DrPython–WEB: a tool to help teaching well-written Python programs

Tommaso Battistini[1], Nicolò Isaia[1],
Andrea Sterbini[1][0000−0002−5361−2786], and
Marco Temperini[2][0000−0002−8597−4634]

[1] Computer Science Dept. - Sapienza University of Rome - Italy
sterbini@di.uniroma1.it
[2] Dept. of Computer, Control and Management Engineering
Sapienza University of Rome - Italy
marte@diag.uniroma1.it

Abstract. A good percentage of students, while learning how to program for the first time in a higher education course, often write inelegant code, i.e., code which is difficult to read, badly organized, not commented. Writing inelegant code reduces the student's professional opportunities, and is an indication of a non-systematic programming style which makes it very difficult to maintain (or even understand) the code later, even by its own author. In this paper we present DrPython–WEB, a web application capable to automatically extract linguistic, structural and style-related features, from students' programs and to grade them with respect to a teacher-defined assessment rubric. The aim of DrPython–WEB is to make the students accustomed to good coding practices, and stylistic features, and make their code better. There are other systems able to perform code analysis through quality measures: the novelty of DrPython–WEB, with respect to such systems, is in that it analyzes also linguistic and stylistic features.

Keywords: Teaching programming · Python · Feature extraction · Good coding practices.

1 Introduction

One of the main tasks of a computer programming course is to allow the students to reach an adequate level of skills, so to be able to produce well written programs, well organized, commented, readable, possibly efficient. The difficulty of accomplishing such a task is particularly felt in Higher Education in Computer Science, as students in that area will become, in a relatively close future, professionals with important responsibilities in private and public sectors [1, 3, 4].

Students' skills to produce programs showing good or even high "quality" are acquired through practice and are applied to various aspects of programming, ranging from the capability to define suitable algorithms to solve a given problem, through to the ability to design a program and the relevant data structures,

to practical coding abilities that allow a student to produce a readable program, i.e. a program whose instructions are 1) textually formatted in a readable fashion, 2) easy to interpret, as far as their purposes are concerned, and 3) as clearly commented as possible.

Both learning and teaching of Computer Programming are challenging tasks, when the traditional approach to education is used [5]. Hence, the availability of web-based automated support can be of great value, especially in Higher Education, where often direct interactions between a student who is solving a programming task, and a teacher who could help, are not easily achievable [7], especially in the case of Italian university courses where the student/tutor ratio is very high. E.g., in our courses (at Sapienza University of Rome) the usual number of students per teacher in the initial courses is 150, which is possibly due to 150 being the maximum allowed number of students per teacher by law.

Many researchers have recognized the impact of the students' programming style on their grades and have developed systems to recognize stylistic features of programs[14, 16, 18], which are usually mainly focused on programming constructs. We follow this line of research by considering also linguistic features to better recognize if the program, considered as a written text, is readable and easy to understand and to maintain.

Moreover, we want to use the results of our style/readability assessment as a didactic tool to accustom students to write readable code. To this aim we chose to use rubrics to automatically assess students' submissions, like it has been done for Appinventor[13, 2]. The teacher defines the rubric to focus the students' attention on the most important programming, stylistic and linguistic features.

In this paper, we present a web-based system, DrPython–WEB, whose use could help a student improve her/his coding skills, by pointing out and recognizing the "elegance" of the student's code in an automated and real-time fashion.

By "elegance" we mean a subset of the several qualities of a program, mentioned earlier, related to structure, readability and maintainability. On these aspects DrPython–WEB focuses its program analysis, and evaluation. In particular, given a program, the analysis is performed on a set of features, extracted from the program (see later), as well as on the good naming quality of the identifiers (i.e., the names given by the programmer to certain structures of the program, such as types, variables, and functions).

DrPython–WEB is clearly inspired to the DrScratch system[19] and other rubric-based systems[13], which statically analyze the submitted Appinventor programs to extract program features and highlight higher levels of competency on different topics (e.g., usage of more complex data structures, complex conditionals, parallelism etc.).

Similarly, DrPython–WEB measures the usage of many Python constructs with the aim of recognizing more expert programmers, and produces a teacher-defined rubric-based assessment to invite students to learn and to use the more advanced Python constructs.

Beside the focus on programming competency, we want also to accustom students to write good-style code (modularized, readable, well documented). During our lessons we use always a "describe first - implement later" development methodology and with DrPython–WEB we want to assess both the expertise and the style of the submitted homeworks.

