가전 기기 소프트웨어를 위한 C 코딩 스타일 검사기

임진수*, 이동주*, 조인행**, 우균*
*부산대학교 컴퓨터공학과
**LG 전자 HA 제어 연구소
e-mail : elphy@pusan.ac.kr

C Coding Style Checker for Home Appliance Software

Jin-Su Lim*, Dong-Ju Lee*, In-Haeng Cho**, Gyun Woo*
*Dept. of Computer Engineering, Pusan National University
**HA Control R&D Lab LG Electronics Inc.

요 약

표준 코딩 스타일은 개발 단계에서 오류 가능성이 있는 코드를 사전에 차단하고 코드의 가독성을 높여 소스코드의 품질을 높일 수 있는 대표적인 방법이다. 자동차 분야에서는 MISRA-C 와 같은 표준 코딩 스타일이 있으며 이를 검사하는 다수의 도구가 개발되었다. 본 논문에서는 소프트웨어의 안전성 및 신뢰성을 기반으로 국제 및 산업 표준 권고 사항과 경험적인 사례를 바탕으로 가전 기기 소프트웨어에 적합한 코딩 스타일 43 종을 정의했으며 이를 구현한 자동화 도구를 개발했다. 개발된 C 코딩 스타일 검사기를 이용하여 냉장고와 세탁기에 탑재된 소스코드에 대상으로 실험해 개발자들이 쉽게 위배하는 규칙에 대해서 살펴보았다. 위배한 코딩 스타일 중 주석관련 스타일이 각 소스코드 별로 64%, 24%로 가장 많이 차지하고 있으며, 제어문 관련 규칙이 12%, 17%, 코드 모양(Layout)관련 규칙이 4%, 11% 순으로 나타났다. 본 논문에서 개발한 코딩 스타일 검사 도구는 향후 양산되는 제품에 계속적으로 적용될 것이며 소프트웨어 품질 향상에 실질적인 도움이 될 것으로 기대된다.

1. 서론

표준 코딩 스타일은 오류를 예방하는 장점 이외에 다수의 개발자가 참여하는 소프트웨어 개발 환경에서 코드의 가독성을 높이며, 개별 코드의 통합 및 유지보수를 효과적으로 할 수 있도록 도와준다[1,5,15,18]. 하지만 이러한 장점에도 불구하고 실무에 잘 적용되지 않는 것이 산업계의 가장 큰 고민이다. 그 이유로는 개발자 간의 다른 코딩 습관, 개별 개발자의 이해의 정도와 실수에 의한 누락, 개발자의 불충분한 숙지와 같은 점이 있다. 또한 소스 코드의 사이즈가 큰 경우 코딩 스타일을 일일이 점검하는 것은 개발보다 더 많은 비용과 시간이 요구된다. 따라서 개발 중 이거나 혹은 개발 후의 소스 코드를 분석하고 코딩 스타일을 준수하는지 빠르게 확인할 수 있는 자동화 도구가 필요하다.

본 연구는 소스 코드를 자동으로 분석하여 표준 코딩 스타일 준수 여부를 확인하는 자동화 도구를 구현하고 이를 적용한 결과를 제시한다. 이를 위해 임베디드 소프트웨어에 일반적으로 적용되고 있는 MISRA-C[14]와 전자 제품의 내장형 소프트웨어 국제 안전기준 인 IEC61508 표준[8]을 참조했으며, 사고 사례를 바탕으로 경험적으로 얻게 된 코딩 스타일을 정의하고 자동화 도구에 구현했다.

2 장에서는 기존의 상용 코딩 스타일 도구의 특징에 대해 소개하고 3 장에서는 이슈가 되고 있는 코딩 스타일에 대해 살펴보며 4 장에서는 개발한 코딩 스타일 자동화 도구의 구조와 인터페이스에 대해서 설명한다. 5 장은 가진 기기에 탑재된 소스코드에 적용 결과에 대해 살펴보며, 마지막으로 결론을 맺는다.

