Universidade do Minho
Escola de Engenharia
Departamento de Informática

Luís Martinho de Aragão Rego da Silva

Intelligent feedback system for programmer's profile improvement

January 2020

Universidade do Minho
Escola de Engenharia
Departamento de Informática

Luís Martinho de Aragão Rego da Silva

Intelligent feedback system for programmer's profile improvement

Master dissertation
Master Degree in Software Engineering

Dissertation supervised by
Prof. Pedro Rangel Henriques
Prof. Maria João Varanda

January 2020

STATEMENT OF INTEGRITY

ACKNOWLEDGEMENTS

A first note of gratitude to both supervisors of this work, Professor Pedro Rangel Henriques and Professor Maria João Varanda for applying their experience in the form of invaluable and unwavering support throughout the entire process. Their effort is visible in every major decision that was taken to shape this small contribution to the world of academia. It is also worth noting the incredible patience they showed despite all my setbacks of what, at times, might have seemed to be lack of commitment.

Also to the initial beta testers: Estevão, Hugo, Mendes and Resende; that, despite time constraints, were ready to trade feedback and discuss the underlying mechanisms of a tool that only worked in a language that was not their cup of tea.

It has been a longer than expected journey where I have been engaged in several exhilarating projects. As such, there's bound to be plenty of people that, in one way or another, motivated me to move forward. For that reason, I have to thank everyone at CeSIUM for reminding me since the beginning that I should prioritise my dissertation instead of being a volunteering masochist and accepting to be the president for a second mandate. To all those at AAUM, as although it was just a year, it was an unforgettable one at that, and those too kept nagging me to conclude this chapter. For the opportunities that Subvisual provided in launching my career and the elaborate strategies devised to make me focus in advancing, one step at a time, this work. And finally, for those at Utrust who shared my anxiety in this final year of a torn product manager who developed a dissertation in is off time.

But above all, to my family and friends from these and other communities who joined me in building something that I would be proud of.

ABSTRACT

This document is a Master's dissertation on a degree in Software Engineering, in the area of Language Engineering.

The main goal of this thesis is to support a software developer's growth by providing feedback on improvement areas based on the classification of his programming profile. Information about his profile as well as recommendations for improvement shall be extracted through the analysis of his source code.

A programmer's ability can be classified as one of four possible profiles and the distinction among them falls upon the levels of both skill and readability. By aiming at proficiency on these criteria one can achieve a more valuable profile.

As proof of concept a tool, that identifies weaknesses in a programmer's ability and provides improvement feedback, was developed using as basis Daniel Novais's *Programmer Profiler Tool (PP)* tool with a more educational approach.

Keywords: feedback, improvement, profiling, programmer

RESUMO

Este documento é uma dissertação em Engenharia Informática, na área de Engenharia de Linguagens.

O principal objetivo desta tese é suportar o crescimento de um programador providenciando sugestões baseadas na classificação do seu perfil. Informação sobre o perfil, bem como as recomendações para melhoria, são extraídas através da análise de código-fonte providenciado pelo programador.

A aptidão de um programador pode ser classificada como um de quatro possíveis perfis estando a distinção destes sobre os níveis de competência e legibilidade. Ao atingir proficiência em ambos os critérios obtém-se um perfil mais completo.

Como prova de conceito foi construída uma ferramenta que identifica áreas de melhoria, apresentando sugestões de correção, tendo sido desenvolvida a partir da ferramenta PP por Daniel Novais, com uma abordagem mais educacional.

Palavras-chave: melhoria, perfil, programador, sugestão

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF LISTINGS

ACRONYMS

C

CEFR Common European Framework of Reference for Languages: Learning, Teaching, Assessment. 1

CFS Control Flow Statements. 39

CLI Command Line Interface. 13

I

ITS Intelligent Tutoring Systems. 8, 55

K

KC Knowledge about concepts. 8

KH Knowledge about how to proceed. 8, 32, 33

KM Knowledge about mistake. 8, 32, 33

KMC Knowledge about meta-cognition. 8

KTC Knowledge about task constraints. 8

N

NSCCFS Not So Common Control Flow Statements. 39

P

PP Programmer Profiler Tool. iv, v, x, 1–5, 7, 8, 10, 12, 14, 15, 33, 43, 45–47, 49, 50, 53

1

INTRODUCTION

Feedback plays a decisive role in learning and development for any field of study (Hounsell, 2003). However, in the quickly evolving area of computer science opportunities to provide personalised feedback to programmers are scarce. Such occurs due to both the increase in enrolments but also the diversity in students' backgrounds, Often, due to the very high student to professor ratio, the only form of assessment available is summative assessment which has been shown to have low impact in student growth (Orrell, 2006). Due to these issues, several systems have been trying to fill the gap and provide more ways to help programmers improve. Even so, the options available for use in a classroom context are quite limiting, sometimes due to being tailored to specific exercises and so requiring manual configuration for more use cases or, by only considering syntactic mistakes. More experienced programmers find even more difficulty to obtain feedback as the existing tools tend to focus on the particular challenges of beginners.

The aim of this dissertation is to create a tool which is able to provide personalised feedback to programmers despite their proficiency. Through the use of this automatic improvement tips, a professor should be more equipped to understand the class, identify both negative and positive outliers, and also improve the student's personal growth. The tool should also adapt to the context in order to not require prior adjustments.

In order to achieve this goal, there is a need to understand a programmer's ability and establish a sort of grading system to distinguish novice from experienced students. For that, the parallel between proficiency in programming languages and that of natural languages is used. After all, the former has standardised ways of establishing profiles such as the *Common European Framework of Reference for Languages: Learning, Teaching, Assessment (CEFR)*. This has been explored in (Novais, 2016) where the *Programmer Profiler Tool (PP)* was constructed. The PP analyses the source code of a set of correct solutions to a given programming problem, written in Java, by different programmers and infers a profile from the following attributes: Skill and Readability. Skill is related to language knowledge and adequate use of the algorithms while readability is about documentation of code and best coding style practices. The scoring system compares the scores for the attributes among all the programmers as such, it scores a solution differently depending on its peers.

For those reasons, and as it is open source, it was decided to develop on top of the PP and add an intelligent feedback system to strive for programmer improvement. Two main topics are to be explored:

- The author concluded that since the scoring system is too heuristic the accuracy of the results when considering different contexts could be greatly improved. Hence a machine learning approach was proposed.
- How to provide feedback. This will be the core of the topic explored as the tool had been built in a way that it could be used in different subjects but was not prepared for any of them yet. The focus will be in making it useful in the formal education system or as a complement to it.

For the first one there was an intention to explore a dataset such as the one provided by CodeWars[1]. In this website, the community evaluated solutions to a problem on two criteria "Best Practices" and "Clever" which could possibly translate directly to the mentioned programming perspectives. Unfortunately, a dataset that could be used for data mining wasn't available. Even though it could be an option to build one, to be useful, it would require hundreds of assessments done manually before providing results. Instead, the decision was to use a dataset with solved exercises in Java that already filtered the ones that worked as the tool is only capable of processing compilable solutions.

Regarding the second topic, the adaptation of the PP tool to an educational context will open the possibility to provide feedback for improvement, which can be contextualised to the programmer's profile. To provide an example to the previous point, a novice wouldn't gain much from being told about the most complex data structure when he might not even grasp the most basic ones. Whilst, on the other hand, an expert wouldn't benefit from improving a lot on his skill when he could gain much more from readability recommendations. This shows the potential of directly applying the inferred profiles from the source code in directing and adjusting personalised feedback.

A further step could be done where a graphical interface would be built and thus allow the use of the tool as an educational complement to be available for students outside the educational system. Perhaps it would have to be adjusted to be more efficient in a self-learning situation.

The approach presented is ambitious and challenging yet could greatly contribute to reduce the issues that professors are constantly facing. However, programming languages are in constant development and some practices can be, at times, subjective. With that in mind, the purpose of this research is to grasp the basics of the topic and adapt the PP to be used as an auxiliary tool in teaching.

1 https://www.codewars.com

1.1 OBJECTIVES

This master dissertation has the following objectives:

- to expand the PP to produce accurate results in more demanding scenarios, such as with hundreds of peers instead of just classroom context;
- to discuss and choose relevant feedback depending on the specific areas of proficiency;
- to explore and analyse how to motivate students to improve their attributes;
- to produce improvements tips based on the characteristics of the programmer while leveraging the profile inferred from his source code;

To achieve these objetives, a tool will be developed that can be used by a professor or deeply customised by any programmer. Thus, it should be fairly easy to extend it for other uses by building on top as it is open-source code. The same is what will be done on top of PP.

1.2 RESEARCH HYPOTHESIS

Through the analysis of a programmer's source code against that of his peers it is possible to deliver personalised feedback adjusted to the particularities of his profile. This should complement the formative assessment in the formal education system in a way that supports and motivates a student towards self-improvement. The achievement mechanism inherent with the profiles scoring could be the basis towards overcoming the limitations of a low professor-to-student ratio.

1.3 DOCUMENT STRUCTURE

This dissertation is divided in seven chapters:

1. **INTRODUCTION** A brief explanation of both the context and the goals of the dissertation.
2. **STATE OF THE ART** In this chapter an overview is done of the current research in related work. Furthermore, a survey is done of where the extended tool fits amongst the other alternatives.
3. **APPROACH** On the third part the proposed approach to achieve the introduced goals while analysing the challenges faced. The architecture of the overall system is also documented.

4. **SCALING UP SYSTEM** An overview of the decisions and changes done to improve the capacity of the tool to handle different and more demanding contexts. There, the details of the changes caused by the alterations done are analysed.

5. **PROVIDING FEEDBACK** Similarly to the previous chapter, this one dives into the core of the system which is it's ability to provide personalised feedback to each programmer analysed.

6. **CASE STUDIES** Afterwards the combination of all the decisions previously discussed into the final result capabilities of the upgraded PP. That is done by exploring a few examples and also collecting some outside opinions of the impact this might have *in loco*.

7.**CONCLUSION** Finally outputting what was achieved in this dissertation in relation to the initial objectives and hypothesis while also presenting possible paths for improvement.

2

APPROACHES TO SUPPORT PROGRAMMER'S GROWTH

Throughout this chapter two things will be explored: the studies and approaches to both rank and evaluate programmers and also the ways in which tools can support one's growth. As the goal of this dissertation is to increase existing functionality of (Novais, 2016)'s programmer profiling tool, it was further necessary to analyse research since 2016 specifically around that topic.

2.1 ASSESSMENT OF PROGRAMMER SKILL

Whilst focus has been on adapting PP for educational use, assessment of programmers is a wider topic that is applied in different contexts. After all, formal education is but part of a software engineer's career, and his initial grade tends to be just one of a long list of possible factors to analyse. As such, recruitment has been seen as one of the most diverse moments where each company and recruiter, often subjectively. ranks programmers.

Two main approaches to evaluation have been identified, these usually add value to each other and are fairly generic across technical disciplines like other engineering areas. These will be explored in the following subsections.

2.1.1 *Challenges and Exams*

In (Daly and Waldron, 2004) two assessment methods which are considered the standard in education are analysed, these are:

1. Programming Assignments
2. Written Exams

For the first one, it has been identified that 40% of students had plagiarised at least one assignment, making its purpose unreliable. As for written exams, the authors pointed that ”examiners often see how a weak attempt could be improved to produce a solution and so provide credit with foresight that frequently is not there”. Instead they propose lab assignments should be the default option of assessment and tools like Coderunner (Lobb and Harlow, 2016) apply that same knowledge.

During recruitment, programming assignments are also used frequently such as technical interviews. Yet there are more ways to assess a programmer than just those used by recruiters and professors. In fact, some approaches make use of *gamification* techniques to do so. *Gamification* is used often to boost interest in learning, as explained by (Swacha and Baszuro, 2013), and has been applied in almost all code challenges platforms such as *CodeWars*[1]. In this kind of platforms, which there are dozens of, programmers are ranked based on number and difficulty of programming challenges that they solved, with leaderboards to compare with other players. These features are further explained on (Fuchs and Wolff, 2016). It is viewed as an achievement system in which the best programmers are the ones who have solved more challenges. Often, what this means is that they spent the most time and effort yet it doesn't contemplate the approach they used.

This has become an go-to resource for some companies, as they can prospect talent directly from the top of a leaderboard. Or they can filter candidates by offering the same challenge to all and comparing who solved it more quickly or in the best way, in fact there are even mechanism of what is called *Player vs Player* where multiple programmers compete online for whom solves a few exercises quicker.

As such, due to their versatility, challenges and exams have been the way of assessing the attributes of a programmer. These are the only available way to use for novice programmers hence in an educational environment there is no other option.

2.1.2 *Experience*

Perhaps the most traditional for other disciplines, experience is hard to measure as the standards greatly vary. Furthermore, having experience does not imply that one can code well. This is especially true as the industry is constantly changing. For recruitment purposes experience is written through a *curriculum vitae* and validated through recommendation letters and explored during interviews. Unfortunately due to all these factors, evaluating experience tends to be very subjective and although it is one of the stages for recruitment it is usually paired with other tools for evaluation.

Experience is still heavily taken into account to subjectively rate a programmer. It is also often paired with other evaluation methods to guarantee trustworthiness of the sources.

There are novelty or niche ways to analyse experience such as the activity of a user in online tools. Software communities have accumulated data for many of their users. Open Source Communities like *GitHub* track number of contributions and when they occur, which relate to the interests of the programmer and his popularity. On the other hand, platforms

1 https://www.codewars.com/

like *StackOverflow*, where the community answers questions and is rated by the peers, offer a way to further analyse their profile. An approach has been proposed by (Huang et al., 2016) whichm using a tool called CPDScorer, combines information from both these sources in order to identify programmer ability and interests claiming ao 80% precision. This study, although using very different input fields, can be an interesting guideline both due to its precision and to the use of data mining and machine learning which will also be used in this work.

However, since this information can not be extracted from source code analysis and students tend to not have work experience, this topic won't be explored further.

2.2 PROFILING PROGRAMMERS

The current grading system used in education is one-dimensional, it evaluate students' performance on a scale of passing to failing. Yet there are multiple characteristics that can be perceived and so should be evaluated separately.

As proposed by (Pietriková and Chodarev, 2015) one can profile a programmer through source code analysis into two relevant profiles: subject and object. Subject profile is described as the base knowledge a programmer has, such as whether he knows how to use the conditional construct of a specific language. Object profile represents the profile of knowledge necessary to handle a specific task. The tool developed searches for the mismatch between these two profiles to identify the profile of the programmer. As the analysis is done based on the source code it means this can be automated and remove some of the subjectivity or infered knowledge previously observed in some evaluation approaches. The tool developed by the authors required optimal solutions to be used for comparisons, which can often prove to be a limitation.

On the other hand PP uses two attributes: skill and readability (Novais et al., 2016). The former describes the language knowledge, such as advanced constructs but also the simplicity of the algorithms used. The latter analyses the effort placed into ensuring the code can be easily understood by another developer. Due to supporting two different attributes, one can identify students which show skill from those with readability and also trace ideal profiles with balanced scores. This would not be achieve in the one-dimensional grading system.

In (Paterson, 2017) readability is considered an important aspect to support learning and concluded that instructors would benefit from having the means to evaluate, in a simple and automatic way, this attribute. A study based on that initial exploration was done where (Hofmeister et al., 2017) it was shown that experts and novices follow different gaze paths when reading code. For instance, that novices saccades were shorter and more focused on 52.4% of the elements, versus 41.3% compared to experts. A deep research was conducted by (Scalabrino et al., 2019) on how to automatically access this attribute. In it, 121 metrics were analysed, some of which had already been configured for use in the PP. This article was done specifically for Java so it faced many of the same specific challenges. However they conclude there is currently a lack of developer-related metrics.

2.3 DELIVERING FEEDBACK

There are also a fair share of platforms and services that analyse source code either to offer feedback on improvements, such as reducing duplicate code, or to expose vulnerabilities.

For the former situation refer to *Codacy*[2] which rates the current codebase, and could therefore extrapolate the average evaluation to the programmer himself. Furthermore tools such as linters, which are approached in further detail in (Fast et al., 2014), have become more frequently used in IDEs. (Flowers et al., 2004) also discuss an automated tool named Gauntlet which corrects syntax errors in Java. All of these tend to automatically correct mistakes and align with the best language practices instead of guiding the individual on how to improve.

In the latter case, *Checkmarx*[3], could come as an example. This platform tests for security issues and provides feedback accordingly.

The examples above focuses on ensuring program quality instead of aiming to improve a programmer's skill to not repeat the same mistakes. The tools to help boost learning are currently usually *Intelligent Tutoring Systems (ITS)*. A systematic overview of these tools was done by (Crow et al., 2018) and it is concluded that "a lot of work can be put into developing intelligent feedback and hints relating to semantic and syntactic issues in programming tasks".

Pursuing feedback directly, as analysed by (Keuning et al., 2016) the nature of feedback can be segmented to 5 types. Where *Knowledge about task constraints (KTC)* and *Knowledge about concepts (KC)* both require prior configuration per exercise, limiting the versatility of the tool. Out of the examples mentioned above which are analysed from the source code, clearly most of them naturally go towards *Knowledge about mistake (KM)* and *Knowledge about how to proceed (KH)*. Finally, one last type is mentioned which is *Knowledge about meta-cognition (KMC)* but it very scarcely used and requires a lot of specialisation to achieve it.

There has also been research around the impact of personifying feedback such as in (Lee and Ko, 2011). Here, once again, *gamification* was employed through the use of a robot in a game which helps present the mistakes identified in a softer way. It was concluded that by personifying the tool, feedback can increase novice programmers' motivation to program.

2 https://www.codacy.com
3 https://www.checkmarx.com/

Overall there are multiple approaches to feedback, but it is identified that adapting it to the specific programmer and applying *gamification* can be of great value. However, automatic feedback is often limited in the ability to adapt to the context.

3 APPROACH

During the previous chapters it was shown that there is currently a substantial gap in providing feedback to programmers not only to students in higher education but also across the board. Furthermore it was identified that, by improving personalised feedback, students could be more motivated to learn.

As such, this chapter will explain the proposed solution by exploring its challenges and the approach used to tackle them.