The novelty of DrPython–WEB, w.r.t. other feature extraction systems [18], is in its analysis of linguistic features, type of keywords used in the documentation, their semantic distance from the exercise topics, use of self-explanatory identifiers.

We developed DrPython–WEB with a twofold aim: on the one hand we would like to encourage students to practice and improve their coding style; on the other hand we wanted to support both student's awareness and teacher's assessment procedures, by providing them with visual summaries of data, reporting the elements on which the overall evaluation of the code was based.

We haven't yet used the system in class but we are going to experiment with it in the next courses. Therefore we cannot yet present a comprehensive analysis of the actual effects of its use for the students and teachers.

So, in this paper we present the system, and its features, showing how we used it on a relatively large dataset of programs (produced by students during a recent edition of a course on *Basics in Programming* held at our University). Such dataset is made of programs produced to solve tasks related to the several mandatory homework requested during the course, and the solutions submitted for final exams. The programs available spans 4 years and 2 parallel courses, with 4 mandatory homeworks and 4 optional, for an average of 350 students a year, and an approximate total of 10000 homework submissions and 1000 exam submissions. These homework and exams, until now, have been graded w.r.t. their correctness through unit tests. A small bonus was awarded to students that implement faster and/or less intricate (with smaller cyclomatic complexity) code. With DrPython–WEB we want to start awarding a bonus also to more elegant code.

The main goals in this paper are then the following:

Goal 1: *To show that DrPython–WEB can automatically extract the stylistic features of a program, and assess their usage to push students towards a better programming style.*
We will see that DrPython–WEB is able to 1) perform an automatic check of the hundreds of programs in our sample, 2) analyze, in such programs, the coding qualities we associated above to "elegance", and 3) express a quality grade for each program.

Goal 2: *To show how DrPython–WEB supports personalization of the assessment depending on the teacher's preferences, each stage in the course, or just the specific assignment's characteristics.*
In this respect, we will see that the analysis performed by DrPython–WEB can be configured by the teacher, who is able to finely-tune the assessment by specifying in a rubric her/his preferences about the features to be taken into consideration, and their weight in the computation of the overall quality

grade. The metric for the overall grade is the result of the weights chosen by the teacher for the rubric's features. Notice that, as features are normally difficult to normalize, so we usually award bonuses by first ranking the grades obtained and then by selecting the highest performing 50% of them.
In particular, the possibility to configure many assessment rubrics allows the teacher to adapt the analysis of a given batch of programs, depending on the relevant characteristics of a given task, and/or the aspects to be taken care of at a given point-in-time of the course.

We plan to add the DrPython–WEB rubric-based stylistic self assessment to our Q2A-I system [6]. In Q2A-I students self-assess their python exercises, which are tested with unit tests, and receive bonuses for faster solutions and/or for less intricate programs (with lower cyclomatic complexity). In Q2A-I students participate in a formative peer-assessment phase where they suggest each other how to improve their algorithms by reading and commenting each other's algorithm descriptions.

We are still working on the analysis to understand if the already available data, collected in the previous years, shows that the linguistics features are linked to the exercise and exam grades (notice that in the previous years we have not asked the students to write nice code because of the difficulty of automatically checking for stylistic properties). No easy linear or monotonic relation seems to arise from our initial analysis yet. This is expected, as the student's population is made of several groups of students with different skills and behaving differently.

In the following sections we will:

1. Present the software library *DrPython*, which we developed to provide core functionalities for the analysis of a program: using these new functionalities DrPython–WEB was developed.
2. Present the use of DrPython–WEB on a set of sample programs, in order to show the characteristics of the system and see its potential application on the field.
3. Present some conclusions, submitting that DrPython–WEB, although subject to further improvements, can be an effective means to help the students to improve their coding style.

2 DrPython: feature extraction module

DrPython–WEB is based on the feature extraction library we developed for this task (named DrPython). We chose to base the system on feature extraction because features allow for an expandable and easy to understand definition and description of assessment rubrics. Others have used feature extraction on Python programs (e.g. [18]) to extract textual and structural features from programs. To these type of features we add linguistic features (see below).