2. 상용 코딩 스타일 도구

이 장에서는 잘 알려진 상용 코딩 스타일 도구들의 특징에 대해서 살펴본다. 다음 표 1 은 각 도구 별로 제조사, 주요 기능, 지원 표준 규칙, 사용자의 규칙 수정 또는 생성, 임베디드 타겟 컴파일러의 지원에 관하여 나타낸 것이다.

Programming Research 사의 QAC[7]는 MISRA-C 규칙 검사로 가장 널리 알려진 도구이다. 자동차 산업 분야에서 많이 사용되고 있으며, 자동차 분야에서 사용되는 타겟 컴파일러를 지원하고 있다. IPA/SEC[10]와 같은 일본 임베디드 협회에서 사용하는 표준 코딩 가이드 규칙도 지원하고 있다. 하지만 사용자가 규칙을 수정하거나 추가하는 것은 불가능하다.

Parasoft 사의 C++ Tester[11]는 Unit Testing 자동화 도구이다. Unit Testing 이외에 주요 기능으로 코딩 스타일 검사 기능이 있으며 MISRA-C 와 Effective C++[13]에 관한 코딩 스타일을 지원하고 있다. C++ Tester 는 GUI 수준의 Rule Wizard 기능이 있으며 사용자가 간단한 규칙을 수정하거나 생성할 수 있도록 제공하고 있다. 현재 C++ Tester 도 다양한 타겟 컴파일

<표 1> 상용 코딩 스타일 검사기

도구이름	주요기능	지원코딩규칙	코딩규칙추가 및 수정	타겟 컴파일러 확장 지원 여부
QA-C	Coding Style Checker	MISRA-C, IPA/SEC	N/A	커스터마이징
C++ Tester	Unit Testing	MISRA-C, EFFECTIVE C++	Rule Wizard	커스터마이징
Polyspace	Static Analysis	MISRA-C, JSF++	N/A	커스터마이징
SQMint	Coding Style Checker	MISRA-C	N/A	Renesas Compiler 만 지원
Code Inspector	Coding Style Checker	MISRA-C, IEC61508	Rule Template	파서 설정기능 제공

러를 지원하고 있지만, 지원되지 않는 컴파일러에 대해서는 커스터마이징을 해야 한다.

Mathworks 사의 Polyspace[4]는 요약 해석(Abstract Interpretation)기술을 기반으로 런타임 오류(runtime error)를 자동으로 검출하는 것이 주요 기능이다. 이외에 MISRA-C 와 JSF++[3]과 같은 표준 코딩 스타일을 검사를 제공한다. Renesas 사의 SQMint[16]는 Renesas 사에서 제공하는 타겟 컴파일러를 지원하며, MISRA-C 규칙 검사를 주요 기능으로 하고 있다.

SuresoftTech 사의 Code Inspector[20]는 MISRA-C 이외에도 IEC61508 과 같은 국제 표준에서 권고하는 사항을 코딩 스타일을 지원한다. Rule Template 기능을 제공해 미리 정의된 룰을 수정할 수 있으며, 파서 설정 기능을 통해 키워드 수준의 타겟 컴파일러 확장을 인식할 수 있다.

대부분의 상용 도구는 잘 알려진 표준 코딩 규칙을 검사하는 것이 주요기능이다. 일부 도구에서는 사용자가 코딩 스타일을 수정하거나 추가할 수 있도록 기능을 제공하고 있지만, 단순한 스타일에 대해서만 가능하기 때문에 규칙 추가에는 한계가 있다. 타겟 컴파일러의 지원 부분은 지원되지 않는 컴파일러에 대해서는 커스터마이징를 통해 지원되고 있는 실정이다.

3. C 코딩 스타일 이슈

기존의 C 코딩 스타일 규칙들은 코드의 가독성을 위한 들여쓰기, 이름 규칙, 간결한 표현, 주석 사용과 같은 스타일과, 코드의 이식성을 위한 함수 사용여부에 대한 제약이 주류를 이루었다[5,18]. 최근에는 소프트웨어의 품질을 높이는 측면에서 소스코드의 신뢰성 및 안정성, 유지보수성에 관련된 코딩 스타일이 이슈가 되고 있다. 대표적으로 잠재적 오류를 일으킬 만한 코드 형태를 금지하는 코딩 스타일, 문서 형태로 가공할 수 있는 주석의 사용, 코드 크기를 정량적으로 제한하는 코딩 스타일이 있다.