3.1 PROBLEM DEFINITION

By focusing in higher education it has become clear that, right now, there is a need to overcome the tendency to resort only to summative assessment. Professors, have a lack of tools that can provide students individual feedback while simultaneously being flexible enough to not require changes in the teaching or evaluation method. These are the requirements that the solution proposed must comply to. What is thus proposed is a solution that can:

- analyse the source code of any solution;
- adapt to the context, such as a specific class of students;
- provide personalised feedback;
- motivate the student to improve;
- work out of the box without prior exercise configuration.

3.2 EXPANDING THE PROGRAMMER PROFILER TOOL

To achieve such requirements, namely the adaption to the context without requiring prior configuration, the solution must have an inference engine. For that reason, the Programmer Profiler Tool developed by (Novais, 2016) was a great basis to work on top of. The Programmer Profiler Tool, which has been commonly named PP, is an open source profiling tool capable of processing exercises in the Java language. It was developed as part of a Master's Dissertation in Software Engineering at Minho University. The code itself can be found on GitHub[1].

By taking as input a set of correct solutions to a problem and performing static analysis, it can infer a profile for each of the programmers. The profile is calculated by comparing the results among each other on two distinct attributes: skill and readability. The existing profiles are:

- **Novice**: Low Skill and Low Readability;
- **Advanced Beginner R (Readability)**: Low Skill and Average Readability;
- **Advanced Beginner S (Skill)**: Average Skill and Low Readability;
- **Advanced Beginner + (Both)**: Average Skill and Average Readability;
- **Proficient**: Low-to-Average Skill and High Readability;
- **Expert**: High Skill and Low-to-Average Readability;
- **Master**: High Skill and High Readability.

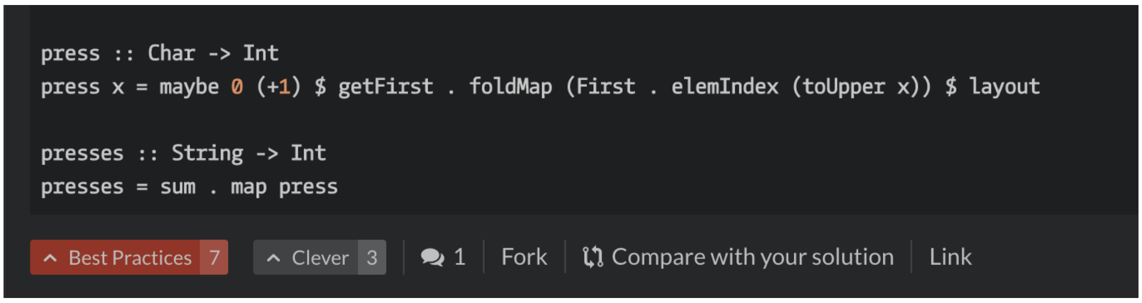
The information obtained can thus support the personalisation of feedback. By knowing the type of profile a programmer has, it is possible to adapt the available feedback. For instance, a programmer with a lot of skill, but low readability, could be directed to a book such as Clean Code by (Martin, 2009). Whilst such resource might be too overwhelming for a programmer which still lacks both skill and readability.

3.3 HOW PROFILES AND ATTRIBUTES FEED THE FEEDBACK SYSTEM

Personalised feedback can be very beneficial to students (Lee and Ko, 2011). Furthermore, the expectations of one group's performance can be completely different from others. By being able to pick on these nuances of context to provide adequate feedback is the mission of an intelligent system.

One possibility to achieve it would be by doing an artificial intelligence approach. For example, by using a similar input to CodeWars's community evaluation where they rate solutions for "Best Practices" and "Cleverness". In figure 1 both labels can be seen, and the conversion to our existing attributes is almost immediate. "Readability" for the "Best Practices" button and "Skill" for the "Clever", which were the defining criteria in (Novais, 2016). Such an approach using data-mining techniques, was explored in (Kagdi et al., 2007).

1 https://github.com/danielnovais92/ProgrammerProfiler



```
press :: Char -> Int
press x = maybe 0 (+1) $ getFirst . foldMap (First . elemIndex (toUpper x)) $ layout

presses :: String -> Int
presses = sum . map press
```

Figure 1.: Community based evaluation of two criteria

Instead during this project the approach was to use the information already generated by PP during (Novais, 2016) to adapt the feedback. The information available is:

1. programmer's profile;
2. score in readability;
3. score in skill;
4. PMD[2] violations and their impact;
5. score obtained by each metric analysed.

A data structure with the compilation of all this information was created, using some auxiliary data like the impact in percentage of violations or the highest obtained result for some metrics. For each programmer with an exercise solved a feedback file is generated by doing the following steps. From the profile (1) the attribute to focus on is obtained. After all, the profiles with lower readability than skill should focus on the former attribute and vice-versa. If the profile is balanced, like an Advanced Beginner +, then the feedback which can have the most impact in either attribute is selected.

With the knowledge of which attribute to prioritise, identify the violation (4) which had the highest negative impact to the relevant score (2 or 3). This means the programmer will always be applying the least effort for the maximum gain. Since violations tend to be easy to solve, a simple suggestion about the most impactful metric (5) is also provided.

If no violation is available then feedback will only focus on the metric (5) with the most impact, but with more detail then if a violation had been identified.

Afterwards, a full section on motivation is generated by comparing the difference of score for both (2 and 3) if the suggestions were followed. This might even mean the programmer was upgraded to a better profile.

2 pmd.github.io

3.4 SYSTEM ARCHITECTURE

In figure 2 the architecture of the full system can be seen. It is done with the professor as an user. As can be seen, the input is a compilation of solutions by programmers from one or multiple classrooms depending on the context that wants to be analysed.

From the *Command Line Interface (CLI)*, the professor can identify the folders with exercises to be analysed and also provide a base solution for each. If, multiple folders should be processed separately, by default, the base solution will be the first file on each the directory. In that case, the systems used multi-threading to improve efficiency of analysis.

The solutions are then processed by both the PP Analyser (using AnTLR) to extract relevant metrics and also, PMD Analyser (using PMD) to obtain violations to the predefined rules. This information is then used to calculate scores by making comparisons between all the solutions and also the base solution. With the scores, the profiles can then be inferred and are added to a general data structure with all data obtained.

This data is then used simultaneously to generate feedback and for multiple exports. The feedback generation has a few steps:

- Generate general information about the purpose of the tool;
- Identify attribute to focus on;
- Provide main suggestion;
- Provide secondary suggestion (if main one is a violation fix);
- Show impact of the suggestion fix;
- Show progress towards better profiles.

The outputs are both individual feedback files per programmer and an overview of the metrics and profiles obtained through the analysis. The professor could intervene manually on the feedback files for the students either to use them to adapt teaching methods or to forward them to students. An interface could have been built to avoid needing to use the CLI.

3.5 REQUIREMENTS

In Novais (2016) there are several proposals for improvement of the tool. For instance, processing syntactic and semantic errors. However improving the source code analysis itself has not been labelled as a necessity to validate the hypothesis as the focus was on adapting the tool to be used in education and so provide personalised feedback. Even so, the scalability in this project was improved to better handle more varied environments.

Overall this means there is currently a set of requirements to use PP successfully, most of which were inherited.

First, it is limited to Java version 7 as the new constructs of Java 8 were not considered. As the purpose was to validate the ability to provide intelligent feedback based on a programmer's profile, the need to extend the tool to more languages could be detrimental.

The solutions are still assumed to be valid, this means usually an auxiliary tool such as mooshak or an online programming context platform could be useful. Solutions with errors or warnings or not even fulfilling the challenge proposed could still be run with the PP but due to the mechanism of comparing different results, they could deviate the other solutions being analysed.

Furthermore, the tool requires multiple solutions to a problem in order to be effective. This is not necessarily an issue in education as in a classroom several students would solve the same exercises. However, this means that for self-learning, exercises would have to be pre-loaded with solutions.

All in all, the tool is still in a beta phase, and does not have a standalone version. A user, in this case a professor, must run it locally in order to begin profiling. This could have been done through a new platform where professors would create classes and submit solutions to challenges on them, while potentially analysing growth of the students across a certain period of time.

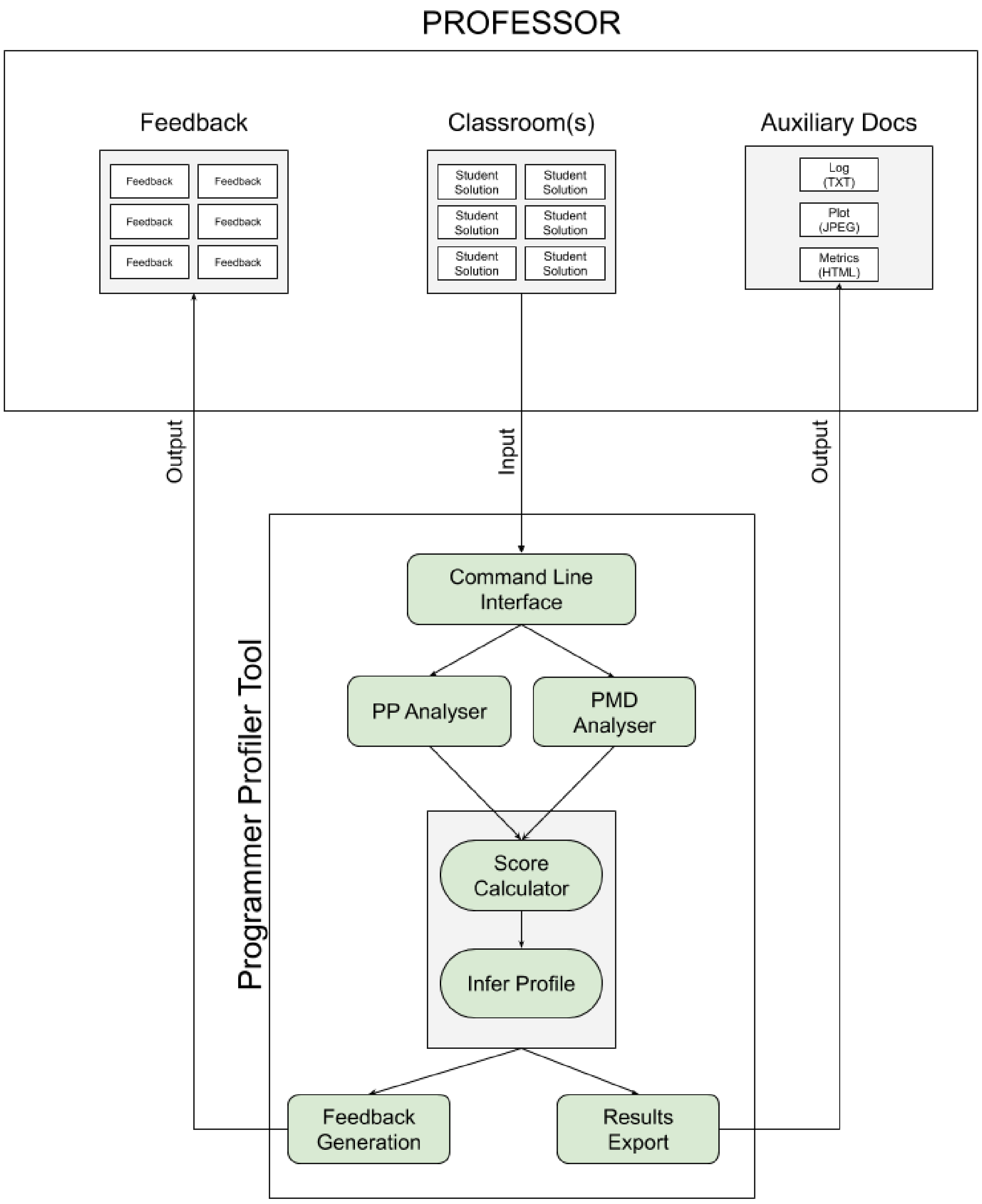


Figure 2.: PP System Architecture

4

SCALING UP THE SYSTEM

4.1 ANALYSING HUNDREDS OF EXERCISES

In order to ensure the Programer Profiler tool was ready to be used in a more generic environment, it was needed to test it with a far more diverse input of exercises. As such, instead of requesting more exercises from a classroom, platforms which provided hundreds of challenges and solutions were explored. In that search, online programming exercise platforms came up as an ideal solution. These type of platforms have several years worth of exercise solutions from all experience levels and with users across the globe. Other services are often either tailored for specific use cases such *Stack Overflow* with just code bits. There are also less filtered contexts such as Open Source projects like found in *Github* where there is great difficulty in comparing solutions for profiling.

By request CodeChef, a not-for-profit educational initiative, supplied compilable Java solutions for 11 exercises. These 11 exercises are have different difficulty levels and as explained above, the diversity of the code to be analysed is quite wide, with distinct coding styles and clearly different experience levels.

The list is as follow:

BEGINNER 300 solutions for HS08TEST, 300 solutions for START01.

EASY 300 solutions for TEST, 300 solutions for JOHNY.

MEDIUM 300 solutions for COINS, 300 solutions for TREEROOT.

HARD 134 solutions for ORDERS, 31 solutions for DOMSOL.

CHALLENGER 300 solutions for FACTORIZ.

LIVE CONTESTS 300 solutions for HOLES, 300 solutions for DOUBLE.

The challenger exercise does not have a solution in polynomial time, so each of them has been scored through a point system. This means it is quite difficult to compare them clearly for profiling as they might have distinct outputs. After all, some are more complex to achieve a higher ranking whilst others get the minimum possible. The exercises from the live contests have an unknown difficulty level, but they also were available during a limited amount of time.

Knowing that the PP tool is focused on the analysis of both skill and readability, it is expected that developers have less concern than usual for readability when applying themselves for programming challenges. That is even more predicted for the live contest exercises. However, due to the way it works, as the profile is obtained in comparison to other solutions, it's the assumption that the programmers are more focused on skill is unlikely to be validated easily.

On trying to analyse almost any of the exercises above the PP tool often got entangled on the comparison of metrics between exercises. The number of differences between each solution could be very wide, which did not happen on the previously available test cases. Furthermore, violations were a far more common occurrence which also pushed scores to the negative.

This meant there was a need to adjust the tool to provide valid and useful results in a far more demanding use case. Through analysis it was also detected that most of the issues occurred due to the presence of outliers, and these were quite common on harder problems, particularly on the Challenger one.

To demonstrate this, one can look at the Factoriz Challenge which is of challenger category, making it easy to identify 2 very contrasting solutions, one of which is an outlier on table 1.

	Average Solution	Outlier
PMD Violations	23	726
# Methods	1	47
# Statements	37	1454
Lines of Code	73	1365
Total Lines	115	1588
# Declarations	26	570

Table 1.: Comparison between a average and outlier solution for Factoriz Challenge

In fact, the deviation compared to the normal solution is minimal, with most solutions ranging from 1-50 PMD violations. On the other hand, the outlier presented 726 violations, each of those currently decreasing the profiler scores, hence why it had negative score. Even though, as this is a challenger category the solutions could be aiming for a very different rating on CodeChef, they are actually quite close as one obtained 266527.5 points while the other 284921.3. Therefore the sharp differences between both solutions are still representative of what may happen in a completely fair comparison of 2 programmers. It's fundamental to know that there's been no pre-filtering of any of the solutions obtained as all of them can be compiled and executed and obtained a valid score.

With that in consideration, it also means that there might be solutions with infinite number of lines, statements, methods or even classes. The goal has been such that the Programer Profiler Tool, after these changes, should be capable of handling any exercise as long as it is a valid, executable java program in order to support the growth and development of each individual.

4.2 ALGORITHM CORRECTIONS

Due to the concerns explained above, several corrections to the profiling calculation algorithms were required. Furthermore, as the tool hadn't been updated for over 2 years, the third party tools it was using were fairly outdated.

So it served as an opportunity to reevaluate some of the existing approaches and update the third party tools that were used in the project.

Currently the PP tool has two major approaches to extract all the relevant information required to compare solutions. The first one is called PP Analyser which is responsible for extracting all metrics from the Java source code. The second is the PMD Analyser which extracts violations to given rules.

As such the corrections have been separated in three major efforts which correspond to the following sections. The PP Metrics rehaul at section 4.2.1, where the metrics obtained are reviewed to encompass the growing differences that were evidenced by the new data set. The PMD update at section 4.2.2 to describe the changes caused by the update to the tool. And finally, the Scoring at section 4.2.3 where the combination of changes to the scoring system are explained.

4.2.1 *PP Metrics adjustments*

The list of metrics for programmer profiling is an integral part of the tool. They are, after all, the only way to get a positive impact on both skill and readability scores, as violations only punish mistakes. However, their impact was quickly shown to be inconsistent when being applied on such a diverse data set.

For example, one can look at a few of the readability metrics specifically *LOC* (Lines of Code), *%LOC* (Percentage of Lines of Code) and their counterparts of comment lines and blank lines. In total, these are 6 metrics which attributed readability in pairs, as one analysed the percentage in the source code, whilst the other the number of occurrences. For instance, considering an exercise A with about 30 lines of source code and 70 blank lines, and an exercise B with 300 lines and 700 blank ones. These 2 exercises have the same percentage of lines of code and lines of comment but, as exercise B has the most lines overall, it will get benefited on the 2 metrics that count occurrences instead of percentage. Meaning that solutions with more lines have a tendency to have higher readability score on not just one metric but three.

Indeed, there were metrics that seemed to overlap, so to reduce the unbalance and streamline the comparisons some were removed. Of the 6 mentioned above, only the number of lines of code, the percentage of blank lines, and the percentage comment lines were kept. If the percentage of lines of code is higher, that means the percentage of blank lines and comment lines is smaller, hence the user doesn't get as high score. After all, if the code is longer, there is ought to be more separation and documentation. However, the weight was also slightly increased to make up for the impact.

