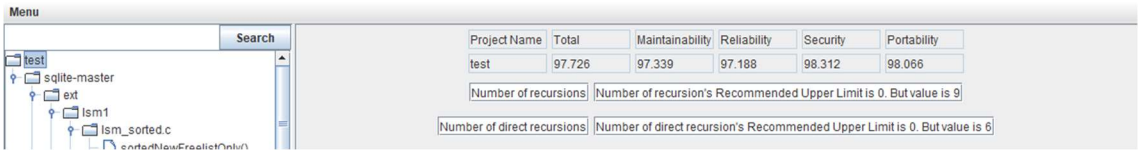
Function Name	Total	Maintainability	Reliability	Security	Portability
IsmGetU64()	99.126	100.000	97.500	99.002	100.000

All

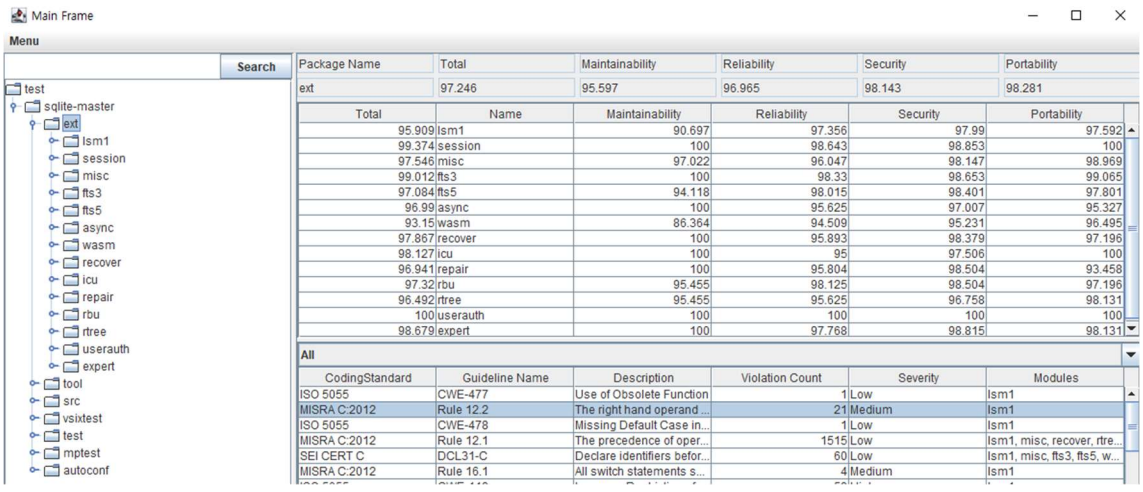
CodingStandard	Guideline Name	Description	Violation Count	Severity
MISRA C:2012	Rule 8.13	A pointer should point to a const-...	1	Low
SEI CERT C	DCL00-C	Const-qualify immutable objects	1	Low
MISRA C:2012	Rule 10.7	If a composite expression is use...	2	Medium
SEI CERT C	DCL13-C	Declare function parameters that ...	1	Low

4.3. 품질 측정 도구 실행 결과

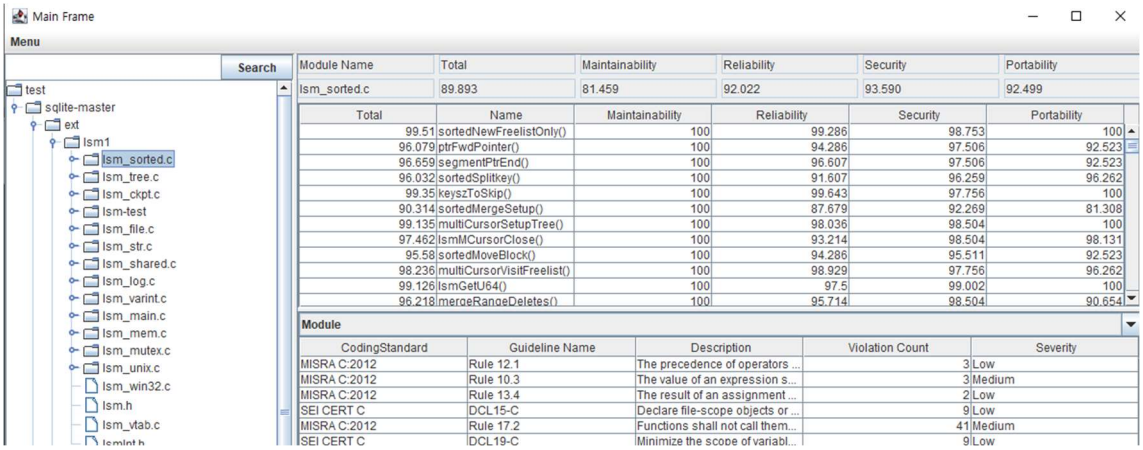
예시 프로젝트 중 하나인 SQLite를 통해 실행 결과를 확인하면 다음과 같다.



