Pylint를 이용한 Python 코딩 규칙 검사 시스템

김영훈[1], 우균[2]
[1,2]부산대학교 정보컴퓨터공학부
{yonghun83, woogyun}@pusan.ac.kr

A Coding Rule Checking System for Python Using Pylint

Yeonghun Kim[1], Gyun Woo[2]
[1,2]Dept. of Information Convergence Engineering, Pusan National University

요 약

코딩 규칙 준수는 대규모 프로젝트에서 프로그램의 버그를 줄이기 위해, 또 효과적인 유지보수를 위해 필수적이나 코딩 규칙을 학습하기 위한 초보자용 도구는 거의 없는 실정이다. 본 논문에서는 Python 프로그래밍 수업에서 코딩 규칙을 학습할 수 있도록 도와주는 시스템을 제안한다. 제안된 시스템은 학습자를 위해 별도의 설치 없이 Python 코딩 규칙 검사 결과를 영어와 한글을 병행하여 출력하는 규칙 검사 뷰어를 통해 학습자의 편의성을 제공한다. 또한, 품질 점수를 계산하여 학습자의 코딩 규칙 학습의 동기를 부여한다. 제안 시스템의 성능을 평가하기 위해 SonarQube와 검출 기능을 비교하였다. 2023년도 1학기 Python 프로그래밍 수업의 제출 코드를 검사한 결과, 제안 시스템이 SonarQube보다 247% 더 많은 종류의 규칙을, 또 235% 더 많은 개수의 규칙을 검사하는 것으로 나타났다. 이러한 비교 연구 결과를 고려할 때, 제안 시스템은 학습자에게 더 나은 코딩 규칙 학습 기회를 제공할 수 있을 것으로 기대된다.

1. 서론

코드 스멜(code smell)을 회피하는 법, 코드의 가독성을 높이고 유지보수에 효율적인 코딩 스타일(coding style) 등 코드를 작성하면서 지켜야 할 규칙을 코딩 규칙이라 한다. 코딩 규칙을 준수해야 개발자들이 코드를 더 효과적으로 이해하고 검토할 수 있으며 협업에도 도움이 된다. 소프트웨어 구조가 복잡해짐에 따라 협업 능력 향상에 관심이 높아지고 있으며, IT 기업에서는 협업을 잘하는 개발자를 우선 채용하고 있다.

코딩 규칙은 이론적인 학습만으로는 충분하지 않다. 코딩 규칙을 자연스럽게 적용하기 위해 평소 코드를 실제 규칙에 맞게 작성하는 것이 필요하다. 코드를 작성하는 습관은 한번 굳어지면 고치기 힘들어 프로그래밍 언어를 배울 때부터 적절한 습관을 형성하는 것이 중요하다[1].

하지만 프로그래밍을 처음 배우는 시기에 코딩 규칙까지 학습하기는 어렵다. 아직 학습하지 못한 이론이 코드 스멜의 해결 방안으로 제시되었을 때 코드를 수정하는 방법을 이해하기 힘들다. 그리고 프로그래밍 언어의 소스 코드에서 오류, 경고, 스타일 가이드 위반 등을 검출하는 규칙 검사 도구를 사용하더라도, 검사 결과 메시지가 영어로 출력되면 무엇이 문제인지 이해하기 힘들 수 있다. 또, 환경이 바뀔 때마다 규칙 검사 도구를 설치하고 이전과 같은 설정을 유지하기도 어렵다. 특히 SonarCube와 같은 대형 시스템의 경우에 더욱 그러하다.

본 논문은 별도의 설치 없이 처음 프로그래밍을 배우는 학습자를 위해 Pylint를 이용하여 코딩 규칙을 검사하는 시스템을 제안한다. 제안 시스템은 코드 스멜 회피와 코딩 스타일 준수 방법을 제시하는데, 영어 메시지와 더불어 한국어 메시지도 제공함으로써, Python 초보자도 쉽게 코딩 규칙을 학습할 수 있도록 한다.