Through looking at Listing 4.1 it is possible to see the configuration used for metrics. Here, all five increase skill on a direct proportion as that is shown by the priority label of the json file. Furthermore, there was no metric that directly decreased skill. This meant that the longer and more varied a solution was, the more credits it got in both skill and, as there are other metrics, readability increased due to longer and more detailed code too as explained above. This had a strong impact in the evaluation of the outlier described in Table 1 as it was inferred to be of a Master profile, easily obtaining high scores though the board for simply having an over the top complicated solution which actually lacked both readability and skill.

```
[
  {
    "methodname": "getNumberOfClasses",
    "_this"     : "+" ,
    "implies" : "+S",
    "priority": 2
  },
  {
    "methodname": "getNumberOfMethods",
    "_this"     : "+",
    "implies" : "+S",
    "priority": 2
  },
  {
    "methodname": "getCFSVariety",
    "_this"     : "+" ,
    "implies" : "+S",
    "priority": 4
  },
  {
    "methodname": "getNumberOfNSCCFS",
    "_this"     : "+" ,
    "implies" : "+S",
    "priority": 6
  },
  {
    "methodname": "getDifferentTypesOfNSCO",
    "_this"    : "+" ,
    "implies" : "+S",
    "priority": 5
  },
  {
    "methodname": "getTotalNumberOfTypes",
    "_this"    : "+" ,
    "implies" : "+S",
    "priority": 4
  }
]
```

Listing 4.1: Metric configuration of the ones which attribute skill

Whilst the variety of control flow statements provides information on the language knowledge of the programmer, the number of classes and methods is mostly a readability based analysis. Again, it's important to not underestimate that the possible number of classes and methods is limitless, which means an outlier with hundreds of each could massively affect all others.

A good example of the issues introduced by the current system is to look at an exercise with 1 method and 10 statements versus one with 20 methods and 200 statements. In this situation the former obtained less skill score than the latter. As such methods concerning number of classes and methods now only affect readability instead of both factors. Furthermore, the weights have been updated, with a smaller number of statements having the greatest impact on skill. A programmer's expertise should be measured by the number of statements he used, not by the number of methods nor lines of code. However, does still have an impact in readability.

The priority label was also replaced by weight (which is the term used above), as it had the same naming as PMD violations' but the opposite impact (higher number means more impact).

What came to be was the list represented at 4.2, weight was also adjusted to make up for the removal of some metrics and to balance both skill and readability.

```
[
  {
    "methodname": "getNumberOfClasses",
    "_this"    : "+" ,
    "implies" : "+R",
    "weight": 4
  },
  {
    "methodname": "getNumberOfMethods",
    "_this"     : "+",
    "implies" : "+R",
    "weight": 3
  },
  {
    "methodname": "getLinesOfCode",
    "_this"     : "+" ,
    "implies" : "+R",
    "weight": 3
  },
  {
    "methodname": "getPerComment",
    "_this"     : "+" ,
    "implies" : "+R",
    "weight": 4
  },
  {
    "methodname": "getPerEmpty",
    "_this"     : "+" ,
    "implies" : "+R",
    "weight": 2
  },
  {
    "methodname": "getTotalNumberOfCFS",
    "_this"     : "-" ,
    "implies" : "+S",
    "weight": 5
  },
  {
    "methodname": "getTotalNumberOfCFS",
    "_this"     : "+" ,
    "implies" : "+R",
    "weight": 3
  },
  {
    "methodname": "getCFSVariety",
    "_this"     : "+" ,
    "implies" : "+S",
    "weight": 3
  },
  {
    "methodname": "getNumberOfNSCCFS",
    "_this"     : "+" ,
    "implies" : "+S",
    "weight": 5
  },
  {
    "methodname": "getDifferentTypesOfNSCO",
    "_this"     : "+" ,
    "implies" : "+S",
    "weight": 3
  },
  {
    "methodname": "getNumberOfStatementsWithoutRES",
    "_this"     : "-",
    "implies" : "+S",
    "weight": 5
  },
  {
    "methodname": "getTotalNumberOfDeclarations",
    "_this"     : "+" ,
    "implies" : "+R",
    "weight": 3
  },
  {
    "methodname": "getTotalNumberOfTypes",
    "_this"     : "+" ,
    "implies" : "+S",
    "weight": 2
  },
  {
    "methodname": "getNumberOfRES",
    "_this"     : "+" ,
    "implies" : "+R",
    "weight": 2
  }
]
```

Listing 4.2: Metrics after adjustment

4.2.2 *PMD Update*

PMD, one of the third party libraries used, was at version '5.4.1.' and during this revision it was updated to version '6.10.0'. This introduced several new violations that increased the capabilities of the system as well as a full overhaul of the groups it had available.

This major update also made a new caching option available and, by applying it directly through the command line tool, the speed at which hundreds of exercises were analysed was speed up from a few minutes to a couple seconds. This supported development greatly as minor adjustments to the algorithm could be analysed almost instantly.

The rule groups went from:

UNUSED CODE The Unused Code Ruleset contains a collection of rules that find unused code

OPTIMIZATION These rules deal with different optimizations that generally apply to performance best practices

BASIC The Basic Ruleset contains a collection of good practices which everyone should follow

DESIGN The Design Ruleset contains a collection of rules that find questionable designs

CODE SIZE The Code Size Ruleset contains a collection of rules that find code size related problems

NAMING The Naming Ruleset contains a collection of rules about names - too long, too short, and so forth

BRACES The Braces Ruleset contains a collection of braces rules

During (Novais, 2016)'s progress the decision was to assign the impact of each violation rule individually instead of by these groups. However, this also brought the disadvantage of being unable to easily upgrade the system as with each new violation added, new changes to the tool itself had to be done.

The new version has made a list of groups available which don't match at all with the previous one. Instead, they have revised entirely how the rules are organised bringing in, for example, some rules from unused code and optimisation to two entirely different groups. The new version has made the following available:

BEST PRACTICES Rules which enforce generally accepted best practices.

CODE STYLE Rules which enforce a specific coding style.

DESIGN Rules that help you discover design issues.

DOCUMENTATION Rules that are related to code documentation.

ERROR PRONE Rules to detect constructs that are either broken, extremely confusing or prone to runtime errors.

MULTITHREADING Rules that flag issues when dealing with multiple threads of execution.

PERFORMANCE Rules that flag suboptimal code.

SECURITY Rules that flag potential security flaws.

$$\sum_{i=1}^{m}(I_i * P_i * R_i) - v$$

Figure 3.: Scoring before changes

By having a quick look at the new groups it seemed almost natural to match them directly with our current two factors: skill and readability. For example, security is a topic that clearly has nothing to do with readability, the opposite could be said regarding code style and documentation. As such, to current those issues and to simplify upgrades to the PMD tool the following match has been done:

Both - Best Practices

Readability - Code Style, Design, Dcumentation

Skill - Error Prone, Multithreading, Performance, Security

4.2.3 *Scoring*

Scoring in (Novais, 2016)'s tool was done as seen in formula 3 where:

- I is the impact of the metric, either positive or negative.
- W is the weight
- R is the ratio in comparison with the results from peers. It can be below 0 if there is a deviation from average results.
- m is the number of metrics analysed
- v is the number of types of violations detected

The previous version of the tool did not take into account neither the violation rules' priority nor the occurrence. As such the number of violations was directly subtracted from the obtained score of all metrics. Once again, this happens for both readability and skill.

As this new version is now comparing hundreds of solutions, it's even more important to be able to clearly distinguish each of them. Thus, it was decided to take into account both of these factors. After all, if two solutions were able to get the same score but one had more occurrences of critical violations then the tool should move their skill and/or readability score points accordingly.

Previously, each rule that was violated on a given solution reduced its score by 1 and there weren't usually more than five violations. However, now it must be considered that there might be an infinite number of violations and that the skill or readability score may

$$\sum_{i=1}^{m}(W_i * R_i) - \frac{\sum_{i=1}^{v} W_i * O_i}{2 * maxV}$$

Figure 4.: Scoring after proposed changes

be smaller than that number. These are, after all, some of the problems that arose with the new data set. So, in order to be able to consider such a wide range of violations, it was decided that the impact of violations would be proportional to the score instead. Which means, the solution with the worst violation score for skill would get a negative impact of 50% on its skill scored obtained in the PP Analysis, all others would get a proportional impact based on the number of violations. Same would happen for readability violations.

On the other hand, the ratio which was an essential part of the formula for calculating the score for each metric, as part of the PP Analysis, usually ranged from 0 to 1. However, when the metric had an inversely proportional impact, the ratio could be below -10, which caused the outlier to have reductions of over 100 in score. This was replaced by keeping the ratio between 0 and 1 and the comparison for that ratio to be done against the highest or lowest value of that metric accordingly, instead of using the base solution.

So what has been implemented now is a multi-step process which is described as follow:

1. For each violation rule verified
   a) Priority is taken into account for that particular rule, low priority violations count for '0.25' whilst the highest '1'.
   b) Value is multiplied by the number of occurrences.
2. A sum of the violation score for that solution is obtained.
3. The score is divided by the max violation score for that type (skill/readability) and multiplied by '0.5' to obtain a ratio between '0' and '0.5' .
4. The skill/readability score is multiplied by ('1' - '0.5') of the previous value. Effectively reducing up to 50% if it had the highest violation score for that type.

This can be summed up in the formula 4. Where the following were added:

- O the number of occurrences of that particular violation rule
- $maxV$ the maximum violation score obtained from all peers

4.3 RESULT ANALYSIS

4.3.1 *Scale up*

Overall, these were some of the adjustments implemented in order to scale the tool for more scenarios, namely improvement feedback on online platforms. In order to analyse and compare the differences through this process the medium difficulty exercise called Bytelandian gold coins was used. And the following solutions will be compared: solution A, solution B, solution C.

Due to the tool at the point that was prepared by Novais (2016) crashing with large sized exercises, the comparison of the results will still consider some adjustments from the original tool. This includes the upgrade of PMD, which introduces new violations types, as well as minor adjustments on the calculation algorithms, mostly the one related to the ratio which could drive exercises to get massively negative scores.

Figure 5 represents the distribution with all 300 solutions. It's clearly visible that almost all solutions are profiled as "Experts". With the average skill being higher than the average readability, which seems consistent with the programming challenges environment expectations. However, the distribution is also very tight with several points practically on top of one another. Solution A was profiled as Proficient while B and C were as Experts.

On the figure 6 the graph shows the final distribution after all the changes explained on the previous subsections. Now most of the profiles are considered Advanced Beginner S. There is still a larger influence on the skill score, but the distribution is slightly more spread about. Solution A was profiled as Advanced Beginner S (almost Expert), solution B as Advanced Beginner R while C as Proficient.

To summarise the results of some of these changes table 2 can be viewed. Only some of the key metrics have been listed. Solution A had been profiled as "Proficient", this is a profile leaning towards more readability than skill, however it has:

- The least number of skill penalties;
- The smallest number of statements;
- Far less total lines, almost a 1 to 10 factor compared to solution B;
- Just 2 methods and 1 class;
- Quite a few readability penalties and no comment lines.

By looking at these factors it's obvious the solution A leans towards skill instead of readability. In fact, by making a direct contrast to solution C, it is clear why they almost swapped profiles. Solution C leans towards readability while keeping a good skill score, some of the factors for comparison with solution A:

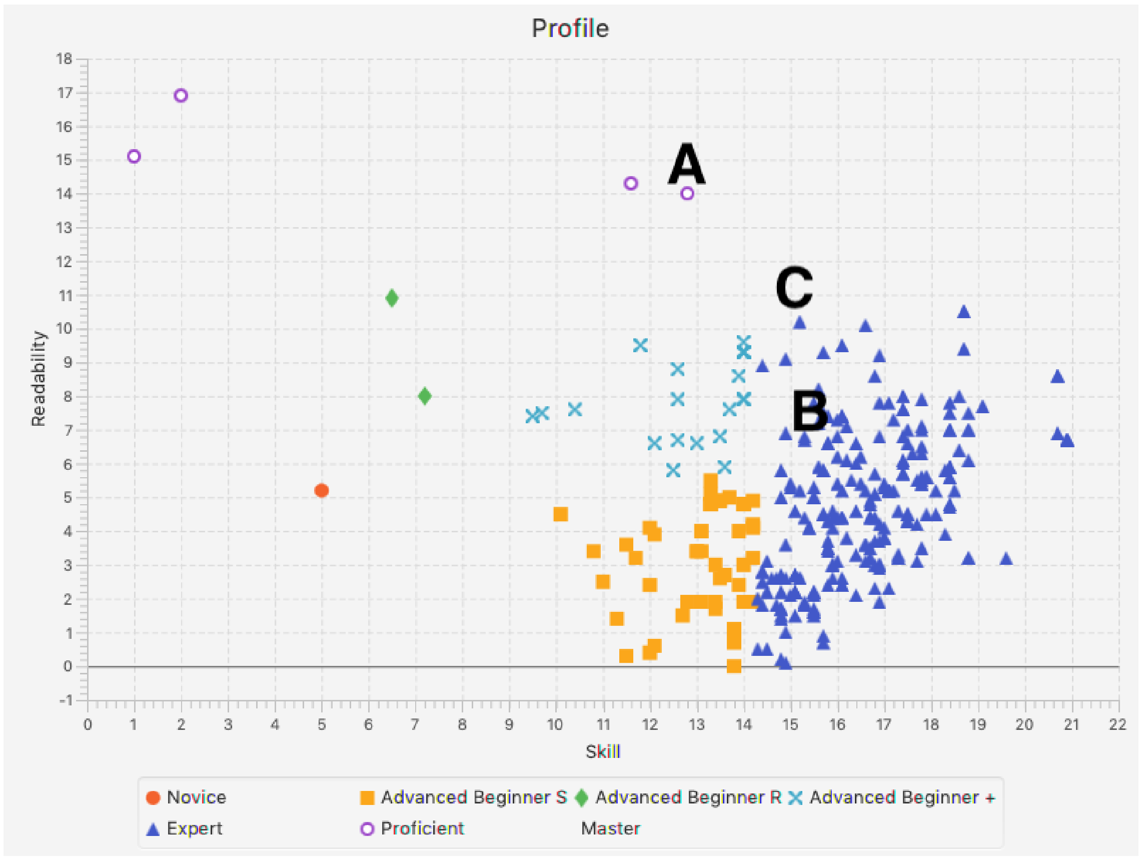


Figure 5.: Distribution of solutions without metrics and full violation changes with A,B and C evidenced

- One skill penalty;
- Three more classes;
- Four times the number of lines of code and of statements;
- 2.7 percent of lines of comment;
- Just 2 methods and 1 class;
- Quite a few readability penalties.

Finally, solution B clearly has a too long solution compared to the others, with the most penalties and no good points in its favour. This, makes it lean more towards readability, hence the profile given is ”Advanced Beginner R”.

Finally, and just from a programmer's direct point of view, there are some things that are easily noticeable and also serve as a validation of the adjustments made.

Solution C is clearly the most readable, it has good descriptions, spacing, more classes and methods. Solution A, was able to solve the exercise in simply 25 lines of code, and

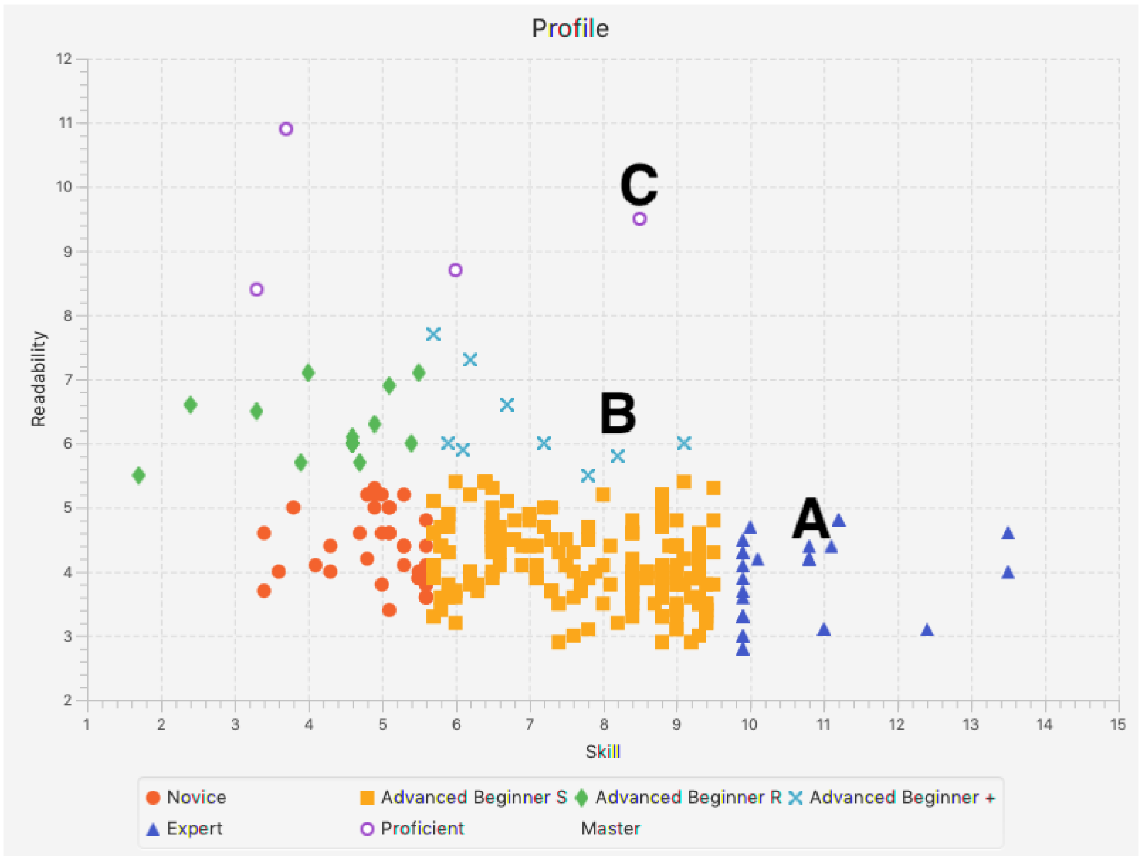


Figure 6.: Final distribution of solutions with A,B and C evidenced

one of the smallest number of statements. On the other hand, Solution B is very long, it's actually complicated compared to other alternatives, it seems more the work of a beginner.

4.3.2 *Control Group*

To further assure the consistency of the adjusted tool a comparison with the combination of the classroom results which were used on (Novais, 2016) is made. In order to see the differences in results, Table 3 can be analysed. Although it shows that 38 out of 73 programmer profiles have been changed, it is significant to note that all but 1 transitioned from adjacent profiles. Adjacent profiles are those which through a small difference in profile, caused moving in the readability or skill axis just enough to change to one of the closer profiles. For instance, the profile Advanced Beginner + which has a balanced mix of readability and skill is in the center of the grid, as such can move to any other profile. However, an advanced more tailored for readability like Proficient or Advanced Beginner R can't shift directly to an Advanced Beginner S or Expert.