소스코드의 신뢰성 및 안전성을 위한 코딩 스타일은 잠재적 오류를 가지고 있는 코드 형태의 사용을 금지하고 있다. 대표적으로 IEC61508 Part7 의 C.2.9 항목에는 "subprograms should have a single entry and asingle exit only"이라고 권고하고 있다. 위의 권고 사항을 C 코딩 스타일로 정의하면, "return 문은 한번만 사용해야 한다."라고 정의할 수 있으며, 본 코딩 스타일 검사기에서는 Safety 항목의 R63 이라는 코딩 스타일로 정의하고 있다.

주석과 관련된 코딩 스타일은 주석의 형태 및 사용 방법 등을 제약한다. MISRA-C 의 MISRA_2_2 에서는 C 주석만 허용하고 있으며, MISRA_2_3 에서는 중첩된 주석의 사용을 금지하고 있다. 하지만 현재 대부분의 C/C++ 컴파일러에는 C++ 형태의 주석(외줄 주석)을 지원하고 있으며, 개발자의 편의성 입장에서 이를 허용하는 추세이다. 따라서 본 코딩 스타일 검사기에서는 C++ 형태의 주석은 허용하되 이중 주석을 금지(R41)하고 함수 내부에 기술되는 인라인 주석에 대해 들여쓰기를 맞추는 스타일(R42)를 정의하고 있다.

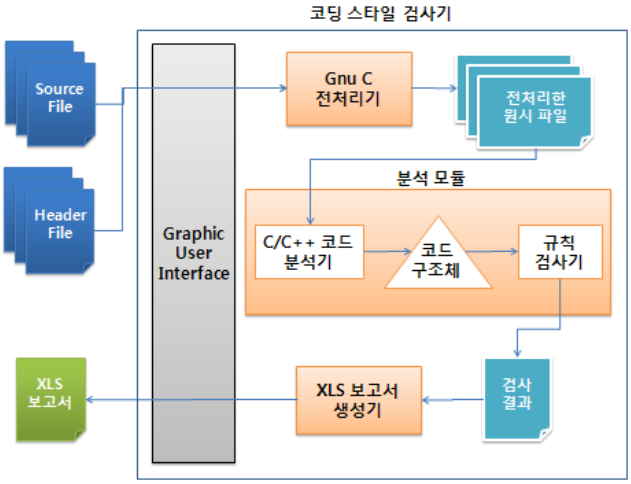
유지보수를 위한 코딩 스타일은 구현하는 함수의 최대 순환 복잡도(Cyclomatic Complexity)[12]와 라인수를 정의하며 이를 준수하도록 권고한다. IEC61508 의 Part7 의 C.2.7 에서는 순환 복잡도를 최대 10 으로 제한하고 있으며, C.2.9 에서는 함수의 LOC 를 100 으로 제한하고 있다. 본 코딩 스타일 검사기에서는 국제 표준의 수치보다는 다소 높은 범위를 허용한다. R71 에서는 순환 복잡도를 최대 50 으로 허용하며, R72 에서는 필수 복잡도(Essential Complexity)[6]를 최대 30 으로, R71 에서는 함수의 LOC 를 최대 200 까지 허용한다. R7 은 소프트웨어 설계와 직접 연관된 코딩 스타일이기 때문에 단계별로 수치를 줄여서 적용할 계획이며, 위 코딩 스타일에서 복잡도와 라인수의 허용 수치는 복잡도와 라인수가 고려되지 않은 소스코드에 대한 제약사항이다.

4. C 코딩 스타일 검사기 개발

코딩 스타일 검사기는 GUI 프로그램을 기반으로 전처리기(GNU C Preprocessor), C/C++ 코드 분석기(Code Analyzer)와 규칙 검사기(Rule Checker), XLS 보고서 생성기 모듈로 구성된다. 그림 1 은 코딩 스타일 검사 도구의 모듈 구성을 나타낸다.