We developed the DrPython library to analyze the student's program and algorithm description to recognize/extract three types of features:

- **Code syntax features:** the number of specific language constructs in the program (functions, classes, super-classes of each class, methods, try-except, list-comprehensions, if-then-else, generators, lambda, recursive functions, variables, arguments),
- **Code quality measures:**
  - McCabe's cyclomatic complexity [8], that captures how much a function control flow is intricate,
  - Halstead's measures [9], that captures a function's conceptual complexity from its vocabulary size and number of operators used,
  - *Code smells* [10], i.e., code structures that often imply bad coding practices.
- **Linguistic features:**
  - *Good identifiers*, i.e., self-explanatory names that convey the meaning of their function. This relieves the programmer from having to recall what type of data is in a variable and what its place is in the algorithm, as well as the action performed by a function/method,
  - *Good documentation practices* i.e., using comments and doc-strings to describe the reason for particular programming choices. This helps the reader to better understand the meaning of the algorithm implemented.
  - The usage of *pertinent keywords* related to the exercise description both in comments/doc-strings or in the algorithm description. This allows DrPython to automatically check (roughly) if the documentation is adequate to the task.

The code syntax features are extracted/counted by means of the **redbaron**[3] source code analysis library that allows to easily query the code structure for specific constructs. Redbaron queries use a syntax similar to CSS selectors (as it's done in jQuery w.r.t. the DOM of HTML pages). This in turn will allow us to easily expand in future the set of code syntax features extracted.

The code quality measures are computed by means of the **radon**[4] library which computes the code metrics: Mc'Cabe cyclomatic complexity of each function, Halstead measures, SLOC, comment count and other simple code metrics.

Finally, to extract the linguistic features DrPython uses the automatic term extraction module **pyATE** [11] to select the 25 highest ranked keywords returned by its **Combo Basic** algorithm [12], and the text analysis library **spacy**[5] to analyze the documentation/comments and the algorithm description. To decide if a particular identifier used by a student is of good/medium/bad quality, DrPython performs the following steps:

- It extracts the pertinent keywords with pyATE from the teacher's exercise task description
- It decomposes the identifier into its component words
- It compares the words (by means of spacy semantic similarity and the WordNet semantic network) to grade their similarity to the pertinent keywords

[3] https://redbaron.readthedocs.io, accessed 1/11/21
[4] https://radon.readthedocs.io, accessed 1/11/21
[5] https://spacy.io, accessed 1/11/21

– It classifies the identifier in the top/medium/bad group depending on having its max similarity to a keyword above 90%, between 40% and 90% or lower than 40%, respectively. We have initially chosen the thresholds to split identifiers in the top/medium/bad classes as the 90% and 40% values, i.e. we consider a very good identifier to be very similar to the exercise keywords, a good identifier sufficiently similar, and a bad identifier rather dissimilar from the keywords. A more detailed study of the data to find the best threshold will follow.

DrPython can be used both as a stand-alone program, to be run from the command line, or integrated in the DrPython–WEB web-based application described below.

For example, with DrPython one could analyze many student files and collect all extracted features as a CSV file and study, for example:

– How the extracted features correlate with each other or with other data (exam grades or readability judgements manually collected)
– How different assessment rubrics will produce different grade distributions

To make the assessment rubrics easier to use, and to automate the submission and assessment of the programs, we have developed the web-based application (DrPython–WEB).

3 Dr.Python-WEB: The System

The DrPython–WEB system allows the teacher to define one or more **assessment rubrics** to grade the submitted programs/algorithms depending on the features extracted, in order to encourage students to use more readable Python constructs, a better linguistic style, and to better modularize their code.

DrPython–WEB is a classic LAMP[6] based web-application written in Python where:

– The teacher defines assessment rubrics depending on the exercise and/or the course phase.
  We want to allow the teacher to define different rubrics in different phases of the course (or even for specific exercises). This way the teacher would be free to, for example, assign more weight to course topics/python constructs that have been recently explained, or to python constructs that are particularly effective to solve efficiently the specific exercise.
– The students submit their code to get the style assessment grade and compare their results with each others'.
– The teacher has an overall view of the students' leaderboard and a detailed view of all submitted programs and assessment results, and thus has the ability to finely tune the rubric in case the assessment is distorted, and update the assessment.

[6] LAMP=Linux, Apache, MySQL, PHP/Perl/Python

Homework 1 (Not mapped)	Edit template	Show me all student results	Show me all student evaluations
Feature	**Weight value**	**Minimum range**	**Maximum range**
Effort	1.5	5.0	100.0
Effort	1.0	100.0	500.0
Effort	0.5	500.0	5000.0
Cyclomatic complexity	0.8	2.0	15.0
Cyclomatic complexity	0.5	15.0	30.0
% Good identificators	1.2	70.0	100.0

Fig. 1. Assessment rubric that awards more points for lower cyclomatic complexity, lower Halstead's Effort and high percentage of good identifiers: see further explanations in the following text.