	Solution A	Solution B	Solution C
Skill PMD Penalty	0	1	1
Readability PMD Penalty	7	14	8
# Classes	1	2	3
# Methods	2	18	6
# Statements	4	60	17
Lines of Code	13	99	52
Percentage of Comment	0	2.3%	2.7%
Total Lines	26	214	73
# Declarations	4	16	10
Profile - Before	Proficient	Expert	Expert
Profile - After	Advanced Beginner S	Advanced Beginner R	Proficient

Table 2.: Comparison 3 solutions before and after the PP Tool scaling adjustments

Out of the 73 profiles, there were just 2 (Std26, Std31) which moved to non-adjacent profiles. The one which presented the biggest change was student 31 who suffered strong penalties due to PMD violations causing his profile to be of a novice on exercise 1 and 4. In fact, it's precisely due to the violations that it's become harder to reach the higher levels like proficient, expert and master. Yet, now all the master students have become Experts.

The changes to the profiles can be easily noticed also in the following description:

- Novice: 3 $\rightarrow$ now 2
- Advanced Beginner (Readability): 5 $\rightarrow$ now 9
- Advanced Beginner (Skill): 14 $\rightarrow$ now 22
- Advanced Beginner (Both): 20 $\rightarrow$ now 22
- Proficient: 17 $\rightarrow$ now 7
- Expert: 6 $\rightarrow$ now 2
- Master: 0 $\rightarrow$ now 1

Although it's clearly an important change in profiles obtained, it's still important to take into account the number of changes that were performed. Furthermore, considering that in 97.26% of the situations the new profiles were just a shift to one of the adjacent ones it's clear that the impact was minimal.

Std Profiles	Before Scale	After Scaling
P	P	P
D	AB+	E
V	E	E

Z	E	E
A	N	N
M	AB+	AB+
F	AB-S	AB-S
Std2	N	AB-S
Std3	P	AB+
Std4	P	AB-R
Std5	AB-S	AB-S
Std6	AB-R	AB+
Std8	N	AB-R
Std9	P	P
Std10	AB+	AB-S
Std11	AB-S	AB-S
Std12	AB-S	AB-S
Std13	AB+	AB-R
Std14	P	AB+
Std15	AB+	AB-S
Std16	P	AB+
Std17	AB-R	AB+
Std18	AB-S	AB-S
Std19	AB+	AB+
Std20	E	AB-S
Std21	AB+	AB+
Std22	AB-S	AB-S
Std23	E	AB-S
Std24	P	AB+
Std25	AB+	AB+
Std26	P	AB-S
Std27	AB+	AB+
Std28	E	AB-S
Std29	AB-S	AB-S
Std30	P	AB+
Std31	E	N
Std32	AB+	AB-S
Std33	P	P
Std34	P	AB+
Std36	P	P
Std38	AB+	AB-R

Std39	AB+	AB-R
Std40	AB+	AB+
Std41	AB-S	AB-S
Std42	P	AB-R
Std43	AB+	AB+
Std44	AB+	E
Std45	AB+	AB+
Std46	P	AB-R
Std47	AB-S	AB-S
Std48	AB-S	AB-S
Std49	AB+	P
Std50	AB+	AB+
Std51	E	AB-S
Std52	P	AB+
Std53	N	AB+
Std54	AB+	AB-S
Std55	AB+	AB+
Std56	AB+	M
Std57	AB-R	AB-R
Std58	P	P
Std59	AB-S	AB-S
Std60	AB-S	AB-S
Std61	AB+	AB+
Std62	E	AB-S
Std63	AB-S	E
Std64	AB-S	AB+
Std65	P	P
Std66	AB-R	AB+
Std67	P	P
Std68	AB-R	AB-R
Std69	AB-S	N

Table 3.: Comparing profiles after the adjustments for 4 classroom exercises analysed in (Novais, 2016)

5 PROVIDING FEEDBACK

The main component explored in this work is how to provide feedback to a programmer based on his source code. Traditionally feedback systems report all errors prior to compilation which can be overwhelming to a programmer. Also, in (Orrell, 2006) participants ”suggested that one of the most valuable purposes of assessment was to give students feedback on their achievements.”. It was also analysed that academics were focusing more on summative assessment instead of formative as it was more expedient despite producing poor learning habits in students due to them being more concerned with passing instead of learning.

Thus, the concept of the programmer's profile can be leveraged as a form of achievement while guiding them towards a balance between skill and readability. This would also by definition be taking into account the context of his solution in relation to the peers. After all, a programmer with 2 years experience can be considered expert if his peers are just starting to learn how to code, but he would likely be quite a novice compared to those with 10 years experience.

As such this has been tackled in a series of steps which will be explained throughout this chapter, ranging from the selection of feedback, through leveraging the profile but also touching motivation and implementation.

5.1 DELIVERING FEEDBACK

As noted back in section 2.3 there are different types of feedback where KM is the most common. In (Keuning et al., 2016) it is also concluded that unless it's a test-based system, the capability to adapt to teachers' requirements is quite limited, requiring code changes and recompilation of the tool.

The goal has been to allow the PP tool to still remain very adaptable and easily brought to new contexts while requiring no manual intervention. This would allow a professor to apply it to whichever exercises he finds more relevant and to be more demanding if the students in one group were more skilled than other groups. This means only KM and KH recommendations can be applied.

Our score formula which is shown in 4 uses *PMD* to attribute a penalty based on violating rules, this is very akin to a test based system which enables KM feedback. By knowing violations it is possible to provide information to the user to fix that particular rule. At the same time, feeding from metrics like statements used or percentage of comments to assign points to score and readability, which could be used to generate KH feedback. Unlike with violations there's no clear definition of 'pass' and 'failure', instead it is possible to generate information based on the context such as: "On average, users were able to solve this exercise with half the statements that you used, perhaps there's a easier solution?".

The PP requires a group of solutions for the same exercise and a base solution from which it is possible to derive comparisons. It was already reporting all rules that were violated per solution and generating a chart of the profile distribution along with a log of all information made. None of the elements were meant for a single contributor but instead for the person running the tool, potentially a teacher or someone in a company evaluating candidates. As the feedback is meant to allow a programmer to improve, it is necessary to generate this information with that user in mind. This could either reach him directly through an online tool or be forwarded by someone like a professor after being slightly mediated. In both scenarios a file for each programmer is created which he can read directly.

One of the research work that was followed in order to deliver feedback is (Nicol and Macfarlane-Dick, 2006) in which seven principles are proposed to facilitate self-regulation. Examples of how to apply this concept are shown at the end of this chapter. Furthermore, only one or two suggestions are generated for each programmer. The perceived advantages of this limitation are: ensure novices (and similar profiles) are not overwhelmed by a large checklist of improvements; motivate students to improve step by step instead of knowing what to expect; allow self-assessment instead of a fully guided tutorial of improvement; focus on improvement to an attribute at a time, considering multiple changes simultaneously might have an unexpected impact on the score, making the motivation section less accurate; avoid too many tips become public unnecessarily so as not to be easily exploited, example if one finds that the tool suggests to improve documentation, students might start trying to write gibbering documentation lines.

Therefore, the structure adopted for each feedback file was the following:

INTRODUCTION TO PP TOOL Explaining the purpose of the tool and the number of peers.

PROFILE AND SCORE ANALYSIS Inform of the profile obtained in which solution along with the score in both skill and readability.

RECOMMENDATION Suggest a way to improve that has the most impact with minimal effort.

MOTIVATION Apply basic 'gamification' to persuade the programmer to follow the recommendation and achieve a better and balanced profile.

5.2 LEVERAGING PROFILE

As discussed on (Pahl and Kenny, 2008), an Intelligent Tutoring, the "challenge in computer-aided learning is to achieve personalisation in a learning experience based on skills training and activity". In our tool it has been managed to adapt the feedback system to each programmer in multiple ways. Two of which, the information provided by the assigned profiles was used.

First, a selection is done on whether to focus on improving **skill** or **readability** by looking at where the profile is placed. This can be easily understood at image 7. A programmer whose strength is on readability will obtain feedback focused on skill and vice-versa as this is the kind of feedback that will have the most long-term impact on his growth. The balanced profiles will instead get the tip which has the most immediate effect on their score as they don't need to focus on any attribute. In both situations the ideal profile is "Master", a balanced profile which excels in both skill and readability.

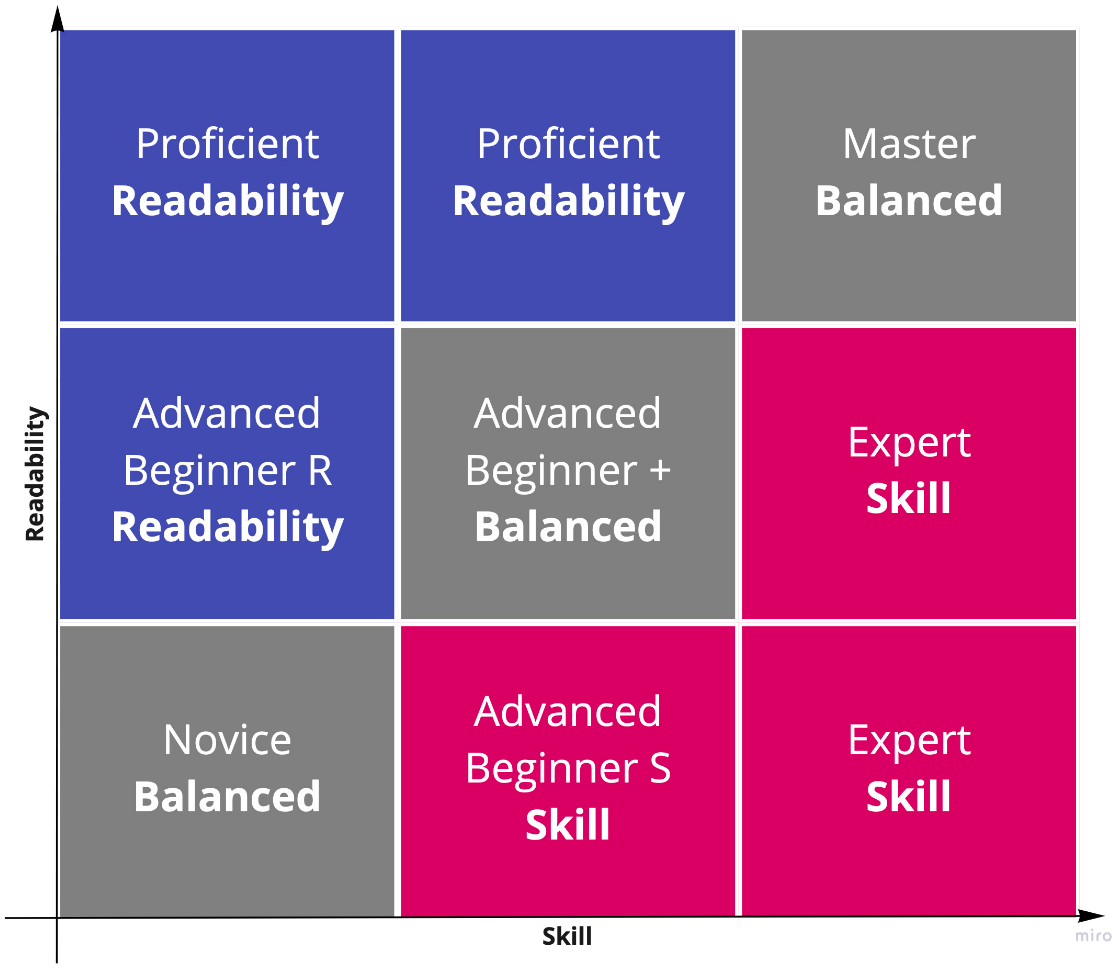


Figure 7.: Mapping profile strength: skill, readability or balanced

Secondly,the profiles are used as an achievement system with implied gamification. This has been explored further on several research projects such as (Swacha and Baszuro, 2013). As our feedback will be mostly focused on improving one attribute at a time, the usual path for achievement can be seen through the arrows in figure 8. It is thus intuitive that transitioning from a Proficient or Expert profile to a Master is quite challenging but that is the limitation of the profile distribution as these two stretch across being great in one attribute but potentially terrible in the other one. How this information as been represented will be discussed in section 5.4 which focuses on motivating the programmer for improvement.

As such the profiles allow us to personalise both the feedback obtained and the way it is communicated. This information is particularly valuable when considering that during the scaling up efforts over 300 solutions were used for each exercise. Furthermore these large input is composed of very different approaches that got distinct evaluations.

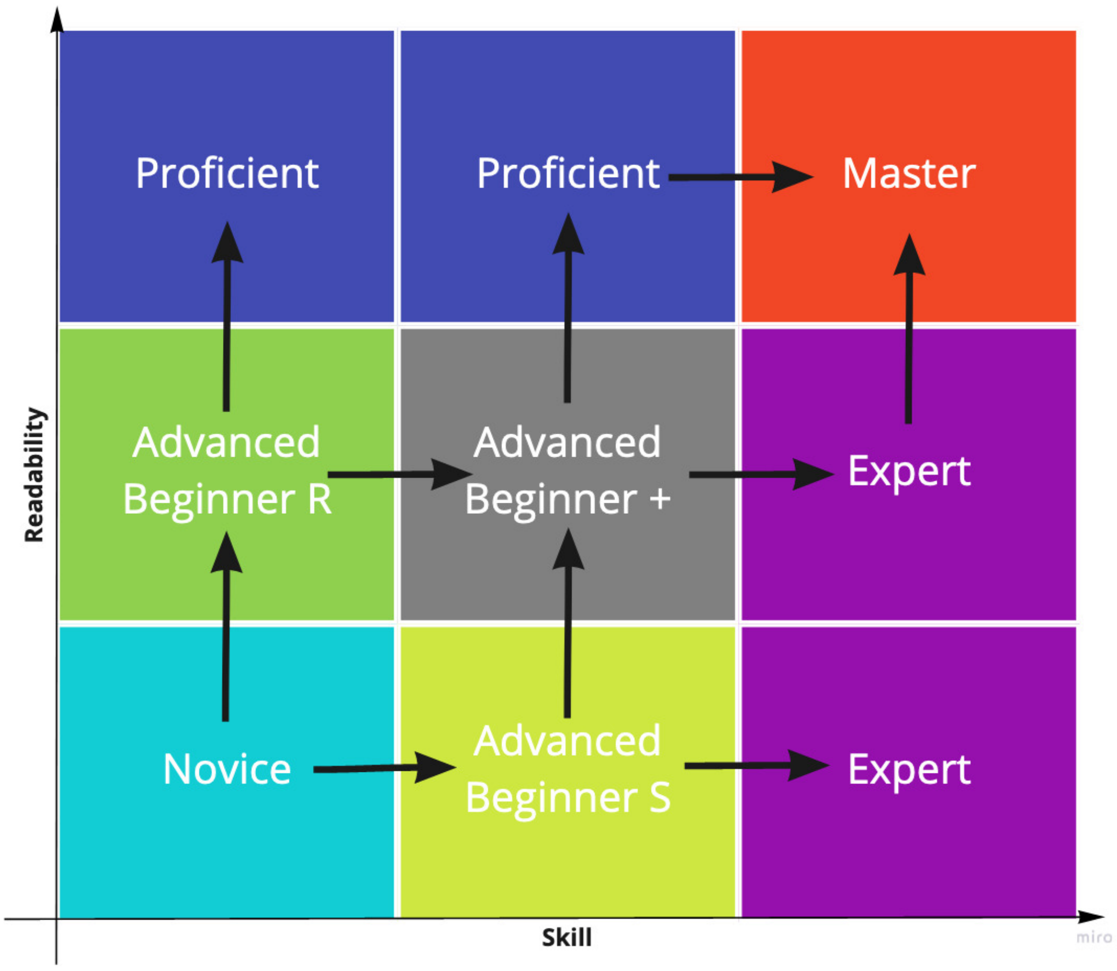


Figure 8.: Growth paths for profiles after feedback adjustments

5.3 TYPES OF FEEDBACK

After knowing for each programmer whether the focus will be on improving skill or readability, the next step is to select either violation type of recommendations, subsection 5.3.1, or metrics recommendations, subsection 5.3.2.

As each Java rule provided through PMD has already an embed description complete with examples and is just about a pass or fail scenario it is far easier to solve than, for instance, to fully document the code or find a better algorithm which allows less use of statements. For this reason, if there are any violations to a rule, that is the feedback applied first. Only afterwards feedback of metrics that extracted from the source code is delivered.

5.3.1 *Violations*

In difficult exercises it is easy to provoke dozens of violations which are caught by the PMD tool. Back in section 4.1, on the challenger exercise, a solution which had over 700 violations was explored. So to solve all of these simultaneously would be too cumbersome, instead the focus will be on the one that had the most negative impact on the score. This means the weight of the rule along with the number of occurrences is calculated. By fixing just this one violation type he will get, for minimum effort, the most positive personal growth.

One example of how this feedback is delivered can be seen in listing 5.1. The information structure is as follows:

1. Type of suggestion, in this case follow PMD rule
2. The rule violated with a direct link to the PMD documentation which provides examples.
3. Number of occurrences this rule was violated in the solution.
4. Basic information of the set and description
5. Lines of the solution where it was violated
6. Extra tip to work towards

```
### Suggestion: **Follow PMD Rule**
You violated rule [ControlStatementBraces](https://pmd.github.io/pmd-6.16.0/
    pmd_rules_java_codestyle.html#ControlStatementBraces) **2** times.
This rule is part of the set Code Style

**Description**: This statement should have braces
You have violated it in the following lines of the project:
```

```
+ 14
+ 16

#### Goal
You might still have some violations to improve on, but we advise you to more
    than just that. Check out:
**There seems to be a better way to solve this exercise using fewer statements
    . Perhaps you over complicated the algorithm?.**
```

Listing 5.1: Example of Violation Feedback in raw Markdown

Even if the programmer corrects only half the lines where the rule was violated, the next time he runs the program he most likely will get another suggestion as the previous one no longer holds such impact on his overall score. This type of feedback is the most common to be obtained but since there are dozens of different rules and they are customisable, it still allows the programmer to feel that he is progressing with small steps and hopefully not make the same mistakes again.