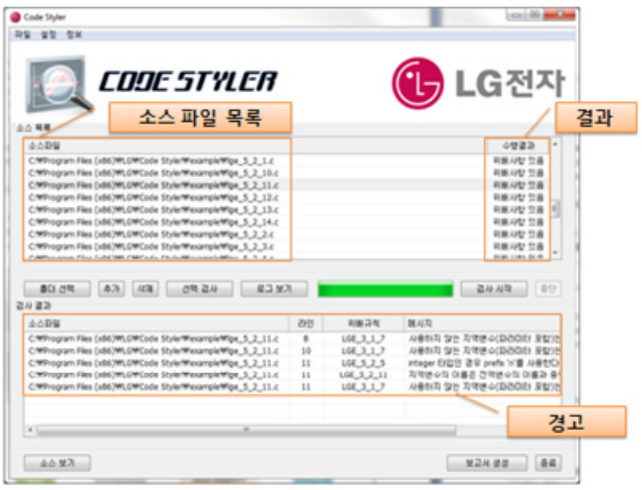
코딩 스타일 검사기는 GUI 를 통해 검사 대상 소스코드를 입력 받아 전처리 과정을 거치고 입력 소스코드를 분석하여 정의한 코딩 스타일을 준수하는지를 검사한다. 검사 결과는 GUI 또는 엑셀 보고서 생성기를 통해 결과를 제시하도록 설계되었다.

그림 2 는 코딩 스타일 검사기의 GUI 이다. GUI 는 검사할 소스파일 목록과 각 소스 파일의 수행 결과, 개별 소스 별로 위배한 스타일 목록 화면으로 구성된다. 도구 사용자는 GUI 를 통해 검사할 소스파일을 폴더 또는 파일단위로 선택할 수 있으며, 선택된 소스파일은 그림 2 와 같이 소스목록에 나타난다.



(그림 1) 코딩 스타일 검사기 기능 구조도

코딩 스타일 검사의 수행은 개별 소스 파일 단위로 진행된다. 첫 단계로 전처리기를 이용하여 전 처리를 수행한다. 이때 전처리 된 파일은 도구의 작업 폴더에 임시로 저장된다. 전처리 단계가 성공적으로 수행되면, 다음 단계는 C/C++ 코드 분석기를 이용하여 소스파일을 분석한다. 코드 분석기는 C/C++ 파서[2]를 포함하고 있으며 이를 이용하여 소스코드의 구문 정보를 추출한다. 추출한 구문 정보에서 특정 심볼이 함수인지, 변수인지 또 어떤 변수인지를 계산하여 심볼 및 타입 정보를 추출한다. 코드 분석기는 표준 C/C++ 구문 이외에 가전 기기에서 주로 사용하는 C 타겟 컴파일러의 확장형태의 키워드 또는 구문을 분석할 수 있도록 구현되었다.



(그림 2) 코딩 스타일 검사기 사용자 인터페이스

규칙 검사기는 코드 분석기가 생성한 코드의 구체적 정보를 이용하여 스타일을 위배하는 코드를 검출하며, 파일에 기록한다. 규칙 검사기는 약 43 개의 코딩 스타일을 검사한다. 검사 수행 후, 스타일 위배한 사항들은 GUI 를 통해 쉽게 확인할 수 있다. 그림 2 의 결과는 개별 소스파일 별로 수행 결과를 나타낸다.

컴파일 옵션 부재 또는 다른 이유로 전 처리기 과정이 실패한 경우 “전처리 실패”, 검출한 결 과가 전혀 없는 경우 “위배사항 있음”, 검출한 결과가 1 건 이상 있는 경우 “위배사항 없음”과 같이 표시된다. 사용자는 “위배사항 있음”으로 표시된 소스파일을 선택하면 GUI 를 통해 위배한 사항을 아래 검사 결과 목록에서 볼 수 있다.

전체 소스 파일에 대한 검사 결과는 엑셀 보고서 생성기를 이용하여 엑셀 파일로 생성할 수 있다.