또한, 제안 시스템의 성능을 검증하기 위해 SonarQube와 비교 실험을 수행한다. 제안 시스템에서 채택한 Pylint의 검출 기능이 타 시스템에 비해 낮지 않을까 우려할 수 있다. 이에 널리 사용되는 정적 분석 도구인 SonarQube와 비교하였다. 초보자의 코드를 대상으로 검사 규칙 개수와 검출 개수를 비교함으로써 Pylint 채택의 타당성을 검증한다.

본 논문의 구성은 다음과 같다. 2장에서는 Python의 코딩 스타일과 코딩 규칙 검사 도구를 살펴보고 규칙 검사 도구를 탑재할 온라인 평가 시스템으로 neoESPA를 소개한다. 3장에서는 Pylint를 이용한 코딩 규칙 검사 시스템의 설계 및 구현에 관하여 설명한다. 4장에서는 제안 시스템과 SonarQube의 검출 기능을 비교하고 5장에서 결론짓는다.

2. 관련 연구

2.1 Python 코딩 스타일

Python은 표준 코딩 스타일을 PEP 8(Python Enhancement Proposal 8)로 소개하고 있다[2]. 문서의 내용을 일부 소개하면, 공백 4개를 사용하여 들여 쓰는 것을 권장하는 들여쓰기 규칙과 변수와 함수 이름을 단어들을 밑줄로 구분하여 작성하는 뱀체(snake case), 클래스 이름을 두 번째 이후의 단어의 첫 글자는 대문자로 작성하는 낙타체(camel case)로 짓는 것을 권장하는 명명 규칙이 있다. 그 외에 닫는 괄호 규칙, 문서화 문자열(docstring) 등 다양한 규칙을 포함한다.

PEP 8 이외 구글에서 만든 코딩 스타일인 Google Python Style Guide가 있다. Google 스타일은 들여쓰기에 대한 규칙으로 공백 2칸을 권장하고, 피해야 할 변수명에 대해서 PEP 8보다 더욱 자세한 규정을 제시한다. 본 논문에서는 Python의 표준 코딩 스타일인 PEP 8을 사용한다.

2.2 Python 코딩 규칙 검사 도구

Pylint는 Python 코드에서 코드 스멜 검출 및 코딩 스타일 준수 여부를 검사하는 정적 분석 도구이다[3]. Pylint는 코딩 규칙 검사 이외에 코드의 복잡성, 코드 중복 등을 분석한다. 또한, 규칙마다 설명이 있는 문서를 제공해 학습자가 규칙에 대한 정보를 쉽게 찾을 수 있다.

SonarQube는 공개 소스 정적 코드 분석 도구 중 가장 널리 사용되는 도구이며 Python 이외에도 Java, C, C++, Javascript 등 여러 언어의 정적 분석을 지원한다[4]. 또한, 분석 결과를 시각적으로 보기 쉬운 형태로 제공하여 사용자가 필요한 정보를 쉽게 파악할 수 있다.

Pylint 이외에 유명한 규칙 검사 도구로 Pyflakes, pycodestyle, autopep8, Flake8이 있다. Pyflakes는 코드 스타일보다 오류가 발생할 수 있는 코드에 대한 검사를 지향한다. pycodestyle은 Python 표준 코딩 스타일 PEP 8에 적합한지 검사하는 도구이다. autopep8은 pycodestyle을 이용하여 코딩 스타일을 확인하고 옳은 형식으로 수정해 주는 규칙 검사 도구이다. Flake8은 Pyflakes, pycodestyle과 코드 복잡도 검사 도구 McCabe를 하나로 합친 것이다. 본 논문에서는 한 번의 검사로 학생들이 Python 수업에서 제출한 코드의 코드 스멜과 코딩 스타일을 모두 검사할 수 있는 Pylint를 채택하였다.

