Universidad Carlos III de Madrid

Software Development

Guided Exercise 4 Writeup

Section C9.84.19637-1

Krislyn Dorsey, Megana Atluri, and Sarah Hoover

100504996, 100504469 and 100504446

April 28, 2023

Table of Contents

1. INTRODUCTION 3
2. ATTRIBUTES 4
3. STORES 5
4. MODELS 6
5. KEYS 7
6. EXCEPTIONS 8
7. SINGLETON PATTERN 9
8. ARTICLES SUMMARIES 10
9. CONCLUSION 15

1. Introduction

The purpose of this guided exercise was for the class to practice a variety of refactoring techniques and to explore simple and basic design patterns. We were given working and well-tested code and were tasked with refactoring, redesigning, and reorganizing it. Since the code we were originally provided was operational, this assignment allowed us to focus on practicing design versus code functionality.

As well as renaming functions and variables and other minor changes, our refactoring of this program focused on five main parts. The six parts were reorganizing the attributes, the stores, the models, the keys, the exceptions, and implementing the singleton pattern.

2. Attributes

The first aspect that we worked on for refactoring was the attributes. When the project was first downloaded, the validation tests for the address, order type, phone number, and zip code all existed in the register order function located in the order_manager.py file. This caused the register order function to be long and complicated. In order to reduce the size of this method, we extracted the section that tests the validation of the address, order type, phone number, and zip code. We used extract function methodology to complete this. At this point there were separate validation functions for the product id, address, order type, phone number, and zip code in the order_manager.py file. This version of the code, however, was still not clean. The order_manager.py file is to manage orders that have already been requested. Because of this, we thought it made more sense to move these validation methods to order_request.py. To do this we used the move function. Once we refactored the code to move the validation to order_request.py we noticed some other issues. Now, order_request.py was too long. In order to combat this, we made a validation directory and created an Attribute abstract base class. Each validation attribute inherits from the base class. By doing this we were able to clean up the register order function as well as the order_request.py class. We followed similar functionality for the tracking code, email, and order id attributes.

3. Stores

Data can be stored more straightforwardly and flexibly using JSON stores, which is one method refactoring can use. For example, in our code, we had to make four different stores -- JSON Store, Order Delivery Store, Order Request Store, and Order Shipping Store. JSON stores can also be used in refactoring by enabling developers to divide large programs into more manageable, minor services. In our program, we developed a JSON store class for taking most of what we found in Order Manager that can be handled separately and can be taken from a separate class. Order Delivery, Shipping, and Request stores are essential microservices, which might be easier to refactor and maintain than a single application and can employ JSON stores as their data layer. In addition, it is simpler to alter one service without affecting the others when programmers use JSON stores to create disconnected services. Furthermore, our JSON stores can assist with refactoring efforts by offering a more advanced and adaptable data storage option. We found it simpler to add new features or enhance current ones due to not needing to make significant adjustments to the underlying database structure. When working with the original code given, JSON stores can be a helpful tool in refactoring efforts since they give our team more flexibility and scalability.

4. Models

In order to organize our classes, we created a models folder. We moved our order_delivery.py, order_request.py, order_shipping.py, and send_product_input.py into this subfolder. It is important to note that we did not move order_manager.py into the folder because OrderManager is not a model. Overall, even though we did not have to change any functionality when creating the models folder (other than implement an init file), the models folder was integral to our code as it allowed for more readable and overall cleaner code organization.

5. Keys

To implement the keys, we created a module called keys in the models folder. The purpose of the keys is to hold the necessary keys for the models class. This means that we have four key classes: OrderDeliveryKeys, OrderRequestKeys, OrderShippingKeys, and SendProductInputKeys. Each class holds its respective keys. For instance, OrderRequestKeys holds the following: ZIP_CODE = "_OrderRequest__zip_code", TIME_STAMP = "_OrderRequest__time_stamp", PHONE_NUMBER = "_OrderRequest__phone_number", DELIVERY_ADDRESS = "_OrderRequest__delivery_address", ORDER_TYPE = "_OrderRequest__order_type", PRODUCT_ID = "_OrderRequest__product_id", and ID = "_OrderRequest__order_id". Another important aspect to note in regard to the keys is that these classes are Enum classes which allows them to be accessed throughout the project through the use of the .value method. This is because Enum classes are classes "which are a set of symbolic names bound to unique, constant values." The use for these classes is to promote the code's readability by taking out hard coded strings to access certain keys of the models classes.