However, as violation-type of feedback can often have a low impact for improvement and be quite easy to fix, there is also a secondary goal based on the most impactfull metric. The possible recommendation are simplified versions of what will be introduced in the following section.

5.3.2 *Metrics*

Due to the scale up, adjustments to the metrics that were used for score calculation were done. The current metrics list can be traced back in listing 4.2. There it is evidenced that the methods which calculate the score in skill and readability are quite different. This means that depending on which attribute needs to be improved, the metrics available to provide feedback on have to be adapted.

Furthermore, it was noticed that if some of the metrics used were known it would make it easy to circumvent the score mechanism of the system. One scenario where was identified is if a suggestion about improving the readability of a solution needed to be provided by way of increasing the percentage of blank lines. Here, a programmer could simply add hundreds of random blank lines in the middle of his code which would serve no actual purpose for readability. Same thing for metrics about lines of code. Due to this, only some of the metrics were chosen as possibilities in feedback while still keeping the other ones as part of the score calculation mechanism.

For readability the available metrics' methods to provide feedback from are:

GETNUMBEROFMETHODS Obtains the number of methods that the algorithm uses. This metric by default assign points to readability as to keep everything in one single method can be cumbersome.

GETNUMBEROFCLASSES Similar to the one above this analyses the number of classes.

GETPERCOMMENT A percentage of the lines of comment existing in the file.

Even though the metrics were selected based on which would be hardest to be misleading in, from the moment that they are known, it's still straightforward how to do it. This is particularly true for the percentage of comments, as a programmer could just add almost blank lines tagged as comments. The issue seems to be more inherent to readability as this is information that is meant for humans and as such even though it could not be used in the algorithm or be gibberish it wouldn't be caught by our system. Which means that with the current readability feedback, after reusing the system a few times it would be possible to understand how to easily achieve a good readability score. This happens at least for the metrics that are available (number of methods, classes and percentage of comments). It would be necessary to fully update how the evaluation of readability occurs by adding for example natural language processing as well as checking the calls of methods and classes to reduce the associated risk. Nevertheless, that is the reason why only some of the metrics are used for feedback. Furthermore, some readability improvements might have a negative effect on skill. For instance, by adding new methods and classes, a programmer would have more statements, and so his skill score could potentially be significantly reduced.

The metrics used for skill are the following:

GETNUMBEROFSTATEMENTSWITHOUTRES Number of statements disregarding relevant expressions such as System calls like *System.out*.

GETTOTALNUMBEROFCFS Number of Control Flow Statements

```
### Suggestion: **Decrease Control Flow Statements (CFS)**
You have shown the use of **4.0** control flow statements. You seem to have
    used **78.95**% more flows than your best peer.

Control Flow Statements can be considered the heart of algorithms. However,
    overusing them can cause a bad performance as well as might show there is
    a easier way to solve the problem. By decreasing the number of CFS you
    might be able to simplify your solution.
Currently you only obtained **2.5** points to your readability score from this
     metric.

**Try using 2 CFS instead.**
```

Listing 5.2: Example of Metric Feedback in raw Markdown

Unlike what occurs in readability metrics, the skill ones are naturally less prone to misleading. Even when it is public that *Control Flow Statements (CFS)* is one of our ways to evaluate skill, in order to find a way to solve the problem using not only less CFS but also less overall statements, would require actual programmer skill. It might involve using a completely different approach or particular language tools that allow them to achieve a better score. Regardless only some metrics are available because either their the others' impact is low or they are very hard to improve on. For example, one way a programmer can be scored on skill is by the number of *Not So Common Control Flow Statements (NSCCFS)*, unlike CFS which are core to an algorithm, a good solution doesn't need NSCCFS. As such, it was on purpose that a feedback like rejected. After all trying to solve exercises with more NSCCFS could be just complicating a solution, even though he could get a slightly higher skill score.

The metric feedback is very distinct from the violations one and an example can be seen at listing 5.2. it is structured in the following way:

1. Type of suggestion, in this case decrease CFS
2. The number of solution's CFS
3. A relative comparison with the best peer, it underlines that there is a far better way to solve the exercise
4. A brief explanation of the relevance of this metric.
5. Current obtained score from this metric
6. A suggestion of what could be the next achievement, it is dependant on a growth factor of how different it is from the best solution

5.4 Impact of Change

As mentioned in multiple articles ranging from e-learning to general learning experiences gamification can be used with high return to boost chances for improvements. This has been discussed for example in (Swacha and Baszuro, 2013), (Kapp, 2012) and (Muntean, 2011). The same concept was introduced to the PP Tool by adding a new section right after providing the suggestion for improvement. As can be seen in listing 5.3 the first step is to describe what attribute the programmer would be improving if he followed the suggestion plus the direct impact it has on his score both by relative and absolute value.

The mechanism of scoring provides this platform for constant improvement where a programmer can keep seeing both his readability and skill improvement as he follows the feedback one by one. Also, as shown back in figure 8 the profiles can be leveraged as an achievement system. The ultimate goal is to reach the profile of master, hence why the last part of the motivation section is on what is the impact that the improvement has on the user's profile. In the shown example, he would go from a 'Advanced Beginner S' to a 'Advanced Beginner +' which is a more balanced profile which requires improvement of readability..

```
### Impact of Change

By following the recommendation above your score will fundamentally improve on
**readability**.

- **Change in Readability**: 6.01 -> 5.7336845
(**-4.819168.

- **Change in Skill**: 5.38 -> 7.5675
(**28.90651.

#### Profile after Recommendation
After this change you'll reach an even better profile: **Advanced Beginner
    +**.

Congratulations on obtaining a balanced profile! However, you certainly have a
     lot to improve before reaching a better profile.
Perhaps, it will be easier to focus first on either readability or skill,
    arriving at Proficient or Expert respectively before aiming for Master.
    Good luck!
```

Listing 5.3: Section of motivation

5.5 IMPLEMENTATION

The feedback mechanism was implemented as a new class named 'Feedback' which aggregates all previously obtained information. For that, other data structures were slightly adjusted to provide more easily accessible information such as the impact of each violation on the score which were only needed once before.

First, the header is generated with the introduction to the tool. Afterwards, for each file, the attribute on which to focus on is presented. The first step is to balance profiles that are leaning to a specific attribute as seen in figure 7. If, however, the profile is already balanced, such as with a 'Novice', then the attribute in which the violation impact was greater is selected. This can be seen in listing 5.4.

```
// Skill Leaning Profiles
if (profile.equals("Advanced Beginner S") || profile.equals("Expert")) {
    isReadability = true;
    violationFixed = provideViolationTip(project, isReadability);

// Readability Leaning Profiles
} else if (profile.equals("Advanced Beginner R") || profile.equals("Proficient
    ")) {
    isReadability = false;
    violationFixed = provideViolationTip(project, isReadability);

// Balanced Profiles
} else if (profile.equals("Novice") || profile.equals("Advanced Beginner +")
    || profile.equals("Master")) {
    isReadability = project.getReadabilityViolationImpact() >= project.
        getSkillViolationImpact();
    violationFixed = provideViolationTip(project, isReadability);
}
```

Listing 5.4: Choosing attribute to improve

With this information, the selection is on whether to provide a rule improvement tip or a metric related one. By default the former has priority unless there was nothing reported by the PMD for that programmer. In both of the scenarios the greatest impact to the programmer's attribute score is the basis for selection.

Through these queries it is identified which metric or violation to provide feedback on. There are in total 6 different methods that can be called depending if it's one of the 3 readability tips, 2 skill tips or a general obtained violation tip. One simplified example of the feedback classes is at listing 5.5.

```
private float provideClassesTip(Project project, MetricImpact metric) {
    ArrayList<String> metricFeedback = new ArrayList<>();
    float improvementImpact, improvementRatio;

    metricFeedback.add("### Suggestion: **Increase number of classes**");
    metricFeedback.add("You have used **" + metric.getValue() + "** classes. "
        +
            "This places you at **" + Math.round(10000 * metric.getRatio()) /
                (float) 100 + "% ** compared to the maximum of your peers.");
    metricFeedback.add("");
    metricFeedback.add("Dividing your code into multiple classes, if relevant,
        can dramatically increase its readability. ");
    metricFeedback.add("Currently you only obtained **" + metric.getImpact()
        + "** points to your readability score from this metric.");
    metricFeedback.add("");
```

Listing 5.5: Implementation of feedback regarding number of classes

Finally, the motivation section is generated which is the same regardless of the tip obtained. By using the score difference of the chosen tip, it's impact to the current score and profile is provided. Precisely this section is divided in the possible update on the programmer's score and in the possible profile achievement.

If there is no change in profile despite following the tip the messages are motivation based. For instance, an *Expert* profile would get "Keep improving your readability above anything else in order to notice the biggest changes more quickly." while a *Master* as he can't get already a better profile receives a achievement "Congratulations! You have reached the best profile! You can of course improve either skill or readability depending on the context, but usually that will force you to lose the balance."

On the other hand, when there could be improvements to the profile all the messages are accomplishment based such as for a *Advanced Beginner S* who would receive "Congratulations! You are no longer a Novice. However you certainly have a lot to improve, "Proficient", "Expert" and "Master" are all better profiles than the one you have currently achieved. However you are clearly on the way forward. Just follow the recommendation, try to notice other areas to improve, and rerun! You now lean more towards skill.". A *Master* for instance would simply get "Congratulations! You have reached the best profile!".

All the feedback files are generated using Markdown[1] as it is a plain text format that is easy-to-read, easy-to-write and converts easily to structurally valid XHTML.

5.6 GUIDELINES FOR FEEDBACK

The aim has been to generate individual and personalised feedback to each student. Two complete examples of the final result can be seen in B.

In the following subsections the implementation of each guideline for feedback proposed by (Nicol and Macfarlane-Dick, 2006) will be discussed.

5.6.1 *Helps clarify what good performance is*

It is important for a programmer to know what is considered good performance, hence understanding some of the existing metrics such as the number of statements and documentation along with the profile definition is one of the steps towards clarifying good performance. However, as presented on the article this is not quite enough as written feedback is often misinterpreted.

1 https://daringfireball.net/projects/markdown/

5.6.2 *Facilitates the development of self-assessment (reflection) in learning*

The tool makes explicit the current stage the programmer is in along with where he is aiming towards. The article mentions that teachers should create opportunities for self-monitoring and judging of progression to goals, this is precisely what being able to run their own code against the tool allows them to do. Although, the ability to match their own self-assessment with the score they obtained from the tool would have to be done manually. After all, the PP process does not consider any self-evaluation moment as it was meant to be used from the point of view of the professor.

5.6.3 *Delivers high quality information to students about their learning*

One of the proposed strategies for this is to limit the amount of feedback so that it is actually used. This has been precisely one of the adopted approaches, as only one violation at a time along with a brief description of an improvement to a general metric is generated. Another strategy is to provide corrective feedback which is the tip itself as mentioned above. Finally making sure that feedback is provided in relation to pre-defined criteria, this is an advantage of an automated tool.

5.6.4 *Encourages teacher and peer dialogue around learning*

Due to the nature of the tool, it is meant to be used in an async way. This means that there is very limited that could be done regarding this principle However, it is up to the teacher or the student to the information generated to apply this methodology.

5.6.5 *Encourages positive motivational beliefs and self-esteem*

In section 5.4 the topic of motivating the student for improvement is explored, which is precisely what this principle refers to. In fact, the third strategy proposed in the article is automated testing with feedback. The overall goal was to allow them to see that even if the score is very low in small steps it can be greatly improved.

5.6.6 *Provides opportunities to close the gap between current and desired performance*

The second part of the change in profile portion of the individual feedback entirely related to closing the gap towards the next profile. By aiming to achieve a balanced, both skillful and readable format, the programmer can obtains the steps required to improve and the change in profile caused by those changes. Although, according to (Nicol and Macfarlane-Dick, 2006) there could be a lot more done to make this effective.

5.6.7 *Provides information to teachers that can be used to help shape the teaching*

Although the individual feedback to programmers is not necessarily helpful to shape teaching, the efforts done in the first version of PP Tool (Novais, 2016) can provide very valuable input. This has now become even more reliable due to the scaling up discussed in chapter 4.1. The teacher can see how his class is quickly compared to other's by looking at the generated charts shown in figure 5. That would provide knowledge on outliers both negative and positive as well as if there is any leniency towards either attribute.

6

RESULTS

In order to analyse the results of this project, in this chapter a few case studies are explored. With these, the goal is to identify where the tool is falling short and where it is excelling. The results are made clear in the conclusion.

6.1 EXERCISES FROM PROFILING

During the development of the initial PP in (Novais, 2016), participants from an object oriented programming class were the basis for the validation of the programmer profiling. In the mix, master students in Computer Science and the professor's solutions were also compared to ensure the increased experience was being recognised.

As this project is a reshape of the tool for use in formal education system, the analysis of the provided feedback to the same use case is a relevant form of validation. For that reason, a specific exercise was explored like the one in listing A.1 on the solutions made by: two undergraduate students (50 and 58); one graduate student (Z) and the professor himself (P). Previously their profiles for this specific exercise had been Proficient, Advanced Beginner +, Expert and Expert respectively. Now, due to the changes referred in section 4.1, student 58 became a novice and the Professor a Master, whilst the others' profile did not change. This distribution is also made evident in figure 9.

The solution and generated feedback of the four selected individuals can be seen in appendix A. And through this section each of the results will be analysed, starting with the professor's.

6.1.1 *Professor*

The professor's solution, listing A.2, is the base for comparison to all others. As such, it needn't get feedback for improvement in the normal use case of the tool. However, since it can execute profiling even for the base solution, so does the feedback file get generated. With this, it is possible to validate whether the feedback is useful for more advanced users.

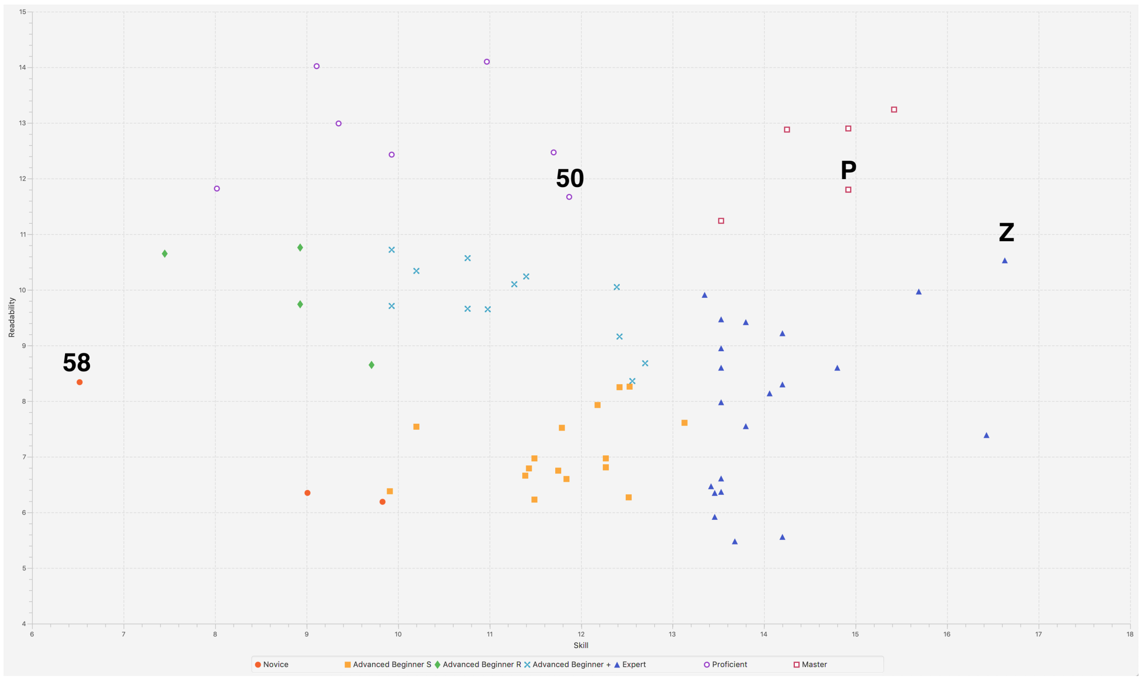


Figure 9.: Profile comparison of selected users for exercise P2

Despite having a profile of Master it doesn't mean it cannot improve further. In fact, it still incurred in 4 violations on PMD, so by fixing those the score could improve substantially. This also proves that it doesn't require a perfect solution to obtain the best profile.

The professor scored high in almost all metrics ranging from not so common control flow statements (due to do while) to overall code readability (due to comments and blank lines). And, has he had obtained a balanced profile, choosing feedback was solely based on the impact. For that reason, the obtained file issued two recommendations. First, to follow java class naming conventions, and secondly to add more documentation. The secondary suggestions are provided when the first one is related to PMD, for it is usually very easy to fix unlike the metric related ones.

For the former suggestion, it was found in line 3 and a description is provided obtained directly from the PMD. The Professor used an underscore in the name of the class "ExIdades_Teste", breaking the Pascal case which is considered the standard in Java according to the the default PMD rules. Furthermore, this is a violation that is tagged as high priority hence why, out of the 4 possible ones, it was the first to be delivered to the user. More information can be seen in the PMD website just like the link was printed in the file - Class Naming Conventions. It would be possible to either disable or configure this rule by loading a different set into the PP. As previously mentioned, currently the *quickstart* set is being used. By following the suggestion, the professor could improve his readability score by almost 8%, and this would be achieved within a couple minutes.

For the latter recommendation, the professor used 2 lines of comments. This is far less than the highest result, which meant instead of 4 readability points, only 1.3 were assigned. However, explicitly improving on this regard doesn't necessarily mean the exercise is better documented. After all, this is simply a direct look into the lines of comment and not at their usefulness.

As the profile is already "Master" there is no profile improvement change. It is also worth noting that since the metrics are constructed in a way that sometimes they affect skill and readability simultaneously but in opposite ways, a user might get to the point where he is is, for instance, reducing number of statements only to increase them on the next iteration. This scenario has not been dealt with and yet has not happened in the data sets used.

6.1.2 *Student 50*

The student 50, listing A.5, also showed 4 violations. In spite of this, for the sake of considering a scenario where there are no violations to pick on first, those have been disregarded. This student's profile leans towards readability as he obtained "Proficient", as such the suggestion will focus on improving skill. It can also be seen that on line 5, he used a comment line, this contributed significantly to his readability score as most users did not use any. However, it was simply a boilerplate but the PP does not validate whether the comment lines actually contribute to readability or if they are gibberish.