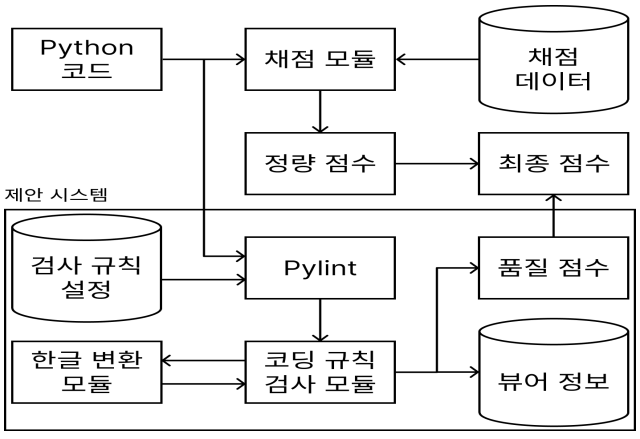
2.3 neoESPA

neoESPA는 부산대학교에서 2015년에 개발한 온라인 평가 시스템이다[5]. 교사가 과제를 등록하고 학습자가 과제 코드를 제출하면 채점 데이터를 이용하여 정량 점수를 출력하는 방식으로 학습자의 코드를 평가한다. 평가할 수 있는 프로그래밍 언어로는 C, C++, Java, Python, Haskell을 지원한다. 추후 여러 기능 도입을 고려하여 시스템을 설계 및 구현하였기 때문에 새로운 기능을 포함한 모듈을 개발하여 적용하기 좋은 시스템이다.

3. 시스템 설계 및 구현

3.1 전체 시스템 구성

이 절에서는 논문에서 제안하는 코딩 규칙 검사 시스템을 소개한다. 논문에서 제안하는 시스템 구조는 그림 1과 같다. neoESPA의 시스템에 제안 시스템을 추가로 개발하여 적용했다.



(그림 1) Pylint를 이용한 코딩 규칙 검사 시스템

학습자의 제출 코드는 neoESPA의 채점 모듈에서 채점 데이터를 이용하여 정량 점수를 출력한다. 또한, Pylint를 이용하여 코딩 규칙 준수를 검사한다. 검사 결과는 코딩 규칙 검사 모듈로 전달된다. 코딩 규칙 검사 모듈은 한글 변환 모듈을 이용하여 영어

로 출력되는 메시지를 한글로 변환하고 기존 검사 결과에 한글 메시지 추가 저장한 뒤 검사 결과의 부류, 규칙 이름, 메시지, 라인 정보, 칼럼 정보를 수집한다. 수집된 값을 바탕으로 뷰어를 위한 정보를 저장하고, 부류별 점수를 이용하여 품질 점수를 출력한다. 채점 모듈과 코딩 규칙 검사 모듈에서 출력된 정량 점수와 품질 점수를 합산하여 최종 점수를 학습자에게 보여준다.

3.2 검사 메시지 한글 변환

제안 시스템은 한글을 모국어로 사용하는 초보자들을 위해 검사 메시지를 영어 메시지와 한글 메시지로 변환하기 위해 한글 변환 모듈을 개발하였다. 규칙별로 영문 및 한글 메시지를 정리하였는데, 출력될 변수가 포함된 메시지와 한글 번역문을 JSON 파일로 정리하였다. 한글 문장은 영어 문장과 어순이 다르므로, 모든 한글 메시지를 확인하여 자연스러운 문장이 되도록 검토하였다.

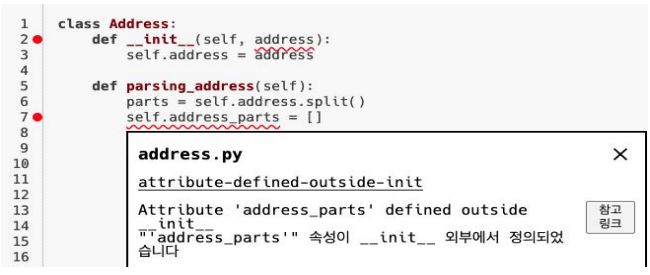
그림 2는 한글 메시지를 출력하기 위한 JSON 파일 일부를 나타낸다. Pylint 출력 메시지에서 변수 이름을 수집하기 위해 영문 메시지(eng 키의 값)에 정규 표현식을 삽입하였다. 수집된 이름은 한글 메시지(kor 키의 값)의 서식 문자열에서 사용된다.

```
"invalid-name": {
    "eng": "(.+) name (.+) doesn't conform to (.+)",
    "kor": "%s 이름 %s이(가) %s에 부합하지 않습니다."
},
"missing-function-docstring": {
    "eng": "Missing function or method docstring",
    "kor": "함수 또는 메소드 문서화 문자열이 누락되었습니다"
},
```

