Universidad Carlos III

Software development 2022-23

Course 2022-23

GE4_2023_GROUP_05

88

József Iván Gafo (100456709)

Marcos González Vallejo (100472206)

Index

Publications: 3
- **Article 1:** 3
- **Article 2:** 4
- **Article 3:** 5
- **Article 4:** 6

Publications:

Article 1:

Title: Refactoring embedded software: A study in healthcare domain
Published on: Information and software technology
Web:
https://www.sciencedirect.com/science/article/abs/pii/S0950584921002068?casa_token=PzQLb5AnbLMAAAAA:KXRVpqTJcqpSpcMxaDIoMey9Jw1V6Q2N9KuZkH-t14BkKr-3mPpkZcoRnHa1CvZ_gO6mPu4nYA
Author: Paraskevi Smiari, Stamatia Bibi, Apostolos Ampatzoglou, Elvira-Maria Arvanitou.
Date of publication: March 2022
Summary: To start with, as code changes and becomes updated through time, it may become more complex which provokes a reduction in quality. Therefore, there is a need for maintenance to maintain the quality of the software. One of the most used activities to improve the quality is refactoring which, as defined by Fowler, is the transformation that improves quality attributes without affecting the external behavior. Additionally object oriented has many benefits in embedded software but it does not have the necessary attention yet, that's why this software needs techniques to improve its design quality. The refactoring strategy follows the following steps: Refactoring planning, in which the engineer should select the quality attributes to be improved and the sub systems that need refactoring; Refactoring design, in which the engineer chooses which refactorings to apply based on the problems in the base code and quality attributes selected in step1; Refactoring evaluation in which the engineer selects the criteria used for evaluating the improvements of the refactoring. The results are presented, first, the refactoring process and actions to identify the areas of improvement, second, the most frequent refactorings and finally, the evaluation of the changes performed by the refactorings.

Why is it relevant? We found this article relevant for our case because in our opinion, it shows the steps to follow when refactoring in a way which makes a lot of sense which is: selecting those areas where the code can be improved, selecting the refactoring technique and finally evaluate the results.

Article 2:

Title: A DQN-based agent for automatic software refactoring
Published on: Information and software technology
Web:
https://www.sciencedirect.com/science/article/abs/pii/S0950584922000556?casa_token=6KZ6_wimcfUAAAAA:N1-d46_myRlc91rtPI7pLAu126--1qiCGRIXBm460pTqfTiS5QAL9Grs4N3b96fbjzWg0PXfhw
Author: Hamidreza Ahmadi, Mehrdad Ashtiani, Mohammad Abdollahi Azgomi, Raana Saheb-Nassagh
Date of publication: July 2022
Summary: To start with, as time goes on, software becomes unstable due to updates (technical debt). This debt can lead to failures in quality of maintenance and flexibility. To solve this problem, refactoring is used, although one of the simplest methods which is manually, is too costly and may not succeed. This is why automatic and semiautomatic models have been developed. Some types of refactoring are search-based refactoring, which consists of discovering new structures by small refactoring actions thanks to metaheuristic search algorithms. Other methods are code smells, which try to refactor with design patterns or algorithms. However, search based algorithms are not too good in time and uncertainty of the result (different outputs) that is why the proposed model tries to cover disadvantages like the ones mentioned. Uncertainty makes it difficult to find the optimal solution and it will need help from the developers. The idea is to introduce a new approach for refactoring which is similar to search-based refactoring, but taking care of uncertainty and time consumption. To do so, a reinforcement learning-based approach is used to make decisions based on previous knowledge when refactoring.

Why is it relevant?
We found this article relevant because it talks about several approaches that we could take when refactoring. Some of those approaches are search-based algorithms and code smells where the refactoring model recognizes patterns based on its previous knowledge.

Article 3:

Title: **Refactoring Codes to improve Software Security Requirements**
Published on: **Procedia Computer Science**
Web: https://www.sciencedirect.com/science/article/pii/S1877050922007517
Author: **Abdullah Almogahed , Mazni Omar , Nur Haryani Zakaria**
Date of publication: **10 September 2022**
Summary:
This article is conducting an investigation to prove whether some refactoring techniques can increase or decrease the security of our code. The researchers used 3 types of cybersecurity metrics that are “Cost of attack”, “Cybersecurity Investment and Development Assessment” and “Cybersecurity Capability and Development Assessment ”. The researchers used 5 study cases where they used some refactoring techniques that are “extract method”(EM), “Inline Method”(IM), “Encapsulate Field”(EF), “Remove setting method”(RMS) and “Hide Method”(HM) . After running the test and analyzing the different study cases they reach the conclusion that some refactoring techniques had a worse impact on security because they were using public visibility which would mean that external code could access it and manipulate it. As a result the researchers recommended using private visibility if possible to increase the cost of attack. On the contrary, it was also found that in some cases it had a better impact on security because they were restricting access to the methods.

Why is it relevant?
This article is relevant because it warns us that we shouldn't forget to protect our code when we are in the process of refactoring our code.

Article 4:

Title: Unveiling process insights from refactoring practices
Published on:Computer Standards & Interfaces
Web:
https://www.sciencedirect.com/science/article/pii/S0920548921000829?casa_token=nUXVEED90G0AAAAA:-r-QGFjaoAMbhOKAHi7hC1C5FvN0nZHao3QMBxAQ7UDEBMwe87auKNSFw7LY3BDtmp3I05bTTQ
Author: **João Caldeira , Fernando Brito Abreu , Jorge Cardoso , José Pereira dos Reis**
Date of publication: April 2022
Summary:
This article is trying to solve 4 questions that are “How do different refactoring methods perform when the goal is to reduce complexity, future testing and maintainability efforts?” , “Is there any association between software complexity and the underlying development activities in refactoring practices?” , “Using only process metrics, are we able to predict with high accuracy different refactoring methods?” and “Using only process metrics, are we able to accurately model the expected level of complexity variance after a refactoring task? “. To do this the researchers did 2 types of refactoring (automatic and manual refactoring) and used plugins to collect the data to later do statistical analysis. After the statistics analysis was performed, the researchers discovered that the automatic refactoring reduced much more the code complexity than the manual refactoring techniques, also they didn’t find a strong correlation between software complexity and underlying development activities and finally   they also discovered that random forest machine learning algorithm were able to predict with a high accuracy in refactoring opportunities meaning it would detect more opportunities to programmers to refactor our code.

Why is it relevant?
This article is relevant because it proves that using automatic refactors is more beneficial because it reduces code complexity and also using it with a machine algorithm it allows the programmers to know when they have to refactor, reducing the time spent by the programmer to look out refactor opportunities.