5. 실험

개발한 코딩 스타일 검사 도구를 이용하여 냉장고와 세탁기 2 가지 종류의 가전 기기의 소스코드에 적용했다. 2 종의 소스 코드는 스타일 도구를 이용하지 않고 개발자가 코딩 스타일을 숙지한 상태에서 작성된 것이다. 다음 표 2 는 테스트 샘플에 대한 코드 크기이다. 파일의 개수에는 소스파일 및 헤더파일이 포함된 개수이며, 라인 수는 주석(Comment)이 포함된 라인 수이다.

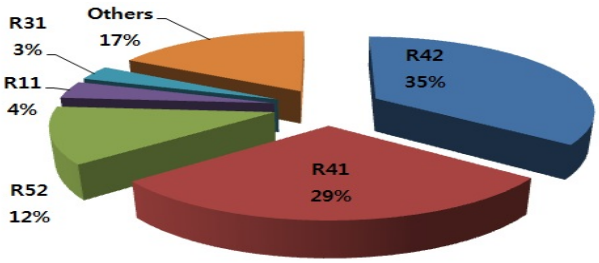
<표 2> 코딩 스타일 검사기 적용 소스코드

속성	TS1	TS2
가전기기	냉장고	세탁기
파일 수	109	100
LOC	43630	58760
타겟 컴파일러	Toshiba TLCS870X	Panasonic MN103S

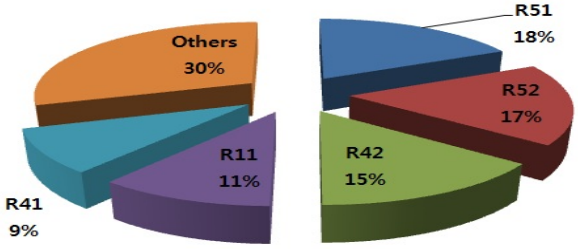
다음 그림 3 은 코딩 스타일 검사 도구를 이용하여 검출한 위배 사항 중 많이 검출된 5 개 스타일에 대한 비율이다. 실험 결과, 2 테스트 샘플에서 유사한 결과가 나타났다. 많이 검출된 5 개의 스타일 중, 4 가지가 두 테스트 샘플에서 검출되었다.

먼저 주석에 관련된 코딩규칙(R41,R42)이 많이 위배되었다. 실제 위배한 소스코드를 살펴보니, 외줄 주석이 있는 코드 전체에 묶음 주석(/* */)로 무심코 처리된 코드가 많이 보였다. 주석은 코드의 수행에 직접적인 영향을 주지 않기 때문에 다른 규칙에 비해 비교적 잘 지켜지지 않은 것으로 사료된다.

또한 if 문 사용시 else 문도 쓰도록 한 코딩 스타일(R52)과 연산자 사이의 빈칸을 사용하는 규칙(R11)도 공통적으로 많이 검출되었다. 이외에 TS1 에서는 지역변수 선언 시 타입을 기준으로 선언하는 규칙(R31)이 주로 검출되었으며, TS2 에서는 조건문 및 반복문 사용시 brace 를 사용하도록 하는 규칙(R51)이 검출되었다.



(a) TS1



(b) TS2

(그림 3) 많이 위배된 5 가지 코딩 스타일

실험 결과에서 나머지 위배 규칙들(Others) 중 복잡 복잡도 관련 스타일을 위배하는 함수도 일부 발견되었다. 다음 표 3 은 각 테스트 샘플 별로 복잡도 관련 스타일을 위배하는 수이다. R71 은 순환 복잡도가 50 을 넘지 않도록 함수를 설계하도록 권고하는 코딩 스타일이다. TS1 에서는 6 건 TS2 에서는 1 건이 발견되었다. 주로 입력에 따른 상태 변화를 제어하는 함수에서 발견되었다. R72 는 필수 복잡도가 30 을 넘지 않도록 함수를 설계하도록 권고하는 코딩 스타일이다. TS1 에서는 2 건 TS2 에서도 2 건이 발견되었다. R72 스타일을 위배한 함수 또한 R71 과 유사하게 입력 센서의 상태를 처리하는 함수에서 발견되었다.

