SOFTWARE DEVELOPMENT

Computer Science

uc3m | Universidad Carlos III de Madrid

Guided Exercise 4

Course 2022/2023

Group 89 -> Inner Group 20

Sergio Barragán Blanco, Eduardo Alarcón Navarro

100472343 , 100472175

100472343@alumnos.uc3m.es, 100472175@alumnos.uc3m.es

Table of Content:

Lint-Based Warnings in Python Code: Frequency, Awareness and Refactoring

This article starts talking about the uses and the importance of linting, which is analyzing code to stay and find possible errors or mistakes that might occur during the execution. Then, the article continues to list some actions such as avoiding the naming of variables the same as the types, such as dict for dictionaries, redefining built-in functions, using is instead of ==, enumeration and other uses of is and the Default modes. The paper is based on doing visceral small studies, the first of all being the analysis of more than 1000 open source python projects and checking the different linting warnings that might occur. To make the study the most accurate possible, they took big projects, small projects, in general, diverse projects. As a small summary, the biggest appearance is the lack of use of the “with” statement, that allows the programmer to use the indent to have a file open. When the indent is restored, the file is closed. The study also saw that there were a lot of false positives, which concerns the second study, where they analyzed if the developers preferred code written with or without lint-based warnings. As well as with the first study, the study received information from 50 programmers from 198 different countries. From all the code snippets shown to the participants, most of the snippets favored the refactored version, as, at least for me when reading the paper, most of them were cleared and easier to understand. There is an exception to this: the DDF or Dangerous Default Value, which had almost 45% of participants preferring the non-refactoring code. The next study is based on the pull request of other projects, as the space is limited, I will only say that most of the developers care about the lint warning pull request and take time to fix the code. I want to clarify this was about python lint-based warnings, not about other languages.

Link to the article: https://ieeexplore.ieee.org/document/10006861/

Code refactoring a Python example

This paper is an introduction to several refactoring techniques using python. Mainly, it uses an example of a code in python, using an implementation of abstract classes (Elf, Draw and Wizzard) and superclasses (Hero). This paper also tests and looks for code smells such as methods with too many lines or methods that have more than one functionality implemented inside them. Other “smelly” aspects of the code happen when modifying the code, for example not adding the necessary comments, duplicating comments, having Lazy classes or having a middle man (which, as a small description is a class that only delegated the work to another class, it does not provide any work.) Then, the paper process has an example, which is about the need to perform some calculations. Afterwards, it does the refactoring, by applying some steps such as separating the calculations, using a specific method to test the function make_calculations. Lastly, it extracted five methods to show the potential of refactoring, naming each method to better understand its functionality.

Then, it proceeds to state the Patient Manager, as the one we have done with the order manager, but with the Patients, showing a diagram with the inheritance of the classes and its functionality. It is showing how to refactor a code, indicating bad smells and some proposed improvements, adding in the end that a Parameterized Factory Method design was applied (Which is a design method that, before creating the object, identifies the type that needs to be created). This aspect is super important as the refactoring code allows not only the programmer to identity what the program does, but in the future, it maintains the code easier.
Link

Refactoring Codes to Improve Software Security Requirements

This paper is mainly about the impact of refactoring in security, it often says that articles usually talk about methods of refactoring and only a few really talked about their impact in performance on a given software. Furthermore, none of this really mentioned security requirements, which nowadays is one of the most important aspects while designing and implementing a new software. In order to strengthen his point, there's a table as a Summary of existing studies on refactoring techniques categorization based on their effect on software quality attributes, of course, security was not included. In order to make this study, a method of individual effects after applying the techniques and a cumulative effect was taken into account. Among all the different types of refactoring it was only observed five of the most common ones (Extract Method (EM), Inline Method (IM), Encapsulate Field (EF), Remove Setting Method (RSM) and Hide Method (HM) it was used java software and some features of management systems in order to study the impact of applying this method as well as what could be changed after applying the method (the difference on security in public or private methods, nº of instances..)
After the tests, it was shown that, as expected, some methods such as RSM and EM did not contribute much to the security and were counterproductive but can still be applied in some specific areas such as remove or extract public and private methods respectively. HM and IM

actually improved the security (which was expected since those methods are focused on reducing the number of public methods) but did not affect the CIDA or CCDA because accessibility was not changed. And finally, EF changes the public field to private, therefore, it has a positive effect on the CIDA and CCDA but increases the number of class operations, worsening the COA.
This study proves that software engineers can’t use refactoring without studying what’s best for their project since common methods such as EM or RSM, when applied wrongly, it can actually worsen the security.

https://www.sciencedirect.com/science/article/pii/S1877050922007517
Note: definitions of COA, CIDA and CCDA provided by the paper

- COA is calculated by dividing a number of Classified Public Methods (CPM) by the total number of Classified Methods (CM) in a class.
- CIDA is calculated by dividing a number of Classified Instance Public Attributes (CIPA) by the total number of Classified Attributes (CA) in a class.
- CCDA is calculated by dividing the number of Classified Class Public Attributes (CCPA) into the total number of Classified Attributes (CA) in a class.

Software Refactoring Prediction Using SVM and Optimization Algorithms

This paper measures the importance of automating the refactoring techniques, to prove this, a SVM was implemented with a whale algorithm and a genetic algorithm. It would be more accurate to say that firstly the SVM was implemented on the datasets and preprocessing, which would fix naming errors, paths and delete some unnecessary components, but then, the SVM is combined with the genetic algorithm (“one of the most used when optimization algorithms in software fault predictions”) and with the whale algorithm (a new yet interesting algorithm with discerns between the best and worst solutions to get the optimal one) separately, which on their own they prove to be useful to some extent as shown in the tables attached to the article. Moreover, the data and methods utilized are public so they can be proven in case of need or learning. Regarding the test, the accuracy was promising in many of the chosen test cases, improved when both learning algorithms were combined, but the one that stood out the most was the F-Measure, since the ML algorithms were also applied, yet the SVM and the combination of both algorithms managed to get better accuracy. This will help programmers to identify which tasks need refactoring and may be a step forward to the automatization of refactoring techniques.

https://www.mdpi.com/2227-9717/10/8/1611