Assessment rubrics are defined by the teacher by specifying what are the features assessed, and what are their weights associated to given ranges of their values.

In Fig. 1 we show an assessment rubric that awards more points to a lower *Halstead's effort* ('Effort' in the figure), to a lower *cyclomatic complexity*, and to a higher *percentage of good identifiers* depending on the ranges observed for their values. Notice that, in general, each feature ranges over non-normalized interval of values. We can imagine that a teacher would built a rubric like the one in figure to spur students to: modularize their program into smaller less complex functions (with lower cyclomatic complexity), to write more readable code (using mainly self-explanatory identifiers), and with a less complex algorithm (with lower Halstead's effort).

Notice that an assessment rubric can assign different points to different ranges of feature values extracted, as shown in the figure, where we show three different ranges for the Halstead's Effort measured. This way, the teacher could associate to each feature a weight function with complex shape.

The teacher can update the rubric by either changing the ranges of application or the points given for each feature/range rule or by adding/removing new feature/range/points rules to the rubric.

As the features measured are normally heterogeneous, we plan to study the distribution of the extracted features over our dataset to propose standardized ranges to the teachers and help them during rubric definition.

All the programs are made available to the other students after the submission deadline. After assessment the students' results and programs are shown and linked in the DrPython–WEB leaderboard, so that each student can compare their program style with others, as shown in Fig. 2

Exercise 1								
		% Good identificators		Cyclomatic complexity		Effort		
Student	Name	Feature value	Feature evaluation	Feature value	Feature evaluation	Feature value	Feature evaluation	Total
student0	program01.py	85.71	1.2	3.0	0.8	49.76	1.5	3.5
student2	program01.py	20.0	0.0	2.0	0.8	6.96	1.5	2.29
student3	program01.py	42.85	0.0	3.0	0.8	241.76	1.0	1.8
student4	program01.py	11.11	0.0	5.0	0.8	143.25	1.0	1.8
student6	program01.py	25.0	0.0	4.0	0.8	30.31	1.5	2.29
student7	program01.py	11.11	0.0	4.0	0.8	68.33	1.5	2.29
student8	program01.py	80.0	1.2	4.0	0.8	167.59	1.0	3.0
student12	program01.py	60.0	0.0	3.0	0.8	182.64	1.0	1.8
student13	program01.py	60.0	0.0	4.0	0.8	61.02	1.5	2.29
student14	program01.py	23.07	0.0	6.0	0.8	52.0	1.5	2.29
student15	program01.py	20.0	0.0	4.0	0.8	48.0	1.5	2.29
student16	program01.py	20.0	0.0	4.0	0.8	15.5	1.5	2.29
student17	program01.py	50.0	0.0	4.0	0.8	30.31	1.5	2.29
student18	program01.py	50.0	0.0	3.0	0.8	474.54	1.0	1.8
student19	program01.py	60.0	0.0	4.0	0.8	69.3	1.5	2.29

Fig. 2. Leaderboard example, showing the features checked for this exercise and the points assigned according to the previous assessment rubric. Notice that the only features shown are those included in the assessment rubric.

4 Conclusions and future work

We have shown a novel library (DrPython) which extracts structural, quality and linguistic features from the programs and documentation submitted by students. DrPython is used within the novel DrPython–WEB application, that allows the teacher to build assessment rubrics specific both to the point in time during the course and/or to the specific exercise.

We plan to use the DrPython–WEB system on our next courses to collect data on the student's submissions and check that its usage improves the student's program quality.

This will allow us to see if the correlation between features and grades will improve (and in what shape) when the system is in-place, with respect to the data collected in earlier courses.

We will try to detect cheating patterns (i.e., when students try to gain points with simple strategies without actually improving their programming style) and to make the feature extraction more robust/precise with respect to cheating. Yet, we are convinced that the effort to "fool the teacher" could, in any case, increase their technical programming skills.

Moreover, we plan to collect readability assessments from the students during the course to study both how the exercise readability improves with time and how the code readability perception of the students changes while they are learning.

From the collected data we intend to study if we can define a program readability measure that takes into consideration the linguistic features also.

Finally, we intend to study how the readability of a program is related to its grade, and/or to the grade received in the final lab-based exam. From our initial analysis, it seems that a simple linear (or monotonic) relation between features and grades is not evident. Thus, we are widening the number of processed programs to apply also clustering and/or ML approaches (which need more data and time).