6. Exceptions

The exceptions were very much implemented the same way as the keys were. The ExceptionMessages class is also an enum class with the purpose of creating symbolic names for each specific type of exception that would be thrown. Doing this, allowed us to streamline our exception implementation and allowed for our code to be significantly more readable. In addition to having the exception_messages.py file, the exceptions folder also has order_management_excpetion.py. We moved this file into the exceptions folder because it is the exception class used for most of our exceptions in this project. By moving order_management_excpetion.py to exceptions, our code became significantly more organized.

7. Singleton Pattern

The Singleton pattern is helpful when ensuring that a class only has one instance, such as when you want to ensure that your application only contains one instance of a database connection or logger object. For example, we used lazy initialization to create singleton patterns in the order manager and the order delivery, request, and shipping store. We had to add these to the stores because as stated previously the store contains the functions that need to be accessed privately. The Singleton pattern in our code is able to test the code to make sure the code is consistent and able to test it well. We have to add it to the stores because those hold most of the functions. However, since the Singleton instance is a global state that any portion of the program can access. The Singleton pattern may make testing more challenging. Therefore, the Singleton class' design needs to be well thought out to guarantee that it can be tested and does not create unexpected dependencies. There were a lot of problems with the imports to concisely be able to test so it took a lot of testing to make sure that the imports were formatted correctly.

8. Articles Summaries

Megana

Article 1

Title: A longitudinal study of the impact of refactoring in android application
Place of Publication: Information and Software Technology Journal
URL: https://www.sciencedirect.com/science/article/pii/S0950584921001531
Authors: Oumayma Hamdi, Ali Ouni, Mel Ó Cinnéide, Mohamed Wiem Mkaouer
Date of Publication: December 2021

The purpose of this article was to investigate the impact that code refactoring has on android applications. More specifically, the article investigates how code refactoring impacts how an android application is maintained and how it evolves over time. They decided to focus on Android applications because over 85% of current smartphones are Androids. To properly research this question, the researchers conducted a longitudinal study. They analyzed 5 Android apps and their evolution over 3 years. Combined, these applications had 9600 refactoring operations. In addition, the researchers considered 15 code smells (weaknesses in the design of the code that could cause deep issues). The study ended up finding that more often than not even though code smells are common and widespread in Android applications, Android classes with code smells were not often targeted by refactoring attempts. Due to that, almost rarely did a refactoring operation remove a code smell. Based on these results, the researchers concluded that even though code refactoring can gave a positive impact on code quality through the reduction of some code smells as well as code attributes, reusability, testability, adaptability, maintainability, and understandability, code refactoring does not fix all code quality issues. In fact, the effectiveness of code refactoring on android applications depends on the complexity of the existing code as well as the skill and experience of the developers who are doing the code refactoring. This article is relevant to the subject of the course because it has to do with code refactoring. The study analyzes code refactoring which is what this exercise is about. In addition, we are able to see the direct impact that code refactoring has on Android applications: something we interact with every day.

Article 2

Title: Comparing Commit Messages and Source Code Metrics for the Prediction Refactoring Activities
Place of Publication: Algorithms Journal

URL: https://www.mdpi.com/1999-4893/14/10/289
Authors: Priyadarshni Suresh Sagar, Eman Abdulah AlOmar, Mohamed Wiem Mkaouer, Ali Ouni, and Christian D. Newman
Date of Publication: July, 13, 2023

