Universidad Carlos III

Software development 2022-23

Guided exercise 4

Group 10

Course 2022-23

TABLE OF CONTENTS

ARTICLE 1

https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=d6aa76bab896ed6257c410671ea937c95be9c490
Arie van Deursen, Leon Moonen, Alex van den Bergh and Gerard Kok in CWI, Software Improvement Group The Netherlands. “Refactoring Test Code”

It is widely accepted that unit testing is one of the most important programming activities in extreme programming. Moreover the aim of this type of code is to have a 1:1 ratio. 1 unitest class for every class in the code. In these classes the tests verify complicated functionalities and unusual circumstances to check everything is working correctly. Also, it allows more changes to be made, since programmers can easily check if their changes are correct by simply running the tests and checking that everything is still working. Nonetheless, it is important that the test cases can be easily modifiable. As such a big percentage of all the codes are unitesting it makes sense for that code to be refactored. This article claims that the “smells” for unitesting are different from those of normal coding. The main test code smells identified by the article are: *Smell Guest* : External usage of resources in tests, which reduces self containment and introduces hidden dependencies. *Test run war* : It occurs when tests work fine when a single programmer executes them but fails if several of them do. Most likely do to resource inference. *Eager test:* A test method that checks several methods of the object making it hard to read and understand. *Lazy test:* When several methods check the same method using the same structure (but different values). *Assertion roulette:* When there are assertions in a test with no explanation. *Indirect testing:* Smells when a test method starts performing tests on other objects it was not originally designed to test. If this arises might indicate there are problems with data hiding in production code. *Sensitive equality:* Occurs when assert equal outputs based on methods like toString. Which may lead to trouble since a different comma, quote or space may change the result and arise an error

Although Bad smells seem to arise more often in production code than in test code, one should not underestimate the importance of having fresh test code. Test refactoring are those changes of test code that do not add or remove test cases, and make the test code better understandable and maintainable. The main test code refactoring the article highlights are: *Inline Resource*. To remove the dependency between a test method and some external resource via setting up a fixture in the code that holds the same content as the resource. *Setup External resource;* if a test relies on external resources it must make sure to create or allocate it before actually executing the test. *Make Resource Unique:* Make all resources names unique so there is no possibility of overlapping. *Reduce Data:* Reduce the data use to the essential so that the tests are less sensitive to changes. In conclusion the article studies the main bad smells in unitest coding and good practices to refactor test code since it is a very big part of all the extreme programming projects.

ARTICLE 2

We analysed the article "Refactoring - Does it improve Software quality?" (https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4273477). Published in IEEE.

Refactoring is one of the best methods for improving software quality by reducing complexity and improving maintainability. One of the problems is to know where to use refactoring. For that there are several methods, one of the most important ones is software metrics.This article studies whether refactoring really improved quality for several real-life software projects. The method compares the initial state of the code before applying refactoring and after. Several attributes in the metrics were used with different parameters to try and get a vision as wide as possible of all the effects the changes had. Some of the sections to check whether the quality has improved were: *McCabe's Cyclomatic Complexity,* Which analyses the number of linearly independent paths, *WMC* (Weighted methods per Class), *DIT* (Depth of Inheritance tree) amongst others. Hence it offered a general view of the code state. After applying the method to: Apache, MySQL connector, JBoss Hibernate, Which are some real life projects, while these projects were being developed, the researchers applied them before and after any refactoring commit in different stages of such development . The study concludes that although it would be expected that the improvement is reflected in the various metrics, this is not the case for these systems. Furthermore, the results indicate a significant change of certain metrics to the worse. This suggests that either the refactoring process doesn't always improve the quality of a system in a measurable way or that the developers have not managed to use refactoring effectively sa mens to improve software quality.

ARTICLE 3

https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9825885

G. Sellitto et al., "Toward Understanding the Impact of Refactoring on Program Comprehension," 2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER), Honolulu, HI, USA, 2022, pp. 731-742, doi: 10.1109/SANER53432.2022.00090.

Toward Understanding the Impact of Refactoring on Program Comprehension

Refactoring is one of the most important techniques used to improve the code smells in the main source code, to improve the code for future development. Several issues have to be faced when refactoring, such as the lack of usable automated tools, fear of changing the uses and capabilities of the code or poor confidence in the positive outcome of refactoring.

Several studies have been conducted, and proven that when not done properly, refactoring can create more code smells than it solves. It has also been found that the reorganization of code in smaller components has a positive impact on readability. The biggest benefits in refactoring are given when several techniques are applied, giving overall more cohesion to the program and improving readability. Renaming code elements was expected to improve readability, but it does not do so in practice, as changing names can be confusing for the programmers that first worked on the code and on their own naming conventions.
In general, the techniques that affect readability the most are the ones regarding extracting methods and creating new class members down the hierarchy. Although it can drastically improve readability, if not used properly, it can lead to harmful consequences, making the code more confusing than before.

Overall, it can be confidently stated that refactoring for sure has an effect on readability. It is not always positive, but with proper cohesion and techniques it can have many benefits in time for the code maintenance and future improvements on features.

ARTICLE 4

James Ivers, Robert L. Nord, Ipek Ozkaya, Chris Seifried, Christopher S. Timperley, and Marouane Kessentini. 2022. Industry experiences with large-scale refactoring. In Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2022). Association for Computing Machinery, New York, NY, USA, 1544–1554.
https://doi.org/10.1145/3540250.3558954

Industry Experiences with Large-Scale Refactoring

Refactoring is generally a task that does not take very long and is given to a software developer to finish in a day. This happens only when the project is smaller and has been taken care of. In bigger projects, that have been in development for several years and have been receiving patches for a very long time, without taking into account refactoring techniques, it can be a much bigger problem that requires the efforts of whole teams for days or even weeks.
After a thorough survey, it has been seen that large-scale refactoring is a major issue that over 80% of programmers have done at least once in their careers, with more than 60% saying they have done it more than once, with all this projects taking from 2 to 20000 staff days, in the most extreme cases, taking usually months to years range as average.
Most of this project had been taken in order to improve the code understanding, especially for large projects of code updation in the future.
The large-scale refactoring is very challenging, and the biggest issue it faces is the lack of tools to use, which makes the time investment needed very big.
All in all, even with the lack of tools, most of the refactoring attempts have been successful, as there is no other choice other than making the effort, as it is overall worth to invest time in refactoring due to all the problems that not doing it creates, especially for large-scale projects.