References

1. Breuker D M, Derriks J, Brunekreef J. Measuring static quality of student code. In: Proceedings of the 16th Annual Joint Conference on Innovation and Technology in Computer Science Education. 2011, 13–17
2. Caiza, Julio C., and Jose M. Del Alamo. Programming assignments automatic grading: review of tools and implementations. 7th international technology, education and development conference (INTED2013). 2013.
3. Yao LU, Xinjun MAO, Tao WANG1, Gang YIN, Zude LI. Improving students' programming quality with the continuous inspection process: a social coding perspective Front. Comput. Sci., 2020, 14(5): 145205 https://doi.org/10.1007/s11704-019-9023-2
4. Radermacher A, Walia G, Knudson D. Investigating the skill gap between graduating students and industry expectations. In: Proceedings of the 36th International Conference on Software Engineering Companion. 2014, 291–300
5. Feldman Y A. Teaching quality object-oriented programming. Technology on Educational Resources in Computing, 2005, 5(1): 1
6. Papandrea S., Sterbini A., Temperini M., Popescu E. (2018) Q2A-I: A Support Platform for Computer Programming Education, Based on Automated Assessment and Peer Learning. In: Hancke G., Spaniol M., Osathanunkul K., Unankard S., Klamma R. (eds) Advances in Web-Based Learning - ICWL 2018. ICWL 2018. Lecture Notes in Computer Science, vol 11007. Springer, Cham. https://doi.org/10.1007/978-3-319-96565-9_1
7. Chen W K, Tu P Y. Grading code quality of programming assignments based on bad smells. In: Proceedings of the 24th IEEE-CS Conference on Software Engineering Education and Training. 2011, 559
8. McCabe (December 1976). A Complexity Measure. IEEE Transactions on Software Engineering (4): 308–320. https://doi.org/10.1109/tse.1976.233837
9. Halstead, Maurice H. (1977). Elements of Software Science. Amsterdam: Elsevier North-Holland, Inc. ISBN 0-444-00205-7.
10. https://wiki.c2.com/?CodeSmell

11. Lu, Kevin. (2021, June 28). kevinlu1248/pyate: Python Automated Term Extraction (Version v0.5.3). Zenodo. http://doi.org/10.5281/zenodo.5039289
12. Astrakhantsev, N.: Methods and software for terminology extraction from domain-specific text collection. Ph.D. thesis, Institute for System Programming of Russian Academy of Sciences (2015)
13. Nathalia da Cruz Alves, Christiane Gresse von Wangenheim, Jean Carlo Rossa Hauck, and Adriano Ferreti Borgatto. 2020. A Large-scale Evaluation of a Rubric for the Automatic Assessment of Algorithms and Programming Concepts. Proceedings of the 51st ACM Technical Symposium on Computer Science Education. Association for Computing Machinery, New York, NY, USA, 556–562. DOI:10.1145/3328778.3366840
14. Chakraverty S, Chakraborty P. Tools and Techniques for Teaching Computer Programming: A Review. Journal of Educational Technology Systems. 2020;49(2):170-198. doi:10.1177/0047239520926971
15. Paulo Blikstein, Marcelo Worsley, Chris Piech, Mehran Sahami, Steven Cooper and Daphne Koller (2014): Programming Pluralism: Using Learning Analytics to Detect Patterns in the Learning of Computer Programming. Journal of the Learning Sciences, DOI: 10.1080/10508406.2014.954750
16. Rogers, S., Garcia, D., Canny, J. F., Tang, S., and Kang, D. (2014). ACES: Automatic evaluation of coding style (Doctoral dissertation, Master's thesis, EECS Department, University of California, Berkeley).
17. Nathalia da C. Alves, Christiane G. von Wangenheim, Jean C. R. Hauck, and Adriano F. Borgatto. 2020. A Large-scale Evaluation of a Rubric for the Automatic Assessment of Algorithms and Programming Concepts. In Proceedings of the 51st ACM Technical Symposium on Computer Science Education (SIGCSE'20), March 11-14, 2020, Portland, OR, USA. ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3328778.3366840
18. Z. Wang, A. Alsam, D. Morrison and K. A. Strand, "Toward automatic feedback of coding style for programming courses", 2021 International Conference on Advanced Learning Technologies (ICALT), 2021, pp. 33-35, doi: 10.1109/ICALT52272.2021.00017.
19. Moreno-León, J. and Robles, G. (2015, November). Dr. Scratch: A web tool to automatically evaluate Scratch projects. In Proceedings of the workshop in primary and secondary computing education (pp. 132-133).