The purpose of this article was to understand how and if code metrics are able to be helpful indicators for predicting the refactoring that occurs in code. This was mainly done through assessing the commit messages that programers used when submitting source code. In order to carry out this study, the researchers conducted a design that used an already existing database of over 800 Java projects. Combined, these programs had over 5004 commits that the researchers were able to analyze. The researchers created machine learning models in order to predict the correct type of refactoring that could be found. The refactoring categories that they used were the following: extract method ("creating a new method by extracting a selection of code from inside the body of an existing method"), inline method ("Replacing calls and usages of a method with its body and potentially removing its declaration"), move method ("Changing the declaration of a method from one class to another class"), pull-up method ("Moving up a method in the inheritance chain from a child class to a parent class"), push-down method ("Moving down a method in the inheritance chain from a parent class to a child class"), and rename method ("Changing the name of a method identifier to a different one"). Through this model, the research team was able to come to the conclusion that the detection of refactoring types varied based on the type of refactoring that it was. Regardless, the overall conclusion found that the accuracy of the models was 75%. This study is relevant to the class because it once again has to do with code refactoring. In addition, the study uses commit messages in their data model; this is extremely relevant to this guided exercise because as a group we are constantly committing our refactored code.

Sarah

Article 1

Title: A Tutoring System to Learn Code Refactoring
Place of Publication: ACM Digital Libraries
URL: https://dl.acm.org/doi/pdf/10.1145/3408877.3432526
Authors: Hieke Keuning, Bastiaan Heeren, Johan Jeuring
Date of Publication: March 2021

The article discusses the importance of teaching code quality and refactoring strategies to novice programmers. While functional correctness of code has been the main focus of studies on student programming, there can be many functionally correct solutions to the same programming exercise. Poor coding style and poor quality can lead to incomprehensible code with low maintainability and testability, which is an issue for all developers. The article proposes a tutoring system that complements human tutoring by providing hints and feedback on exercises to help students improve their code. The system is designed to teach students about code quality in the context of small programs. The article then goes into the specifics of the program's design and specifications. Various sources from the literature and best practices from software engineering are used to implement the refactoring "rules", which are divided into teacher hints and steps, semantic style indicators, professional tools, and arithmetic and logic rules.

Code functionality has always been the most important piece of my software engineering education. While I have been taught how to program using good style, I think this assignment is a perfect example of why this article is so important. The article talks about how to best teach refactoring to students, specifically intertwined with teaching them the importance of functionality. Obviously we are learning refactoring after implementation, which is harder than beginning with designing clear, readable, and well-organized code.

Article 2

Title: Energy Efficiency Analysis of Code Refactoring Techniques
for Green and Sustainable Software in Portable Devices
Place of Publication: Electronics Journal
URL: file:///Users/sarahhoover/Downloads/electronics-11-00442-v2.pdf
Authors: Ibrahim Şanlıalp, Muhammed Maruf Öztürk, and Tuncay Yigit 2
Date of Publication: 1 February 2022

The article discusses the importance of reducing energy consumption in software development, particularly in portable devices, and introduces the concept of green software engineering. The authors focus on the effectiveness of refactoring techniques in reducing energy consumption. Several previous studies have investigated the relationship between code refactoring techniques and energy consumption in electronic devices, particularly Android mobile devices. Refactoring techniques have been found to have a positive effect on energy consumption, and studies have suggested that code refactoring is a good step towards energy efficiency. However, the previous studies have limitations, which this study aims to fix.

The article discusses a study that investigates the energy efficiency of code refactoring techniques in open-source object-oriented software applications. The study applies five refactoring techniques to source codes written in Java and C# and prepares a triple combination list of the selected refactoring techniques. After constructing all the combinations, they are integrated into the source codes, and energy consumption estimates are made for the original and refactored codes. The study found that the energy consumption of the refactored code was significantly lower than the energy consumption of the original code, suggesting that code maintainability can have a significant impact on energy consumption.
This article is relevant to our class and guided exercise because while our goal was not to use less energy in our design and implementation, reducing energy consumption is always a good thing in software design. Using less energy also most likely increases operational speed, which is another plus which directly affects our program: that it will run faster when handling orders.

