1. Code refactoring: a Python example

The article shows how a refactoring process is needed before applying significant changes to code, since the longer you take to start refactoring the code the harder it will be to complete the task. The pieces or structures of the code where refactoring may be needed as they could cause mayor problems in the future are introduced as “code smells”, for example long methods or dead code (code that is not used).

The paper explains how to detect and solve code smells through a Python code example in which the recommended intake of calories is trying to be calculated. This example helps us realize the importance of creating relatively small and well-defined methods and taking advantage of object-oriented programming in order to solve refactoring problems. It is also recommended to have a graphic or visual design of the code to be implemented to seek for the most convenient solution.

Additionally, some unit tests are provided to check the functioning of the code, and we can observe that they are quite similar to what be have been working with in class.

file:///C:/Users/isabe/Downloads/Code_refactoring_a_Python_example%20(1).pdf

2. Refactoring Codes to Improve Software Security Requirements

This article intends to measure how refactoring code may guarantee the security of the software developed. For that aim, five refactoring techniques are studied: extract method, inline method (which consist on deleting methods that are extremely simple and implicit, which would be the opposite of extract method), encapsule field (which consists on making a field private), remove setting method and hide method (privatizing a method). Some test cases are selected, considering their number of classes, lines of code, size and environment (academic or professional). The test cases are all Java projects from different sources.

The security is measured by several attributes such as the number of public methods, public classes or instance public attributes. After applying the corresponding refactoring techniques it is shown that the inline method and hide method improve the security of the software, while extract method and remove setting method do the opposite, and encapsulate method improves some security aspects and degenerates others.

All in all, this case study gives us an idea of what refactoring techniques are convenient when looking to maximize the security of our software systems. However, we do not think the conclusions reached by this research can be considered as a rule as very few examples (only five) were taken into account.

https://www.sciencedirect.com/science/article/pii/S1877050922007517′

3. A longitudinal study of the impact of refactoring in android applications

The objective of this study is to investigate the impact of refactoring in Android apps. To achieve this objective, the authors did an empirical study by analysing the evolution of five Android apps which were refactored along the multiple commits. The projects selected were written in Java and were required to have more than 1000 commits to make sure it was a serious project and to obtain the most relatable amount of data.

In this study, the authors focus on 15 refactoring operations, including Move Method, Extract Method, Inline Method, Rename Method, Extract and Move Method, Push Down Method, Pull Up Method, Move Attribute, Push Down Attribute, Pull Up Attribute, Extract Super Class, and Rename Class. By analysing the impact of these refactoring operations, the authors aim to improve the overall quality of Android app development.

To understand the issues better, the investigators ask some questions and try to answer them through the analysis of the data. They find it interesting to analyse the amount of code smells present in the code after several cycles of refactoring on the project, they also try to obtain statistical data on the most common code smells, which can help analyse future projects. It is also important to analyse the cases in which the refactoring is performed and those in which it is not. This can also provide insight in which are the refactoring operations that developers find more important, and which are less used

Of course, there is also an analysis of not only the type of refactoring that is applied, but also the impact the operations applied have on performance and on improving code quality.

The results show that code smells are widespread in Android applications, it also found that refactoring operations do not particularly target smelly classes or bad practices, they found that it is rare for them to do so. These are very interesting conclusions and show that the analysis is not as straightforward as it may seem and may need a more focused approach to understand the specific issues of refactoring in Android apps.

https://www.sciencedirect.com/science/article/pii/S0950584921001531

4. A Field Study of Refactoring Challenges and Benefits

This survey studies the benefits of refactoring, considering the opinions and thoughts of a large sample of computer scientists (developers, engineers, managers, etc). In general, they assure that it is important to take into account the risks and challenges of refactoring, as the initial approach and goals of a project can be frustrated and deviated if the software requirements are not kept in mind during all the process. However, it is concluded that the risk is worthy due to the simplicity to use and modify the refactored code.

They were also asked about their definition of refactoring and its goals, and we found interesting that one of them exposed that it is usually done while trying to fix bugs and adding new functionalities, not just to make the code simpler or more fragmented.

https://dl.acm.org/doi/pdf/10.1145/2393596.2393655