For this student, the only suggestion which was generated is to decrease the number of statements used. This can frequently be the hardest to follow as sometimes it requires figuring another way to approach the problem. The feedback file also points out that about one third of the students managed to use less statements and that by simply removing one, the student would climb to a better profile. Even more so, the improved profile would be master which is the best and most balanced.

As this recommendation is based on the metrics, it is not possible to provide directly a line or area where he could remove one statement. Still, it is known that other solutions managed to do it with fewer statements.

By comparing directly this solution to the professor's, it is evident that some things could be improved. The while cycle for instance could be turned into a for to improve readability. Some other improvements might not be caught by the PP right now, however by looking at the score it is still falling short of the professor's. Nevertheless, the intention is that after following the available suggestions after, perhaps, multiple iterations the solution will be both more readable and more skilful.

To reduce number of statements could potentially have implications in the other scores as it might require redoing the algorithm. By doing so, other metrics' scoring would also change, such as the percentage of lines with comments, or the number of statements.

6.1.3 *Student 58*

Student 58, listing A.4, had the lowest skill and low readability. As such he obtained the lowest profile of Novice. From incurring on 8 violations to bottom scores on almost all the metrics, there was a lot to pick on what to improve.

Yet, he obtained the same initial recommendation as the Professor, for being a high priority violation. And, by correcting it, he would be able to achieve a better profile already. This happens both because the rule has a large impact due to the priority but also because he was already quite close to the other profile. Even so, the information provided on the profile after recommendation section is highly adapted to motivate the novice users to improve further, unlike what happened with the Professor.

The other recommendation is also related to documentation. This happens the student needs to improve readability the most, so even the secondary goal aims tu find ways to do so. After all, the student had only used 1 line of code which was also a boilerplate.

By attempting to configure manually to provide a secondary skill solution, the user would get a request to reduce number of statements. By looking at his solution, it does seem to be the right way to go. The student hard coded the number array, and seems to have used many unneeded steps compared to other solutions. However, it seems a very different approach so it might not be the most relevant comparison.

6.1.4 *Master Student Z*

The master student has several years of Java experience compared to student 50 and 58. His code is shown on listing A.5, and like the Professor, it has 4 violations. He followed standard Java Conventions and the tip was a recommendation that has medium to low priority. However, it was violated 3 times hence why the impact to his score would enable him to achieve the Master profile as well.

Despite the fact that his code is clearly more concise than the others' compared here, the secondary recommendation is to split the code into multiple methods. That goes to show that even though his skill score is the highest out of all 70 analysed in this exercise, since the algorithm recognises he needs to improve readability, the proposal is to make it clearer. However, this would clearly reduce the skill score as the number of statements would increase.

The overall primary feedback obtained for these 4 solutions can be seen in figure 10. On it, it can be seen that only profile 50 was recommended higher skill feedback. It was also the only way leaning towards readability. All the others obtained readability first initial feedback.

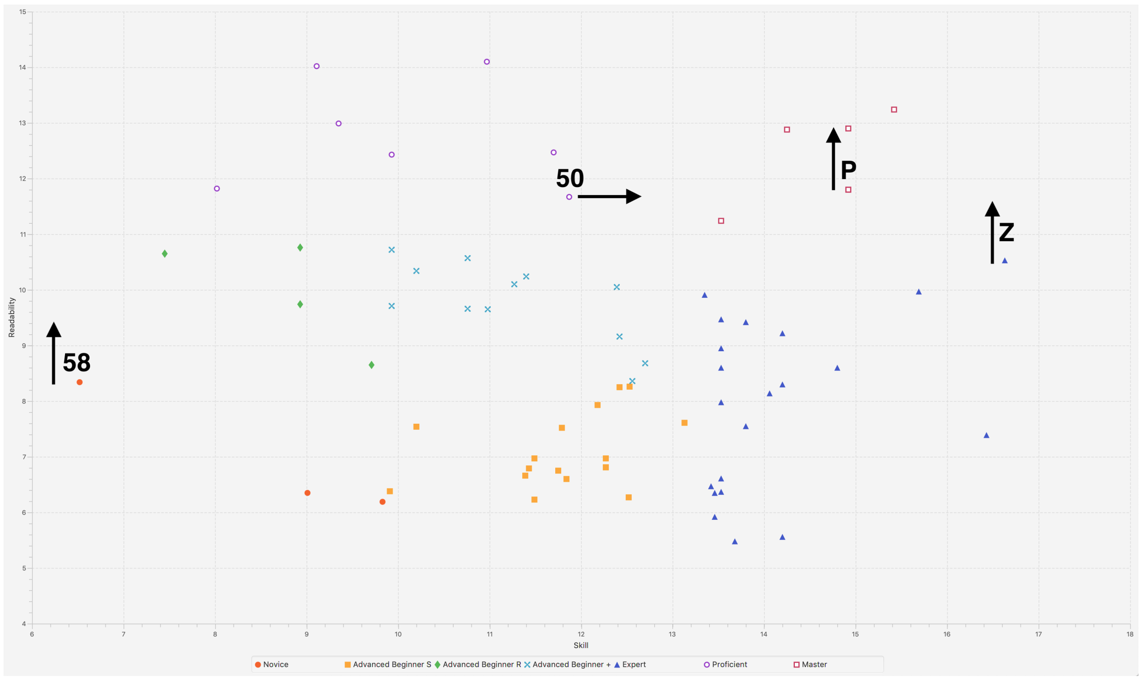


Figure 10.: Focus on improvement for users in exercise p2

6.2 REUSING THE SCALE UP GROUP

In section 4.1 3 solutions are used as a control group of the changes done during this work. Now these will be used again to compare the feedback obtained. The table 2 will also be used as basis for this section.

The generated feedback for solution A, B and C can be found in appendix B. To start with, lets consider what they have in common.

- All had 7 or more violations;
- Of these 3, only one would be able to achieve a better profile by simply following the primary recommendation;
- None would get more than 1 full point in either attribute;

By looking at other solutions analysed in this exercise this is a clear pattern in all. It seems to be the case when there's more competition. Unlike the previous example of exercise 2, this is an analysis of 300 people. Since the PP works by comparing all of these and profiling them, the more competition there is, the harder it is to get to move to another profile. Furthermore, as the exercise itself is more difficult, there are bound to be more violations. This is combined with the context in question which is a code challenge website, where readability and best practices are least taken into account, which provokes more PMD violations.

On the other hand, all the recommendations obtained are different between the 3 exercises. Either focusing on skill for a Advanced Beginner S profile like Solution A, or on readability for a Proficient one like Solution C. The only feedback in common is between Solution B and C where the secondary goal is to reduce the number of statements used, since they have dozens of lines of code unlike solution A.

6.3 PROGRAMMER FEEDBACK

Finally and most importantly, 4 new programmers were asked to test the tool and provide feedback on the results obtained. These are: STV, MNDS, HG and RSND. STV and RSND are university students, whilst MDNS and HG are not. However the last two haven't been programming with Java in years so can be considered with about the same experience as the others. The solutions are distributed as follows:

- **CodeChef Coins**: STV1, STV2, HG and MNDS;
- **CodeChef Johny**: RSND;

The solutions and generated feedback can also be found in appendix B for the coins exercise and in appendix C for the Johny one. Only HG is not considered valid on CodeChef for not handling input properly, this is due to being the least experienced in programming challenges. However for the purpose of this analysis that was disregarded.

6.3.1 *STV*

Student STV has the freshest experience to Java. He has been using it for a couple years and every so often has to touch it again.

He started by solving the problem through solution 2, listing B.7. This enabled him to achieve "Expert" profile, with a very significant difference between skill and readability. Hence, he obtained recommendation to fix one of his two violations regarding readability.

The student disagreed with the primary suggestion as in his opinion, for this purpose, creating an instance or an utility class is a waste. He felt it would be useful though if it had been in a longer project instead of a programming challenge. This is understandable considering there's a conflict between requirements on a platform like CodeChef versus the normal projects a programmer has to deal on a daily basis. Furthermore, he felt there was an added motivation from the feedback and agreed with the secondary feedback of being able to improve documentation. The format of the feedback also seemed to be clear as there was no question as to what was expected from it.

The student also decided to improve his solution before seeing the feedback from the PP. So he proposed solution 1, listing B.6 which uses dynamic programming. The feedback obtained from the tool was the same although profiling and scoring wise he was punished in skill. He obtained a Advanced Beginner S, dropping almost 40% in skill score. Although still a skill leaning profile, it was a massive drop. This is, though, expected given the limitations of the algorithms used. Since there is no consideration for performance, both proposals solve the same problem. Yet one (the dynamic programming) is more complex than the other one, both in number statements and in lines of code. As such even its readability is more difficult. In fact, solution 1 manages to solve the exercise in 0.08 seconds while solution 2 does it in 6.04. The student, after understanding that the performance was not considered, noted that he'd only propose the more readable and simpler version which is the one with the highest score. Even so, given that the programmer executing both solutions got the same violations, generating similar feedback is on point. After all, the goal is to improve the programmer and not necessarily the specific solution.

6.3.2 *HG and MNDS*

Both of these programmers hadn't touched Java in several years. To solve this exercise was the first time that they were remembering the structure and the language. Considering that the tool evaluates the language proficiency it is natural that if they had taken more time their score would be even higher.

On one hand, Programmer HG agreed completely that there was a lack of documentation, which he hadn't added because his goal was just to get the code to work. His profile was after all Expert, and his solution quite similar to STV2.

On the other hand, MNDS applied OOP concepts by fully using the class and objects. Due to that, he had the highest readability of those analysed, earning him the profile of Advanced Beginner +. The suggestion was to reduce control flow statements, and if that was done, perhaps he could get quite close to Master. This also shows the issue with the profiles edge's being so thin, as one can directly jump from Advanced Beginner + directly to Master.

None of these 3 programmers with 4 solutions could however reach a new profile but as previously mentioned, this might be due to the context of competitiveness in this exercise.

6.3.3 *RSND*

On a different exercise, which is of difficulty Easy, RSND obtained 6 violations. His profile of Advanced Beginner S could be improved to Advanced Beginner + if he followed the provided recommendation. This also proved that since the exercise was easier the competitiveness was lower which meant there were less steps needed to move to new profiles.

The primary suggestion obtained was due to not following naming conventions on 3 separate occasions. He understood why the profile was poor as he had first solved the challenge in c++ and so migrated with different conventions. He also would have liked to get more than just one violation, which could be useful for more advanced or experienced programmers. After all RSND was very curious as what he needed to do to get a better score, which showed that the gamification aspects could be working.

The secondary suggestion also seemed reasonable once again since the intention was to improve readability and he had no comment lines and had just focused on delivering a valid solution.

7 CONCLUSION

In the beginning of this work it was established that it was possible to deliver personalised feedback for improvement of a programmer profile based on his source code. The project developed was done by adding functionality on top of the PP by (Novais, 2016) and demonstrates that hypothesis. The tool is available publicly as an open source project on Github[1]. It contains exercise solutions from CodeChef, which can also be found on their website.

7.1 PROJECT DEVELOPMENT STAGES

In order to ensure quality of the final result, this project followed a series of phases which are hereby described:

- Search of the basic bibliography and its revision;
- Search for data sets to validate opportunities;
- Scale up the tool for the provided data sets;
- Ensure quality of the scale up through testing in different contexts;
- Implement the feedback generation system based on the profiles;
- Evaluate, test and draw conclusions towards the outcomes from this evaluation and the studies done along the project.

7.2 OUTCOMES

By building on top of the PP, it is now possible to submit source code of java solutions to any programming exercise and get personalised feedback for improvement. It was proved through demonstration that even for over 300 programmers it is possible to provide suggestions for improvement that have a large impact on scoring.

1 https://github.com/martinhoaragao/ProgrammerProfiler

As such, Professors can apply such a tool in their evaluation routine in order to support programmer's growth without needing manual intervention for configuration. By comparing the solutions between themselves the tool is able to adapt to any challenge or project and help avoid the problems of focusing on summative assessment only. The combined output of the previously done plots, logs and personalised feedback enables teaching hubs to be more efficient.

Furthermore, the application of gamification and motivation attributes greatly supports the users to improve one step at a time. By deliver the most impactful tips towards the least developed attribute, the programmers are then directed towards a complete and balanced profile which enables to be both skilful and write readable code.

Even so, there is still space for larger testing and fine tuning that could impede this beta tool from going into the day to day of teaching environments yet.

As part of this work there was a paper submitted (Aragão et al., 2019) that was published in the 2019 edition of SLATE (Symposium on Languages, Applications and Technologies). The article covered Chapters 1, 2, 3 and 4 of this dissertation.

7.3 FUTURE WORK

Although there are multiple Intelligent Tutoring Systems in the market, most tend to be made for a specific teaching course with limited availability for configuration. Hence inferring feedback simply from source code and personalising it from the profile and comparisons is a new approach. Furthermore, since this has been extended from a previous work, a lot of what applied to it in terms of work in progress has not been touched during the development of this project so it still stands true.

7.3.1 *Supporting Projects*

So far, all the testing has been achieved with small classroom exercises or coding challenges. However, the moments when the concerns that originated this work are more valid, namely the ratio of professors to students, are in projects that involve one or more programmers and are complex assignments. Since frequently, this kind of tasks are accomplished in group, there would be a need to somehow apply the tool from versioning data. By selecting for example entire methods that were submitted through Git, source code analysis could be run. However, in that case, there would be a limited amount of comparison data.

Nevertheless, without supporting projects, even the amount of possible feedback provided is limited. After all, a 50 lines challenge does not allow to apply for example OOP concepts fully.

7.3.2 *Graphical User Interface for self-learning*

Initially there was the intention of building a GUI, however that plan could not be accomplished. By for example making the tool available directly online with an available data set for some exercises, it would mean that two sides experience could be improved: professors and students outside the educational system.

In fact, with the growing content of online education, it seems intuitive to see personalised feedback with gamification features being made available to self-learning individuals. Taking into account that the entire process is automatic and without needing human intervention, it is an easy fit.

At the same time, one of the discovered notes from bibliography is that it is hard for professors to actually run a system like so. Even though the files are generated automatically there is no easy way to send them to students or to process them in bulk or one by one. Through the development of an interface, more functionality could be made easy for professors to enable them to both drive decisions from the data and to improve the generated feedback with some human adjustments.

7.3.3 *Customisation*

Currently there are a couple files that enable customisation, from the metrics file to the PMD setup. However, it is not straightforward how to do it. Even the metrics' one lack some documentation and it is limited by which metrics have been built into the tool anyway. The PMD rule setup for instance requires change in a class and has no customisation outside finding where the tool is loaded.

By picking on customisation as a whole topic, templates could be created for instance, which would allow teachers to choose which kind of setup suits them best. Documentation should also be improved, and as much as possible all relevant customisation should be available outside having to recompile the tool. This was another of the key flaws of most ITS overall. Unfortunately as the focus was on ensuring the viability of the approach, this kind of topics were not a priority.

7.3.4 *Improve Feedback and Personalisation*

Despite some changes in messages between the different profiles, the available personalisation could be enhanced. For instance, perhaps Novice programmers don't need a secondary suggestion. Or even, some violations are too complicated for them to understand while on the other hand Experts and Proficients would rather get the entire list instead of seeing one at a time. This was one of the topics pointed out by the programmers who validated the current feedback.

The motivation section is also mostly a prediction, as following some metrics' suggestions could have unexpected results across the other methods for analysis. Or even, it could be validated that there is a profile with better skill and readability that has a better score in that metric. That would ensure that delivering that feedback is correct as there is a way to improve that metric without losing too much on another. Adjustments could be made to visually show this possible impact.

Currently there are only two types of feedback: violations and metrics' improvement. The rules information comes directly from PMD, and so is limited by what they provide as descriptions to the rules. However, a lot more could be made available, from pointing out books to read to improve readability or directly a chapter related to it. Perhaps a talk, or showing a snippet of a better solution. Also, it was noticed that current readability related feedback like separating classes and methods is not very reliable as that sort of decision should come from problem understanding and object separation instead of simply identifying solutions with more or less classes and methods. In fact, the mechanism of comparing solutions to the highest scorer in each metric and generating feedback towards it might be incorrect. After all, even though a solution is valid using 10 classes, doesn't mean it's the right way to go. Most likely filter solutions by higher profiles or to the base solution would be a better way of approaching this challenge. For example, only provide feedback to increase documentation if either the base solution or a Master solution exists with higher documentation.

Furthermore Metric' related to not so common operators could link to examples of how to use them. There is an endless possibility of improved feedback that could be made personalised. For example, it could be possible to assign data to a programmer and check how often he has gotten certain types of feedback, and make mini-games to ensure he learnt them. Similar concepts are currently being used on dictionaries such as the one in Kindle.

7.4 END NOTE

With the ambition that this work has been able to shed some light into the world of automatic teaching methods, the dissertation is concluded. Hopefully it was a good read on the concept of personalising feedback based on programmer's profiling through source code analysis.

BIBLIOGRAPHY

Martinho Aragão, Maria João Pereira, and Pedro Henriques. Scaling up a programmers' profile tool. In *SLATe 2019–Symposium on Languages, Applications and Technologies*. OASIcs: OpenAccess Series in Informatics, 2019.

Tyne Crow, Andrew Luxton-Reilly, and Burkhard Wuensche. Intelligent tutoring systems for programming education: a systematic review. In *Proceedings of the 20th Australasian Computing Education Conference*, pages 53–62. ACM, 2018.

Charlie Daly and John Waldron. Assessing the assessment of programming ability. *SIGCSE Bull.*, 36(1):210–213, March 2004. ISSN 0097-8418. doi: 10.1145/1028174.971375. URL http://doi.acm.org/10.1145/1028174.971375.

Ethan Fast, Daniel Steffee, Lucy Wang, Joel R. Brandt, and Michael S. Bernstein. Emergent, crowd-scale programming practice in the ide. In *Proceedings of the 32Nd Annual ACM Conference on Human Factors in Computing Systems*, CHI '14, pages 2491–2500, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2473-1. doi: 10.1145/2556288.2556998. URL http://doi.acm.org/10.1145/2556288.2556998.