Krislyn

Article 1

Title: Refactoring Support Based on Code Clone Analysis
URL: https://link.springer.com/chapter/10.1007/978-3-540-24659-6_16
Authors: Yoshiki Higo, Toshihiro Kamiya, Shinji Kusumoto, Katsuro Inoue
Date of Publication: 2004

Research paper "Refactoring Support Based on Code Clone Analysis" by Yoshiki Higo. The method for aiding refactoring through code clone analysis is presented in the study. The suggested approach finds code clones in the source code, sorts them into various kinds, and then makes refactoring suggestions based on the found clones.The concept of code clones and their significance in software engineering are introduced at the essay's outset. The author then covers the limitations of current code clone detection techniques. Unfortunately, these tools frequently do not offer any suggestions for refactoring .Clone identification, clone categorization, and refactoring recommendation are the three steps used by the suggested solution. A clone detection tool is used in the initial step to evaluate the source code and find code clones. Then, the detected clones are divided into various types in the second stage based on attributes like the degree of resemblance, the size of the clone, and the type of code. Finally, based on the discovered clones and their categories, the program offers refactoring recommendations in the third step. In order to assess the efficacy of the suggested strategy, the paper also offers the findings of a case study performed on three open-source projects. The findings demonstrate that the approach can successfully identify code clones and offer helpful reworking suggestions. The method for assisting refactoring using code clone analysis is suggested in the paper's conclusion, and it can help enhance the quality and maintainability of software systems. The technique can assist programmers in finding duplicate lines of code and offering suggestions for restructuring, which will enhance software design and lower maintenance costs. Because code clones are a widespread problem in software systems and can negatively affect software quality, maintainability, and evolution, this work is pertinent to software development. In a codebase, code clones are identical or similar sections of code that appear elsewhere. They might lead to redundant work, consistency, and bugs. The paper suggests a technique for locating and classifying code clones and offers refactoring suggestions based on the found clones. This technique can enhance software quality and maintainability by decreasing code duplication, enhancing consistency, and simplifying code structure. Refactoring is crucial in software development because it allows engineers to enhance the code's layout, organization, and readability. However, refactoring can be laborious, prone to mistakes, and requires thorough source knowledge. The suggested approach can help developers refactor by making suggestions based on code clone analysis, which lowers the time and risk involved with refactoring. Overall, this study is pertinent to software development since it addresses a widespread problem and suggests an approach for refactoring and code clone analysis to enhance the quality and maintainability of software.

Article 2

Title: A Field Study of Refactoring Challenges and Benefits
URL: https://dl.acm.org/doi/pdf/10.1145/2393596.2393655
Authors: Miryung Kim, Thomas Zimmerman, Nachiappan Naggapan
Date of Publication: N/A

The research paper "A Field Study of Refactoring Challenges and Benefits" was written by Miryung Kim and others. The report summarizes the results of a field investigation into the advantages and difficulties of refactoring in software development. The concept of refactoring and its significance in software development are introduced at the paper's outset. Refactoring is enhancing the layout and organization of code without altering its function. Refactoring can enhance the readability, maintainability, and quality of the code. Nine software development teams from various industries participated in the field study, including online, mobile, and game development. The teams were questioned to comprehend the difficulties and advantages of refactoring, and their code repositories were examined. According to the survey, the primary obstacles to refactoring are the expense and work involved, the possibility of creating defects, and a need for more time and resources. However, the study also discovered that refactoring improved code quality decreased technical debt, and increased the code's maintainability and scalability. The article also lists some of the tactics and procedures employed by the teams to get around refactoring's difficulties. These tactics consist of ongoing refactoring, prioritizing refactoring jobs, utilizing automated tools, and integrating team members in the refactoring procedure. The relevance of refactoring in software development and its difficulties and advantages are highlighted in the paper's conclusion. Finally, the paper offers insights into the tactics and procedures employed by software development teams to get over refactoring's drawbacks and gain from it. Overall, this work is pertinent to software development since it emphasizes the value of refactoring and offers insights into its difficulties and advantages. Furthermore, the study's conclusions can assist software development teams in enhancing their methods for creating high-quality, scalable, and maintainable software.

9. Conclusion

In conclusion, the main purpose of this guided exercise was to learn how to refactor and why it is important to refactor. In addition, the singleton pattern was also introduced to us for this exercise. This assignment showed us that refactoring is more than just changing the names of variables. This is why to refactor GE4 we had to reorganize the attributes, the stores, the keys, the models, the exceptions, and implement the singleton pattern. This project has made us understand the importance of code design and how it is just as important (if not more important) than code functionality.