(그림 2) 한글 변환을 위한 JSON 파일 일부

3.3 규칙 검사 뷰어

학습자가 품질 점수를 확인하고 감점이 있을 때 어떤 규칙을 지키지 못하였는지 한눈에 확인하기 위해 규칙 검사 뷰어를 구현하였다. 그림 3은 학습자가 틀린 규칙을 볼 수 있는 규칙 검사 뷰어다.



(그림 3) 틀린 규칙을 확인할 수 있는 규칙 검사 뷰어

문제가 있는 라인의 좌측 행 번호에 빨간 동그라미, 해당 칼럼에 빨간 밑줄을 표시하였는데, 클릭하면 나타나는 메시지를 통해 규칙을 이해하고 코드를 수정할 수 있다. 또한, 참고 링크를 클릭하면 Pylint의 해당 규칙 페이지로 이동할 수 있다.

3.4 코딩 규칙

Pylint에는 420개 코딩 규칙이 있지만, 모든 규칙을 초보자에게 적용하는 것은 적합하지 않다. 예를 들어, Pylint의 규칙 중에 집합 조건제시법(set comprehension) 사용을 권장하는 “consider-using-set-comprehension”이 있다. 이 규칙은 조건제시법을 모르는 초보자에게 적용하기 힘들다.

학습자 수준에 적합한 규칙을 선별하기 위해 2023년도 부산대학교 컴퓨터공학과 1학년 1학기 Python 프로그래밍 수업에서 학생들이 제출한 코드를 활용하였다. 총 4,980개 코드를 Pylint로 검사한 결과, 120개의 검사 규칙이 도출되었다. 이 중 수업 진도나 과제의 특성을 고려하여 96개 규칙을 선정하여 반영하였다. 표 1은 선별된 96개 규칙 중 학생들이 많이 틀린 규칙을 부류별로 3개씩 정리한 것이다.

<표 1> 선별된 Pylint 규칙 중 학생들이 많이 틀린 부류별 규칙

부류	규칙 이름	개수
Error	undefined-variable	143
	used-before-assignment	112
	syntax-error	79
Warning	redefined-outer-name	2,914
	bad-indentation	2,264
	redefined-builtin	474
Convention	invalid-name	5,111
	trailing-whitespace	4,121
	missing-final-newline	1,960
Refactor	too-few-public-methods	640
	no-else-return	339
	useless-object-inheritance	251

3.5 품질 점수 부여

학습자의 코딩 규칙을 학습하기 위한 동기 부여 방안으로 규칙 검사 결과를 품질 점수로 산출하여 제시하였다. 표 2는 Pylint에서 코딩 규칙을 5개의 부류로 구분한 것을 neoESPA에서 4개의 부류로 재정의한 것이다.

Pylint의 Error는 실행 오류 부류이며 Warning은 실행 오류와 코드 스멜이 같이 있으므로, neoESPA에서는 두 부류를 Issue 부류로 재정의하였다.

<표 2> neoESPA의 부류와 Pylint의 부류 대응

neoESPA 부류	Pylint 부류
Fatal	Fatal
Issue	Error
	Warning
Style	Convention
Performance	Refactor

식 1은 품질 점수 산출식이다.

$$Q = \begin{cases} 0, & S < 0 \\ S \times W \times P^n, & S \geq 0 \end{cases} \quad (1)$$

식 1에서 S는 채점 점수, W는 채점 점수 반영 비율, P는 감점 반영 비율, n은 검출된 오류 개수이다. 부류별 가중치에 해당 부류별 검출 횟수 곱을 n으로 하였다. 예를 들어, 부류별 가중치가 1, 채점 점수가 100점, 채점 점수 반영 비율이 50%, 감점 반영 비율이 50%일 때 검출 횟수가 0이면 50점, 1이면 25점, 2이면 12.5점이다.

