**SOFTWARE DEVELOPMENT**

**Bachelor’s Degree in Computer Science and Engineering**

### 

Guided Exercise 4

Alicia Benítez Rogero (100451005)

Beatriz Sonsoles Encinas Muñoz (100451169)

Group 89

**INDEX**

[**Articles on Refactoring**](#_kxd8eazbxrk3) **3**

[**Software Refactoring Side Effects**](#_54bn0vllmif) **3**

[**How Software Refactoring Impacts Execution Time**](#_bvfzva5c2s8h) **3**

[**Refactoring for Reuse: An Empirical Study**](#_memkijicl39x) **4**

[**A prototype for software refactoring recommendation system**](#_9fn957s1rovt) **4**

[**References**](#_t0fqhc9aqv26) **5**

## 

## **Articles on Refactoring**

### **Software Refactoring Side Effects**

Refactoring is a process whose objective is to improve the quality of the software, as well as ensure its maintainability and facilitate its understandability.

The article points out that most regular developers focus first on understandability and maintainability, rather than on refactoring, which is an important part to keep in mind. It also remarks on the fact that refactoring does not guarantee system enhancement so any developer must carefully identify the problems that require fixing and use the appropriate refactoring technique.

The importance of this article resides in the fact that it addresses the side effects of refactoring, often ignored, and provides three approaches to taking care of any new issues that refactoring may cause. These approaches are: firstly, ignoring any new problems, secondly, fixing those problems first, and thirdly, taking into account the side effects to reschedule the refactoring sequence, as altering the sequence affects the quality.

The text also mentions some related work, in which the side effects are not taken into account and proposes a six-step methodology for reducing the side effects of refactoring. The results of applying this methodology to four different open-source projects are shown as well, for illustration. Although the experiments carried out have some limitations that are mentioned in the test, we can conclude that handling the side effects of refactoring improves the quality of the software.

### **How Software Refactoring Impacts Execution Time**

Software systems are constantly changing so developers focus mainly on code maintainability and do not consider the side effects of recommended refactoring warnings. These warnings may have a huge impact in the long run on non-functional requirements, such as maintainability itself, performance, or readability. The text mentions the work of other researchers in the refactoring field and points out that there is much more to find out about

the relationship between refactoring and the non-functional requirements. This study analyses sixteen different types of refactorings concerning execution time. It was discovered that refactoring operations such as Extract Class and Move Method affect execution time both positively and negatively. Through benchmark techniques, it was shown that even a small effect on performance at the method level may have an enormous impact on performance at the system level, thus making refactoring a process to be highly considered.

Despite the investigation not having a wide range of projects, which means not every possible case is covered, and the fact that further investigating is needed, we can conclude

from the obtained results that the impact of refactoring on execution time depends on the refactoring type. Therefore, developers should give greater importance to the refactoring process.

### **Refactoring for Reuse: An Empirical Study**

Several investigators focus their studies on refactoring strategies to optimize software. This can be carried out by applying design patterns, removing duplicated code, or extracting methods, among many others. Nonetheless, no studies break down the way developers make use of refactoring to enhance its reusability. This article analyses the refactoring regarding reusability applied to different Java open source projects, through which they discovered that reusability refactorings have a different distribution than other types of refactoring. This is because reusability refactorings impact code components such as packages, classes, or methods.

The results of the project showed that such refactorings affected methods a great deal more than comments or cohesion of classes. It was also discovered that out of thousands of commits that were analyzed, only less than one percent were commits related to reusability refactoring. Thus, it is demonstrated that developers do not take into account such a necessary feature as being able to reuse code in the future.

### **A prototype for software refactoring recommendation system**

Multiple researchers have focused on the topic of software refactoring, a technique that improves code readability, maintainability, and extensibility. Nevertheless, this technique is difficult to master and costly so many developers tend to neglect it. Even the ones who decide to apply the refactoring to their software find it difficult to choose between the different refactoring methods the one that best suits them. Although refactoring tools facilitate the task of refactoring for the developers, the article proposes and implements a mock-up of a software refactoring recommendation system. This implementation is developed on Eclipse and it consists of an already-developed graphical user interface and an integrated natural language processing tool.

The recommendation system will deal with problems such as refactoring in the wrong order, overlooking bad smells that are hidden, or having a complex and unclear code. The functionality of this system is checked by applying refactoring on different projects. The results showed that in around 85 percent of the cases, the proposed tool was a great help when refactoring the code correctly. Hence, it would be interesting to further expand this system so that software developers can use it as a reliable option to refactor, instead of skipping this important step.

## **References**

[1] A. AbuHassan, M. Alshayeb and L. Ghouti, "Software refactoring side effects," J Softw-Evol Proc, 2021. . DOI: 10.1002/smr.2401, URL: <https://doi.org/10.1002/smr.2401>

[2] L. Traini et al, "How Software Refactoring Impacts Execution Time," ACM Transactions on Software Engineering and Methodology, vol. 31, (2), pp. 1-23, 2022. . DOI: 10.1145/3485136, URL: <https://link.springer.com/article/10.1007/s11334-021-00422-6>

[3] E. A. AlOmar et al, "Refactoring for Reuse: An Empirical Study," 2021, URL: <https://dl.acm.org/doi/full/10.1145/3485136>

[4] Y. Gao et al, "A prototype for software refactoring recommendation system," International Journal of Performability Engineering, vol. 16, (7), pp. 1095-1104, 2020. . DOI: 10.23940/ijpe.20.07.p12.10951104,URL: <https://web.s.ebscohost.com/ehost/pdfviewer/pdfviewer?vid=0&sid=0912ca61-73ca-4d1f-8b43-a7d1e2aef58e%40redis>