T Flowers, Curtis Carver, and J Jackson. Empowering students and building confidence in novice programmers through gauntlet. pages T3H/10 – T3H/13 Vol. 1, 11 2004.

Markus Fuchs and Christian Wolff. Improving programming education through gameful, formative feedback. *2016 IEEE Global Engineering Education Conference (EDUCON)*, pages 860–867, 2016.

Johannes Hofmeister, Jennifer Bauer, Janet Siegmund, Sven Apel, and Norman Peitek. Comparing novice and expert eye movements during program comprehension. *FACHBEREICH MATHEMATIK UND INFORMATIK SERIE B INFORMATIK*, 17, 2017.

Dai Hounsell. Student feedback, learning and development. *Higher education and the lifecourse*, pages 67–78, 2003.

Weizhi Huang, Wenkai Mo, Beijun Shen, Yu Yang, and Ning Li. Automatically modeling developer programming ability and interest across software communities. *International Journal of Software Engineering and Knowledge Engineering*, 26(09n10):1493–1510, 2016. doi: 10.1142/S0218194016400143. URL http://www.worldscientific.com/doi/abs/10.1142/S0218194016400143.

Huzefa Kagdi, Michael L Collard, and Jonathan I Maletic. A survey and taxonomy of approaches for mining software repositories in the context of software evolution. *Journal of Software: Evolution and Process*, 19(2):77–131, 2007.

Karl M Kapp. *The gamification of learning and instruction*. Wiley San Francisco, 2012.

Hieke Keuning, Johan Jeuring, and Bastiaan Heeren. Towards a systematic review of automated feedback generation for programming exercises. In *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education*, ITiCSE '16, pages 41–46, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4231-5. doi: 10.1145/2899415.2899422. URL http://doi.acm.org/10.1145/2899415.2899422.

Michael J Lee and Andrew J Ko. Personifying programming tool feedback improves novice programmers' learning. In *Proceedings of the seventh international workshop on Computing education research*, pages 109–116. ACM, 2011.

Richard Lobb and Jenny Harlow. Coderunner: A tool for assessing computer programming skills. *ACM Inroads*, 7(1):47–51, February 2016. ISSN 2153-2184. doi: 10.1145/2810041. URL http://doi.acm.org/10.1145/2810041.

Robert C Martin. *Clean code: a handbook of agile software craftsmanship*. Pearson Education, 2009.

Cristina Ioana Muntean. Raising engagement in e-learning through gamification. In *Proc. 6th International Conference on Virtual Learning ICVL*, volume 1, pages 323–329, 2011.

David J Nicol and Debra Macfarlane-Dick. Formative assessment and self-regulated learning: A model and seven principles of good feedback practice. *Studies in higher education*, 31(2):199–218, 2006.

Daniel Novais, Maria João Pereira, and Pedro Rangel Henriques. Profile detection through source code static analysis. 51:1–13, 2016.

Daniel José Ferreira Novais. Programmer profiling through code analysis. Master's thesis, December 2016.

Janice Orrell. Feedback on learning achievement: rhetoric and reality. *Teaching in Higher Education*, 11(4):441–456, 2006. doi: 10.1080/13562510600874235. URL https://doi.org/10.1080/13562510600874235.

Claus Pahl and Claire Kenny. Personalised correction, feedback, and guidance in an automated tutoring system for skills training. *International Journal of Knowledge and Learning (IJKL)*, 4(1):75–92, 2008.

James H Paterson. Evaluating the readability of example programs for novice programmers. *FACHBEREICH MATHEMATIK UND INFORMATIK SERIE B INFORMATIK*, page 9, 2017.

Emília Pietriková and Sergej Chodarev. Profile-driven source code exploration. *Computer Science and Information Systems (FedCSIS), pp. 929-934, IEEE.*, 2015.

Simone Scalabrino, Gabriele Bavota, Christopher Vendome, Denys Poshyvanyk, Rocco Oliveto, et al. Automatically assessing code understandability. *IEEE Transactions on Software Engineering*, 2019.

Jakub Swacha and Paweł Baszuro. Gamification-based e-learning platform for computer programming education. In *X World Conference on Computers in Education*, pages 122–130, 2013.

A

EXERCISE P2

```
P2) Write a Java program that, given two ages (integers, M and N) reads N ages
and outputs all ages greater than M, and the average (real number) of all ages
.
For example ,
20 //-> M 5 //->N 15
20
21 40 5
given:
The output should be: 21
40
20.2
```

Listing A.1: Exercise P2 provided to students

A.1 PROFESSOR SOLUTION AND FEEDBACK

```
import java.util.Scanner;
public class ExIdades_Teste {

        public static void main(String[] args) {
        Scanner ler = new Scanner(System.in);
        int n, m, idade;
        int total=0;
        float media;

        do {
            System.out.println("Introduza um numero para comparacao");
            m = ler.nextInt();
        } while (m <=0);

        do {
            System.out.println("Introduza o numero de idades");
            n = ler.nextInt();
        } while (n <= 0);

        // Read the age values
        for (int i = 0; i < n; i++) {
            idade = ler.nextInt();

            if (idade > m) {
                System.out.println(idade);
            }

            total=total+idade;
        }

        //The division result must be casted to float!
        media=(float)total/n;
        System.out.println(media);
    }
}
```

Listing A.2: Solution to P2 by Professor

Welcome to the PP Tool Feedback System! You have been compared to 70 other projects which solved the exact same exercise. By analysing metrics and mistakes a profile has been extracted. This is done with 2 key distinctions in mind Skill and Readability

A.1.1 *Profile Analysis*

Your project which is named *p* has achieved a profile of **Master**! The score obtained is **14.92** skill and **11.8** readability.

A.1.2 *Personalised Feedback*

> The system will now provide personalised feedback to help you improve your score. This is done by prioritising the easiness and impact in your current score.

*Suggestion: **Follow PMD Rule***

You violated rule ClassNamingConventions **1** times. This rule is part of the set Code Style
Description: The utility class name 'Ficha8' doesn't match '[A-Z][a-zA-Z0-9]+(Utils?|Helper)'
You have violated it in the following lines of the project:

- 3

GOAL You might still have some violations to improve on, but we advise you to more than just that. Check out: **Add more documentation.**

Impact of Change

By following the recommendation above your score will fundamentally improve on **readability**.

- **Change in Readability**: 11.8 -> 12.8 (**7.8125%**).
- **Change in Skill**: 14.92 -> 14.92 (**0.0%**).

PROFILE AFTER RECOMMENDATION Your profile won't suffer any change yet, but that is alright! Congratulations! You have reached the best profile! You can of course improve either skill or readability depending on the context, but usually that will force you to lose the balance.

A.2 STUDENT Z FEEDBACK AND SOLUTION

```java
import java.util.Scanner;

public class Solution {

    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        int threshold = in.nextInt(), i, ages = i = in.nextInt(), age, sum =
            0;
        while(i > 0){
            sum += age = in.nextInt();
            if(age > threshold)System.out.println(age);
            i--;
        }
        /*Double or float, according to the desired decimal digits precision*/
        System.out.println(ages > 0 ? (double) sum/ages : 0.0);
    }
}
```

Listing A.3: Solution to P2 by a Master Student (Z)

A.2.1 *Profile Analysis*

Your project which is named Z has achieved a profile of **Expert**! The score obtained is **16.63** skill and **10.53** readability.

A.2.2 *Personalised Feedback*

The system will now provide personalised feedback to help you improve your score. This is done by prioritising the easiness and impact in your current score.

*Suggestion: **Follow PMD Rule***

You violated rule GenericsNaming 3 times. This rule is part of the set Code Style

Description: All classes and interfaces must belong to a named package You have violated it in the following lines of the project:

- 3
- 7
- 10

GOAL You might still have some violations to improve on, but we advise you to more than just that. Check out: **Divide your code into multiple methods.**

Impact of Change

By following the recommendation above your score will fundamentally improve on **readability**.

- **Change in Readability**: 10.53 -> 11.28 (**6.6489334**%).
- **Change in Skill**: 16.63 -> 16.63 (**0.0**%).

PROFILE AFTER RECOMMENDATION After this change you'll reach an even better profile: **Master**.

Congratulations! You have reached the best profile!

A.3 STUDENT 58 FEEDBACK AND SOLUTION

```java
//Exercicio 2

import java.util.*;

public class Exercicio2
{

    private static ArrayList<Integer> numeros ;

    public static float GeraList(ArrayList<Integer> nr, int n , int m){

        Integer num [] = {23,34,3,65,20,24,9,7,51,43,60,87,53,37};
        float resultado = 0;
        float media = 0;
        int i=0;

        numeros = new ArrayList<Integer>(Arrays.asList(num));

        Iterator it = numeros.iterator();
        nr = new ArrayList<Integer>();

        while (it.hasNext()){
            int p = (int) it.next();
            if(p > m ){
                nr.add(p);
            }
        }

        Iterator maior = nr.iterator();


        while (maior.hasNext()){
            if (i<n){
                int q = (int) maior.next();
                resultado = resultado + q;
                System.out.println("Numero ARRAY ->" + q);
                i++;
            }else break;
        }

        return media = (float)(resultado/n);
    }
}
```

Listing A.4: Solution to P2 by a Bachelor's Student 58

A.3.1 *Profile Analysis*

Your project which is named *Aluno58* has achieved a profile of **Novice**! The score obtained is **6.52** skill and **8.34** readability.

A.3.2 *Personalised Feedback*

> The system will now provide personalised feedback to help you improve your score. This is done by prioritising the easiness and impact in your current score.

*Suggestion: **Follow PMD Rule***

You violated rule ClassNamingConventions **1** times. This rule is part of the set Code Style
Description: The utility class name 'Ficha8' doesn't match '[A-Z][a-zA-Z0-9]+(Utils?|Helper)'
You have violated it in the following lines of the project:

- 6

GOAL You might still have some violations to improve on, but we advise you to more than just that. Check out: **Add more documentation.**

Impact of Change

By following the recommendation above your score will fundamentally improve on **readability**.

- **Change in Readability**: 8.34 -> 9.34 (**10.706642%**).
- **Change in Skill**: 6.52 -> 6.52 (**0.0%**).

PROFILE AFTER RECOMMENDATION After this change you'll reach an even better profile: **Advanced Beginner R**.

Congratulations! You are no longer a Novice. However you certainly have a lot to improve, "Proficient", "Expert" and "Master" are all better profiles than the one you have currently achieved. However you are clearly on the way forward. Just follow the recommendation, try to notice other areas to improve, and rerun!

You now lean more towards **readability**.

A.4 STUDENT 50 FEEDBACK AND SOLUTION

```
import java.util.Scanner;

public class Exercicio2
{
    // vari veis de inst ncia - substitua o exemplo abaixo pelo seu pr prio
    public static void main(String[] args)
    {
        System.out.println("Insira a idade minima: ");
        Scanner s = new Scanner(System.in);
        int min = s.nextInt();
        System.out.println("Insira quantas idades quer introduzir: ");
        int quantos = s.nextInt();
        System.out.println("Insira as idades: ");
        int idade;
        double sum = 0;
        int i=0;

        while(i < quantos){
            idade = s.nextInt();
            if(idade<0)
                System.out.println("Insira um numero positivo");
            else
            {
                if(idade>min) System.out.println("Idade " + idade + "    maior
                    do que " + min);
                sum += idade;
            }
            i++;
        }

        System.out.println("M dia de todas as idades: " + sum/quantos);


    }
}
```

Listing A.5: Solution to P2 by a Bachelor's student 50

A.4.1 *Profile Analysis*

Your project which is named *Aluno50* has achieved a profile of **Proficient**! The score obtained is **11.87** skill and **11.67** readability.

A.4.2 *Personalised Feedback*

> The system will now provide personalised feedback to help you improve your score. This is done by prioritising the easiness and impact in your current score.

*Suggestion: **Decrease number of statements***

You have used **6.0** statements. This places you at **66.67%** compared to the best of your peers.

A big difference of statements usually means there is a far easier solution to the problem. Currently you only obtained **3.3333335** points to your skill score from this metric.

Try using 5 statements instead.

Impact of Change

By following the recommendation above your score will fundamentally improve on **skill**.

- **Change in Readability**: 11.67 -> 11.67 (**0.0%**).
- **Change in Skill**: 11.87 -> 13.703333 (**13.378738%**).

PROFILE AFTER RECOMMENDATION After this change you'll reach an even better profile: **Master**.

Congratulations! You have reached the best profile!

B EXERCISE CODECHEF COINS

This appendix introduces feedback to an exercise which be found in COINS. All feedback generated was converted to latex from markdown using Pandadoc[1].

B.1 CODECHEF SUBMISSION 17777014

```
/* package codechef; // don't place package name! */

import java.util.*;
import java.lang.*;
import java.io.*;

/* Name of the class has to be "Main" only if the class is public. */
class Codechef
{
    static long ans(long n)
    {
        long num=n/2+n/3+n/4;
        if(num>n)
        return ans(n/2)+ans(n/3)+ans(n/4);
        else
        return n;
    }
  public static void main (String[] args) throws java.lang.Exception
  {
      Scanner sc=new Scanner(System.in);
      while(sc.hasNext())
      {
    long n=sc.nextLong();
    System.out.println(ans(n));
      }
  }
}
```

1 https://pandoc.org

Listing B.1: Submission 17777014 to COINS

Welcome to the PP Tool Feedback System! You have been compared to 300 other projects which solved the exact same exercise. By analysing metrics and mistakes a profile has been extracted. This is done with 2 key distinctions in mind Skill and Readability

B.1.1 *Profile Analysis*

Your project which is named *17777014* has achieved a profile of **Advanced Beginner S**! The score obtained is **8.44** skill and **3.87** readability.

B.1.2 *Personalised Feedback*

> The system will now provide personalised feedback to help you improve your score. This is done by prioritising the easiness and impact in your current score.

*Suggestion: **Follow PMD Rule***

You violated rule GenericsNaming 3 times. This rule is part of the set Code Style

Description: Avoid empty catch blocks You have violated it in the following lines of the project:

- 8
- 14
- 16

GOAL You might still have some violations to improve on, but we advise you to more than just that. Check out: **Add more documentation.**

Impact of Change

By following the recommendation above your score will fundamentally improve on **readability**.

- **Change in Readability**: 3.87 -> 4.62 (**16.233765%**).
- **Change in Skill**: 8.44 -> 8.44 (**0.0%**).

PROFILE AFTER RECOMMENDATION Your profile won't suffer any change yet, but that is alright! Keep improving your readability above anything else in order to notice the biggest changes more quickly.

B.2 SOLUTION 21236236

```
import java.util.HashMap;
import java.util.Map;
import java.util.Scanner;

class ByteLandianCoin {


  static Map<Long,Long> solutions = new HashMap<Long,Long>();
  public static void main(String args[]) {

    Scanner in = null;
    try {
      in = new Scanner(System.in);
      String line =null;

      while(in.hasNext()){
        line =in.nextLine();
        System.out.println(solution(Long.parseLong(line)));
      }
    } catch (Exception e) {
      System.out.println("Error " + e);
    } finally {
      in.close();
    }
  }


  public static long solution(long coin){
    long solution = coin;

    if(coin<12){
      return coin;
    }

    if(solutions.containsKey(coin)){
      return solutions.get(coin);
    }

    long coinSum = solution(Math.floorDiv(coin,2)) +solution(Math.floorDiv(
        coin,3)) +solution(Math.floorDiv(coin,4));
    if(coinSum>coin){
      solution = coinSum;
    }
    solutions.put(coin, solution);
    return solution;
  }



}
```

Listing B.2: Submission 21236236 to COINS

Welcome to the PP Tool Feedback System! You have been compared to 300 other projects which solved the exact same exercise. By analysing metrics and mistakes a profile has been extracted. This is done with 2 key distinctions in mind Skill and Readability

B.2.1 *Profile Analysis*

Your project which is named *21236236* has achieved a profile of **Advanced Beginner R**! The score obtained is **5.38** skill and **6.02** readability.

B.2.2 *Personalised Feedback*

> The system will now provide personalised feedback to help you improve your score. This is done by prioritising the easiness and impact in your current score.

Suggestion: ***Decrease Control Flow Statements (CFS)***

You have shown the use of **4.0** control flow statements. You seem to have used **78.95**% more flows than your best peer.

Control Flow Statements can be considered the heart of algorithms. However, overusing them can cause a bad performance as well as might show there is a easier way to solve the problem. By decreasing the number of CFS you might be able to simplify your solution. Currently you only obtained **2.5** points to your skill score from this metric.

Try using 2 CFS instead.

Impact of Change

By following the recommendation above your score will fundamentally improve on **skill**.

- **Change in Readability**: 6.02 -> 5.7436843 (**-4.8107758%**).
- **Change in Skill**: 5.38 -> 7.5675 (**28.90651%**).

PROFILE AFTER RECOMMENDATION After this change you'll reach an even better profile: **Advanced Beginner +**.

Congratulations on obtaining a balanced profile! However, you certainly have a lot to improve before reaching a better profile. Perhaps, it will be easier to focus first on either readability or skill, arriving at Proficient or Expert respectively before aiming for Master. Good luck!

B.3 SOLUTION A

Solution A can be seen in Codechef's interface here.

```
import java.util.*;

class COINS
{
    public static void main(String args[])
    {
        Scanner s=new Scanner(System.in);
        for(int i=0;i<10;i++)
        {
            long N=s.nextLong();
            System.out.println(find(N));
        }
    }
    public static long find(long N)
    {
        long s=N/2+N/3+N/4;
        if(s>N)
        {
            return(find(N/2)+find(N/3)+find(N/4));
        }
        else
        {
            return(N);
        }
    }
}
```

Listing B.3: Submission 18403997 (Solution A) to COINS

B.3.1 *Profile Analysis*

Your project which is named *18403997* has achieved a profile of **Advanced Beginner S**! The score obtained is **9.89** skill and **2.82** readability.

B.3.2 *Personalised Feedback*

> The system will now provide personalised feedback to help you improve your score. This is done by prioritising the easiness and impact in your current score.

*Suggestion: **Follow PMD Rule***

You violated rule ClassNamingConventions **1** times. This rule is part of the set Code Style
Description: The utility class name 'COINS' doesn't match '[A-Z][a-zA-Z0-9]+(Utils?|Helper)'
You have violated it in the following lines of the project:

- 3

GOAL You might still have some violations to improve on, but we advise you to more than just that. Check out: **Add more documentation.**

Impact of Change

By following the recommendation above your score will fundamentally improve on **readability**.