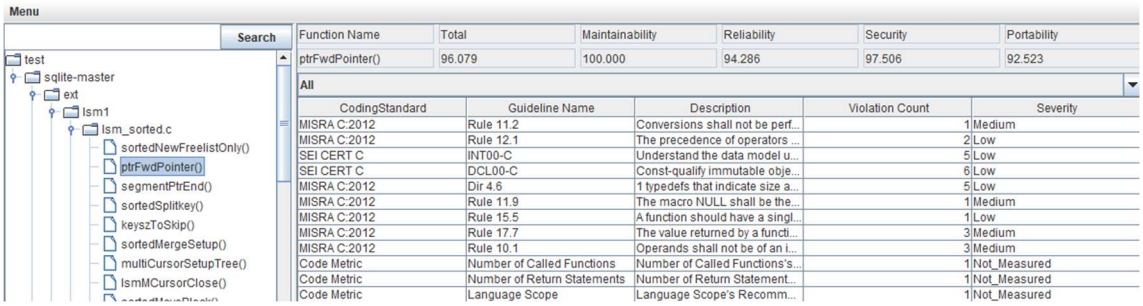
먼저 트리 최상단 노드에서 프로젝트 전체의 점수를 확인할 수 있으며 Code Metric에 위반사항이 있을 경우 확인할 수 있다.



이후 Package에 해당하는 하위 노드에서 해당 Package의 점수와, Package에 속하는 하위 대상의 점수들을 확인할 수 있다. 그리고 밑의 표에서 세부적인 Coding Guideline 위반과 횟수, 위반이 발생한 위치를 확인 할 수 있도록 되어있다.

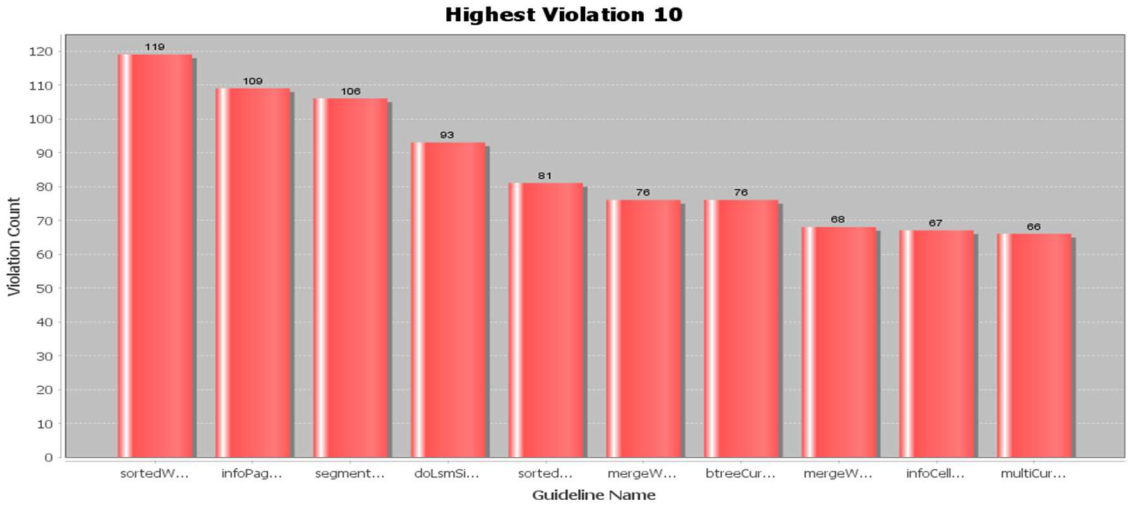


Module은 하위의 Function의 점수 이외에도 Module 자체의 File Scope인 Guideline 위반이 존재한다. 따라서 처음 선택 시에는 Module 자체의 위반을 표시하며 하단 표의 Module로 표시된 부분을 Function으로 변경하면 하위 Function의 세부적인 위반을 확인할 수 있으며 표시되는 차트 역시 변경된다.