<표 3> 복잡도 관련 규칙 위배 결과

규칙 번호	복잡도	TS1	TS2
R71	Cyclomatic	6	1
R72	Essential	2	2
R73	LOC	22	15

6. 결론

본 논문에서는 가전 기기에 적합한 표준 코딩 스타일을 선정하고 이를 검사하는 검사기를 개발했다. 2 종의 가전기기 소스코드에 적용하여 개발자들이 숙지하고 있지만 잘 지켜지지 않는 규칙들에 대해 살펴보았다. 먼저 이중 주석에 관련된 규칙을 많이 위배하였는데, 이는 외줄 주석을 포함하는 전체 코드에 대해 묶음 주석으로 처리하는 개발자의 코딩 습성으로 판단되며, 조건 문에 참인 경우와 그렇지 않은 경우에 대해 동등하게 구현해야 하는데, 조건에 참인 경우에 대해서만 구현된 코드가 많이 발견되었다. 이는 특정 알고리즘을 구현할 때, 조건에 부합하는 경우에 대해서만 고려하거나 예외를 고려하지 않는 개발자의 습성이 반영된 것으로 사료된다.

본 코딩 스타일 검사 도구는 다수의 개발자들이 빠른 시간 내에 코딩 여부를 준수하는지 확인할 수 있도록 사용자 인터페이스를 제공해 개발자들이 표준 코딩 스타일을 준수하는데 도움을 준다. 뿐만 아니라 개발이 완료된 소스코드를 검증하는 평가자도 이 도구를 이용하여 표준 코딩 스타일 준수 여부를 확인할 수 있다. 소프트웨어 개발 단계 및 평가 단계에서 자동화 도구를 도입함으로써 코딩 스타일에 적합한 소스코드를 개발할 수 있게 되고 이를 통해 보다 높은 안전성을 갖는 가전 기기 소프트웨어가 될 것으로 기대된다.

참고문헌

[1] A. N. Aeronautics and S. E. Branch. C Style Guide,1994.
[2] S. Chiba. *A Study of Compile-time Metaobject Protocol*. PhD thesis, The University of Tokyo, 1996.
[3] L. M. Corporation. JSF++: Air Vehicle C++ Coding Standard for the System Development and demonstration Program. http://www.research.att.com/～bs/JSF-AV-rules.pdf,2005.
[4] A. Deutsch. Static verification of dynamic properties. *PolySpace White Paper*, 2004.
[5] C. Elliott, M. Brader, L. Cannon, R. Elliott, L. Kirchhoff, J. Miller, J. Milner, R. Mitze, E. Schan, N. Whittington, et al. Recommended C Style and Coding Standards, 2000.
[6] J. Frederick P. Brooks. No silver bullet essence and accidents of software engineering. IEEE *Computer*, 20(4):10–19, 1987.
[7] P. R. Group. QA-C Source Code Analyzer. http://www.programmingresearch.com/qac main.html.
[8] International Electrotechnical Commission (IEC), Geneva Switzerland. IEC61508: *Functional Safety of Electrical/Electonic/Programmable Electonic Safety-Related Systems*, 1998.
[9] International Electrotechnical Commission (IEC), Geneva Switzerland. IEC62279: *Railway Applications Communications, Signalling and Processing Systems Software for Railway Control and Protection Systems,* 2002.
[10] IPA. IPA/SEC. http://sec.ipa.go.jp.
[11] A. Kolawa. Code Review Best Practices. Parasoft *White Paper*, 2008.
[12] T. J. McCabe. A complexity measure. IEEE *Transactions on Software Engineering*, SE-2(4):308–320, 1976.
[13] S. Meyers. Effective C++ (2nd ed.): *50 specific ways to improve your programs and designs*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1997.
[14] MISRA. *Guidelines for the use of the C language in critical systems*. MISRA Limited, UK, 2004.
[15] A. Reddy. Javatm Coding Style Guide, 2000.
[16] Renesas. SQMint. http://www.renesas.eu.
[17] A. Sandberg. C++ Coding Style, 2003.
[18] R. Stallman. GNU Coding Standards. http://www.gnu.org/prep/standards.
[19] R. Stallman and Z. Weinberg. The C preprocessor. GNU Project, *Free software foundation*, 1992.
[20] SuresoftTech. CodeScroll Code Inspector. http://www.suresofttech.com/eng.