4. 실험 및 평가

이 장에서는 Pylint와 SonarQube의 검출 기능을 비교한다. 이를 위해, 2023년도 1학기 Python 프로그래밍 수업에서 학생들이 제출한 4,980개 코드를 Pylint와 SonarQube로 검사한 결과를 비교하였다. 실험에 사용된 도구 버전은 Pylint 2.17.5와 SonarQube 7.2이다. 비교 실험 결과를 정리하면 표 3과 같다.

<표 3> 규칙 수를 비교하는 실험 결과

구분	Pylint	SonarQube
규칙 개수	120	97
전체 검사 개수	29,490	24,559
고유 검사 규칙 개수	47	19
고유 검사 개수	5,322	2,263

Pylint는 120개의 규칙, 29,490개 항목을 검사하였으며, SonarQube는 97개의 규칙, 24,559개 항목을 검사한 것으로 나타났다. 결과 중 중복되는 규칙이 65개이며, 비슷한 규칙을 제외한 각각의 고유 검사 규칙은 Pylint 47개, SonarQube 19개로 나타났다. Pylint가 SonarQube 대비 247% 더 많은 규칙을 검사하였으며, 고유 검사 개수만 보면 Pylint 5,322개, SonarQube 2,263개로 Pylint가 SonarQube 대비 235% 더 많은 항목을 검출한 것으로 나타났다. 이를 통해 학습자의 코드를 평가하는데 Pylint를 사용하는 것이 SonarQube보다 더 많은 정보를 제공하는 것을 알 수 있다.

5. 결론

본 논문에서는 Pylint를 이용한 Python 코딩 규칙 검사 시스템을 소개하였다. 한글을 사용하는 학습자를 위해 한글 오류 메시지를 출력하고, 규칙 검사 뷰어 기능뿐만 아니라 검사 규칙도 조정할 수 있게 하였다. 또한, 기존 시스템 채점 결과에 품질 점수를 더하여 학습자의 코딩 스타일 학습 동기를 높였다.

Pylint와 SonarQube 오류 검출 기능 비교 결과, Pylint가 SonarQube 대비 247% 많은 규칙과 235% 많은 검사 개수를 출력하는 것을 확인하였다. 이를 통해 시스템에 적용한 Pylint가 학습자에게 더 나은 코딩 규칙 학습 기회를 제공함을 알 수 있었다.

향후에는 Java나 C와 같은 언어에도 코딩 규칙 검사 기능을 추가할 계획이다. 이를 위해 언어별 표준 코딩 스타일을 분석하고, 언어에 적합한 규칙 검사 도구를 찾아 이를 적용할 예정이다.

참고문헌

[1] Y. Prokop, O. Trofymenko, and O. Zadereyko, “Developing students′ code style skills,” in 2023 IEEE 18th International Conference on Computer Science and Information Technologies (CSIT), Lviv, 2023, pp.1-4.

[2] GV. Rossum, B. Warsaw, and A. Coghlan, “PEP 8 - Style Guide for Python Code,” [Internet], https://peps.python.org/pep-0008/, 2013, last visited on April 9.

[3] P. Sassoulas, “What is Pylint?,” [Internet], https://pylint.readthedocs.io/en/stable/index.html#what-is-pylint, 2023, last visited on April 9.

[4] C. Vassallo, S. Panichella, F. Palomba, S. Proksch, HC. Gall and A. Zaidman, “How developers engage with static analysis tools in different contexts,” in Empirical Software Engineering, Vol.25, Issue.2, pp.1419-1457, 2020.

[5] J. Cheon, Y. Kim, X. Liu, I. Wang, S. Byun, and G. Woo, “neoESPA I/O Data Generating System Using QuickCheck,” in 한국정보과학회 학술발표논문집, Jeju, 2021, pp.1565-1567.