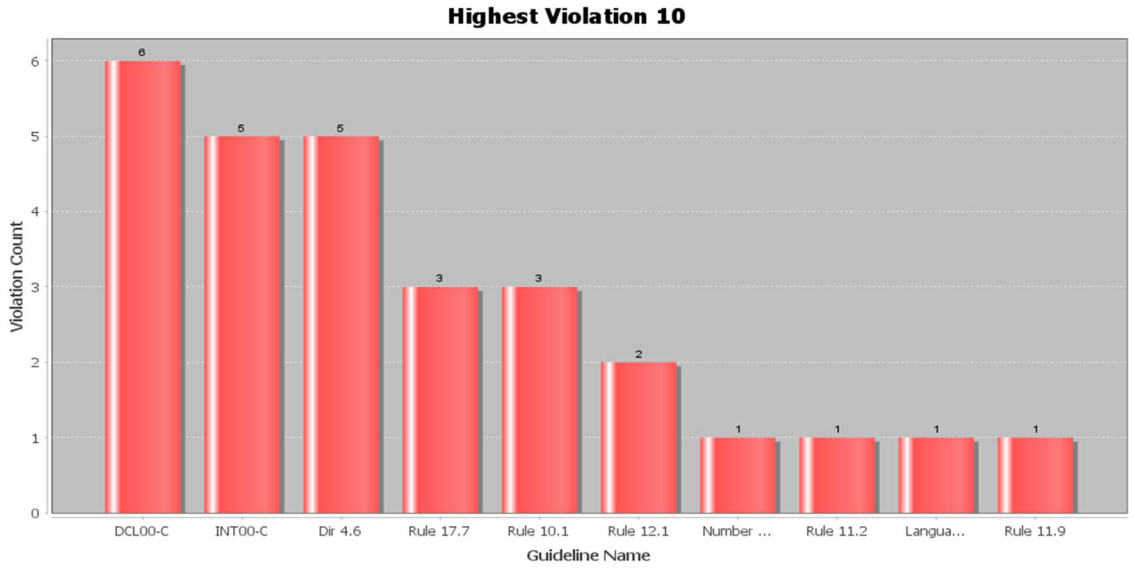


Function의 경우 하위에 속하는 것이 없으므로 Function의 점수와 Guideline위반에 대한 세부 정보 만을 표시한다.

Package나 Module의 하위 Function을 표시할 때에는 다음과 같이 하위 항목의 Guideline 위반 횟수를 확인할 수 있는 차트가 표시된다.



반면 Module 자체의 Guideline 위반이나 Function의 경우 다음과 같이 위반한 Guideline 에 대한 Chart가 표시된다.



5. 결론 및 향후 연구 방향

5.1. 결론

본 과제에서는 C언어 소스코드를 Polyspace 프로그램을 이용해 정적분석 한 결과와, MISRA, CERT C,ISO 5055를 이용해 ISO 25010 Quality Model과 매핑한 매핑 테이블 결과를 이용해 품질 점수를 성공적으로 측정하였다. C 언어로 구현된 품질 측정 대상 소프트웨어는 github의 것을 사용하였다. 점수 측정을 통해 전체 프로젝트의 어떤 모듈의 품질 점수가 부족하고 어떤 품질 점수가 부족한지 파악할 수 있었다. 그리고 파일의 어떤 함수에서 어떤 규칙을 위반하였는지 구체적으로 식별할 수 있었다. 어느 부분에서 어떤 규

칙을 위반하였는지 알 수 있으므로 소프트웨어의 품질 향상을 위한 지표로써 사용될 수 있을 것이라고 생각한다.

5.2. 향후 연구

품질 속성이 현재 사용한 5가지만 있는 것이 아니기 때문에 다른 품질 속성들도 추가하여 더 점수를 세분화 할 수 있으면 좋을 것 같다. 뿐만 아니라 다른 표준을 추가하여도 점수를 더 세분화 할 수 있을 것이다. 현재는 C언어 프로그램만 사용하였는데 다른 언어들을 활용해서도 점수를 측정할 수 있으면 코드 품질을 개선하는데 도움을 줄 수 있을 것이다.

6. 멘토 의견서

소프트웨어의 품질을 측정하는 방법이 왜 필요한지에 대한 추가적인 내용이 필요하다, 시각화 내용이 어떠한 시각화 내용인지에 대한 설명이 모호하다는 의견을 전달받았다.

소프트웨어의 품질을 정량적으로 측정함으로써, 관리자에게 개발 중인 소프트웨어의 품질 수준을 알려주고, 개발자에게 품질 수준이 낮은 모듈을 개선하도록 요구할 수 있다는 부분을 서론에 추가하였고, 그리고 시각화 관련해서는 3.5.3에 품질 측정 결과 시각화에 대한 내용을 챕터로 추가하고, 구성도와 설명을 추가하여 시각화와 관련된 부분의 내용을 보충하였다.

7. 참고 문헌

[1] 토요타 급발진 분석(2014.09) [Online]. Available: https://www.autoelectronics.co.kr

[2] SW 안전과 휴먼팩터의 중요성 - 보잉 737 MAX 사례(2019.07.16) [Online]. Available: https://spri.kr

[3] 정적 분석 도구를 활용한 소프트웨어 검증 [Online]. Available: https://m.blog.naver.com/matlablove/220918921366

[4] MISRA C:2012 - Guidelines for the use of the C language in critical systems, 2013

[5] SEI CERT C Coding Standard - Rules for Developing Safe, Reliable, and Secure Systems, 2016.

[6] Information technology — Software measurement — Software quality measurement — Automated source code quality measures, ISO/IEC 5055, 2021.

[7] ISO/IEC 25010:2011 - Systems and software engineering -Systems and software Quality Requirements and Evaluation (SQuaRE) -System and software quality models

[8] [Online]. Available: https://github.com/FFmpeg/FFmpeg

[9] [Online]. Available: https://github.com/openssl/openssl

[10] [Online]. Available: https://github.com/sqlite/sqlite