- **Change in Readability**: 2.82 -> 3.82 (**26.178009%**).
- **Change in Skill**: 9.89 -> 9.89 (**0.0%**).

PROFILE AFTER RECOMMENDATION Your profile won't suffer any change yet, but that is alright! Keep improving your readability above anything else in order to notice the biggest changes more quickly.

B.4 SOLUTION B

```
import  java.io.IOException;
import java.io.InputStreamReader;
import java.io.BufferedReader;
import java.util.*;
import java.math.*;
import java.io.*;

class hacker
{

    /*Fatt Gyi Bhai*/
    //Dekh le mera code
    public static boolean[] sieve(long n)
    {
        boolean[] prime = new boolean[(int)n+1];
        Arrays.fill(prime,true);
        prime[0] = false;
        prime[1] = false;
        long m = (long)Math.sqrt(n);
        for(int i=2;i<=m;i++)
        {
            if(prime[i])
            {
                for(int k=i*i;k<=n;k+=i)
                {
                    prime[k] = false;
                }
            }
        }
        return prime;
    }


    static long GCD(long a,long b)
    {
        if(a==0 || b==0)
        {
            return 0;
        }
        if(a==b)
        {
            return a;
        }
        if(a>b)
        {
            return GCD(a-b,b);
        }
        return GCD(a,b-a);
    }

    static long CountCoPrimes(long n)
    {
        long res = n;
        for(int i=2;i*i<=n;i++)
        {
            if(n%i==0)
            {
                while(n%i==0)
                {
                    n/=i;
                }
                res-=res/i;
            }
        }
        if(n>1)
        {
            res-=res/n;
        }
        return res;
    }

    static long modularExponentiation(long x,long n,long m)
    {
        long res = 1;
        while(n>0)
        {
            if(n%2==1)
            {
                res = (res*x)%m;
            }
            x =(x*x)%m;
            n/=2;
        }
        return res;
    }

    static long lcm(long a,long b)
    {
        return (a*b)/GCD(a,b);
    }
    static long pow(long a,long b)
    {
        long res = 1;
        while(b>0)
        {
            if((b&1)==1)
            {
                res *= a;
            }

            b >>= 1;
            a *=a;
        }
        return res;
    }

   static long modInverse(long A,long M)
   {
    return modularExponentiation(A,M-2,M);
   }
    static long fact(long n)
    {
        long res = 1;
        for(int i=1;i<=n;i++)
        {
            res = (res*i)%1000000007;
        }
        return res%1000000007;
    }


    public static void main(String[] args) throws IOException
    {
       // in = new Scanner(new File("explicit.in"));
        //out = new PrintWriter("explicit.out");
        new hacker().run();

    }


    static long digits(long n)
    {
        long res =0;
        while(n>0)
        {
            res+=n%10;
            n/=10;
        }
        return res;
    }





    static void run() throws IOException
    {

            //Scanner sc = new Scanner(System.in);
            BufferedReader br = new BufferedReader(new InputStreamReader(
                System.in));
            String str = "";
            while((str=br.readLine())!=null)
            {
                try
                {
                    long p = Long.parseLong(str);
                    System.out.println(getC(p));
                }
                catch(Exception e)
                {
                    break;
                }
            }
    }

    static long getC(long n)
    {
        if(n<12) return n;
        else return getC(n/2)+getC(n/3)+getC(n/4);

    }
    static long abs(long n)
    {
        return Math.abs(n);
    }



    static class Scanner
    {
        StringTokenizer st;
        BufferedReader br;
        public Scanner(InputStream s){  br = new BufferedReader(new
            InputStreamReader(s));}

        public String next() throws IOException
        {
            while (st == null || !st.hasMoreTokens())
            {
                st = new StringTokenizer(br.readLine());
            }
            return st.nextToken();
        }

        public int nextInt() throws IOException {return Integer.parseInt(next
            ());}

        public long nextLong() throws IOException {return Long.parseLong(next
            ());}

        public String nextLine() throws IOException {return br.readLine();}

        public boolean ready() throws IOException {return br.ready();}


    }




}
```

Listing B.4: Submission 20951241 (Solution B) to COINS

B.4.1 *Profile Analysis*

Your project which is named *20951241* has achieved a profile of **Advanced Beginner R**! The score obtained is **6.22** skill and **7.27** readability.

B.4.2 *Personalised Feedback*

The system will now provide personalised feedback to help you improve your score. This is done by prioritising the easiness and impact in your current score.

*Suggestion: **Follow PMD Rule***

You violated rule UnusedImports 1 times. This rule is part of the set Best Practices

Description: Avoid unused imports such as 'java.io' You have violated it in the following lines of the project:

- 5

GOAL You might still have some violations to improve on, but we advise you to more than just that. Check out: **There seems to be a better way to solve this exercise using fewer statements. Perhaps you over complicated the algorithm?**

Impact of Change

By following the recommendation above your score will fundamentally improve on **skill**.

- **Change in Readability**: 7.27 -> 7.52 (**3.3244705**%).
- **Change in Skill**: 6.22 -> 6.47 (**3.8639908**%).

PROFILE AFTER RECOMMENDATION After this change you'll reach an even better profile: **Advanced Beginner +**.

Congratulations on obtaining a balanced profile! However, you certainly have a lot to improve before reaching a better profile. Perhaps, it will be easier to focus first on either readability or skill, arriving at Proficient or Expert respectively before aiming for Master. Good luck!

B.5 SOLUTION C

```
//package codeforces;



import java.io.OutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.PrintWriter;
import java.util.HashMap;
import java.util.Scanner;
import java.util.StringTokenizer;
import java.io.IOException;
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.InputStream;

class ChefandSubsequence {
    public static void main(String[] args) {
        InputStream inputStream = System.in;
        OutputStream outputStream = System.out;
//        InputReader in = new InputReader(inputStream);
        Scanner in = new Scanner(inputStream);
        PrintWriter out = new PrintWriter(outputStream);
        TaskAA solver = new TaskAA();
        solver.solve(1, in, out);
        out.close();
    }

    static class TaskAA {
      static HashMap<Long,Long> memoize = new HashMap<>();
        public void solve(int testNumber, Scanner in, PrintWriter out) {
          memoize.put((long)0, (long)0);
          memoize.put((long)1, (long)1);
          while(in.hasNext()) {
            long n = in.nextLong();
            out.println(max(n));
          }
        }
        public static long max(long n) {
          if(memoize.containsKey(n))return memoize.get(n);
          long ret = Math.max(n, max(n/2)+max(n/3)+max(n/4));
          memoize.put((long)n, ret);
          return ret;
        }
    }

    static class InputReader {
        public BufferedReader reader;
        public StringTokenizer tokenizer;

        public InputReader(InputStream stream) {
            reader = new BufferedReader(new InputStreamReader(stream), 32768);
            tokenizer = null;
        }

        public String next() {
            while (tokenizer == null || !tokenizer.hasMoreElements()) {
                try {
                    tokenizer = new StringTokenizer(reader.readLine());
                } catch (IOException e) {
                    throw new RuntimeException(e);
                }
            }
            return tokenizer.nextToken();
        }

        public int nextInt() {
            return Integer.parseInt(next());
        }
        public long nextLong() {
            return Long.parseLong(next());
        }
    }
}
```

Listing B.5: Submission 19757960 (Solution C) to COINS

B.5.1 *Profile Analysis*

Your project which is named *19757960* has achieved a profile of **Proficient**! The score obtained is **5.96** skill and **8.82** readability.

B.5.2 *Personalised Feedback*

> The system will now provide personalised feedback to help you improve your score. This is done by prioritising the easiness and impact in your current score.

*Suggestion: **Follow PMD Rule***

You violated rule LooseCoupling **1** times. This rule is part of the set Best Practices
Description: Avoid using implementation types like 'HashMap'; use the interface instead
You have violated it in the following lines of the project:

- 29

GOAL You might still have some violations to improve on, but we advise you to more than just that. Check out: **There seems to be a better way to solve this exercise using fewer statements. Perhaps you over complicated the algorithm?**

Impact of Change

By following the recommendation above your score will fundamentally improve on **skill**.

- **Change in Readability**: 8.82 -> 9.15 (**3.6065598**%).
- **Change in Skill**: 5.96 -> 6.29 (**5.246422**%).

PROFILE AFTER RECOMMENDATION Your profile won't suffer any change yet, but that is alright! Keep improving your skill above anything else in order to notice the biggest changes more quickly

---

B.6 STUDENT STV, SOLUTION 1

```java
import java.util.HashMap;
import java.util.Map;
import java.util.Scanner;
class Coin {
    public static Map<Long,Long> calculus;

    private static long getmax(long x) {
        if(x <= 10) {
            return x;
        } else if(Coin.calculus.containsKey(x)) {
            return Coin.calculus.get(x);
        } else {
            long est =  getmax(x/2) + getmax(x/3) + getmax(x/4);
            Coin.calculus.put(x, Math.max(x, est));
            return Coin.calculus.get(x);
        }
    }

    public static void main (String[] args) {
        Scanner in = new Scanner(System.in);
        Coin.calculus = new HashMap<>();

        while(in.hasNextLong()) {
            long value = in.nextLong();
            System.out.println(getmax(value));
        }
    }
}
```

Listing B.6: First solution to COINS by STV

B.6.1 *Profile Analysis*

Your project which is named *STV1* has achieved a profile of **Advanced Beginner S**! The score obtained is **6.54** skill and **4.31** readability.

B.6.2 *Personalised Feedback*

> The system will now provide personalised feedback to help you improve your score. This is done by prioritising the easiness and impact in your current score.

*Suggestion: **Follow PMD Rule***

You violated rule UseUtilityClass **1** times. This rule is part of the set Design

Description: All methods are static. Consider using a utility class instead. Alternatively, you could add a private constructor or make the class abstract to silence this warning. You have violated it in the following lines of the project:

- 4

GOAL You might still have some violations to improve on, but we advise you to more than just that. Check out: **Add more documentation.**

Impact of Change

By following the recommendation above your score will fundamentally improve on **readability**.

- **Change in Readability**: 4.31 -> 4.64 (**7.112068%**).
- **Change in Skill**: 6.54 -> 6.54 (**0.0%**).

PROFILE AFTER RECOMMENDATION Your profile won't suffer any change yet, but that is alright! Keep improving your readability above anything else in order to notice the biggest changes more quickly.

B.7 STUDENT STV, SOLUTION 2

```
import java.util.Scanner;

class Coin {
    public static long getmax(long x) {
        if(x <= 10) {
            return x;
        } else {
            long est =  getmax(x/2) + getmax(x/3) + getmax(x/4);
            return x < est ? est : x;
        }
    }

    public static void main (String[] args) {
        Scanner in = new Scanner(System.in);
        while(in.hasNextLong()) {
            long value = in.nextLong();
            System.out.println(getmax(value));
        }
    }
}
```

Listing B.7: Second solution to COINS by STV

B.7.1 *Profile Analysis*

Your project which is named *STV2* has achieved a profile of **Expert**! The score obtained is **10.06** skill and **4.04** readability.

B.7.2 *Personalised Feedback*

> The system will now provide personalised feedback to help you improve your score. This is done by prioritising the easiness and impact in your current score.

Suggestion: ***Follow PMD Rule***

You violated rule UseUtilityClass **1** times. This rule is part of the set Design
Description: All methods are static. Consider using a utility class instead. Alternatively, you could add a private constructor or make the class abstract to silence this warning. You have violated it in the following lines of the project:

- 3

GOAL You might still have some violations to improve on, but we advise you to more than just that. Check out: **Add more documentation.**

Impact of Change

By following the recommendation above your score will fundamentally improve on **readability**.

- **Change in Readability**: 4.04 -> 4.37 (**7.551483%**).
- **Change in Skill**: 10.06 -> 10.06 (**0.0%**).

PROFILE AFTER RECOMMENDATION Your profile won't suffer any change yet, but that is alright! Keep improving your readability above anything else in order to notice the biggest changes more quickly.

B.8 PROGRAMMER HG

```
import java.util.Scanner;

class HelloWorld {

    private static int exchange(int coin)
    {
        int halfCoin = coin / 2;
        int thirdCoin = coin / 3;
        int fourthCoin = coin / 4;

        if (coin >= halfCoin + thirdCoin + fourthCoin)
        {
            return coin;
        } else {
            int sum = exchange(halfCoin) + exchange(thirdCoin) + exchange(
                fourthCoin);
            return coin < sum ? sum : coin;
        }
    }

    public static void main(String[] args)
    {
        Scanner in = new Scanner(System.in);
        while(in.hasNextInt()) {
            int coin = in.nextInt();
            System.out.println(exchange(coin));
        }
    }
}
```

Listing B.8: Solution to COINS by HG

B.8.1 *Profile Analysis*

Your project which is named *HG* has achieved a profile of **Expert**! The score obtained is **10.06** skill and **4.64** readability.

B.8.2 *Personalised Feedback*

> The system will now provide personalised feedback to help you improve your score. This is done by prioritising the easiness and impact in your current score.

*Suggestion: **Follow PMD Rule***

You violated rule UseUtilityClass **1** times. This rule is part of the set Design
Description: All methods are static. Consider using a utility class instead. Alternatively, you could add a private constructor or make the class abstract to silence this warning. You have violated it in the following lines of the project:

- 3

GOAL You might still have some violations to improve on, but we advise you to more than just that. Check out: **Add more documentation.**

Impact of Change

By following the recommendation above your score will fundamentally improve on **readability**.

- **Change in Readability**: 4.64 -> 4.97 (**6.639839%**).
- **Change in Skill**: 10.06 -> 10.06 (**0.0%**).

PROFILE AFTER RECOMMENDATION Your profile won't suffer any change yet, but that is alright! Keep improving your readability above anything else in order to notice the biggest changes more quickly.

B.9 PROGRAMMER MNDS

```
/* package codechef; // don't place package name! */
import java.util.HashMap;
import java.util.Scanner;

class Bank {
    private HashMap<Long, Long> exchanges;

    public Bank() {
        this.exchanges = new HashMap<>();
    }

    private long bestFit(long n) {
        long half = this.exchange(n / 2);
        long third = this.exchange(n / 3);
        long quarter = this.exchange(n / 4);
        long exchangedCurrencies = half + third + quarter;

        return exchangedCurrencies > n ? exchangedCurrencies : n;
    }

    public long exchange(long n) {
        if (n < 11) {
            return n;
        } else if(this.exchanges.containsKey(n)) {
            return this.exchanges.get(n);
        } else {
            long result = this.bestFit(n);
            this.exchanges.put(n, result);

            return result;
        }
    }


    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        Bank b = new Bank();

        while(in.hasNextLong()) {
            long n = in.nextLong();
            System.out.println(b.exchange(n));
        }
    }
}
```

Listing B.9: Solution to COINS by MNDS

B.9.1 *Profile Analysis*

Your project which is named *MNDS* has achieved a profile of **Advanced Beginner +**! The score obtained is **7.13** skill and **5.92** readability.

B.9.2 *Personalised Feedback*

> The system will now provide personalised feedback to help you improve your score. This is done by prioritising the easiness and impact in your current score.

*Suggestion: **Follow PMD Rule***

You violated rule LooseCoupling **1** times. This rule is part of the set Best Practices
Description: Avoid using implementation types like 'HashMap'; use the interface instead
You have violated it in the following lines of the project:

- 6

GOAL You might still have some violations to improve on, but we advise you to more than just that. Check out: **You could use fewer control flow statements. Try reducing the number of cycles, that usually means there is an easier way to solve this exercise.**

Impact of Change

By following the recommendation above your score will fundamentally improve on **skill**.

- **Change in Readability**: 5.92 -> 6.25 (**5.279999**%).
- **Change in Skill**: 7.13 -> 7.46 (**4.4235916**%).

PROFILE AFTER RECOMMENDATION Your profile won't suffer any change yet, but that is alright! Your profile is balanced, however you have a lot to improve before reaching a better profile. Perhaps, it will be easier to focus first on either readability or skill, arriving at Proficient or Expert respectively before aiming for Master. Good luck!

C

EXERCISE CODECHEF JOHNY

This appendix introduces feedback to an exercise which be found in JOHNY.

C.1 STUDENT RSND

```
import java.util.Scanner;
import java.util.ArrayList;
class  Main {
    public static void main(String[] args) {
        int n_tests;
        ArrayList<Integer> array;
        int number_array;
        int current_k;
        int value;
        int count;
        Scanner in = new Scanner(System.in);
        array = new ArrayList<Integer>();
        n_tests = in.nextInt();
        for (int i = 0; i < n_tests; i++) {
            number_array = in.nextInt();
            for (int j = 0; j < number_array; j++) {
                array.add(in.nextInt());
            }
            current_k = in.nextInt();
            value = array.get(current_k - 1);
            count = 0;
            for (int it : array) {
                if (it<value)
                    count++;
            }
            System.out.println(count + 1);
            array.clear();
        }
    }
}
```

Listing C.1: Solution to JOHNY by RSND

C.1.1 *Profile Analysis*

Your project which is named *RSND* has achieved a profile of **Advanced Beginner S**! The score obtained is **9.25** skill and **2.97** readability.

C.1.2 *Personalised Feedback*

> The system will now provide personalised feedback to help you improve your score. This is done by prioritising the easiness and impact in your current score.

*Suggestion: **Follow PMD Rule***

You violated rule LocalVariableNamingConventions 3 times. This rule is part of the set Code Style

Description: The local variable name 'N' doesn't match '[a-z][a-zA-Z0-9]*' You have violated it in the following lines of the project:

- 5
- 7
- 8

GOAL You might still have some violations to improve on, but we advise you to more than just that. Check out: **Add more documentation.**

Impact of Change

By following the recommendation above your score will fundamentally improve on **readability**.

- **Change in Readability**: 2.97 -> 5.9700003 (**50.25126**%).
- **Change in Skill**: 9.25 -> 9.25 (**0.0**%).

PROFILE AFTER RECOMMENDATION After this change you'll reach an even better profile: **Advanced Beginner +**.

Congratulations on obtaining a balanced profile! However, you certainly have a lot to improve before reaching a better profile. Perhaps, it will be easier to focus first on either readability or skill, arriving at Proficient or Expert respectively before aiming for Master. Good luck!