Software development using design patterns

MSc. Aurela KALLJA[1]
Advisor: Prof. Dr. Petraq Papajorgji

Abstract

Information systems have become an integral part of our lives. The demands for software that help us accomplish our daily tasks are ever increasing, considering the great technological momentum around the globe. Software engineering is the process of analyzing user requirements, designing and developing software applications. Each user request is a problem that an individual or a business has encountered in the daily work processes.

The goal of software engineering is to provide an optimal and efficient solution to these problems to increase the overall productivity of employees in the respective industries and at the end of the day, to increase profits. Providing these solutions is no easy feat, as the problems are often complex and in addition to requiring careful analysis, they also need smart solutions. Fortunately, we have the ability to learn from experience and apply our knowledge in different contexts to achieve our goals. As in any field of life, problems have a recursive nature and it often happens that the same problem is encountered in different contexts. Naturally, we can think that similar problems have similar solutions. The set of solutions to general software design problems in a specific context constitutes what are called design patterns. Design patterns are structures of how some objects dialogue with each other to provide a specific solution to a problem. They are ready-made solutions to known problems,

[1] Aurela Kallja has a background in both economics and computer science. In 2020, she is graduated in "Business Informatics" at the European University of Tirana. In 2022, she is graduated from the European University of Tirana with a Master of Science degree in "Software Engineering", profile "Data management". During her studies, Aurela demonstrated a passion for technology and a strong drive to learn, as evidenced by her completion of a "Java" course at SDA, where she acquired valuable programming skills. Currently, she is working online as a QA testing engineer.

and the real challenge with them is not in their construction, but in the intuitive ability to associate a design problem with the corresponding pattern that offers the most optimal solution. This paper will deal with the development of an information system for an ATM Exchange system, which carries the functionalities of an ATM cash machine and exchange rate chart. The development of this software will be totally based on design patterns, more specifically "Observer" pattern and "Chain of responsibility" pattern. This paper aims to emphasize the advantages of using design patterns and highlight the potential of them to solve general problems in a specific context. It also focuses on software engineers, information systems developers and software engineering students. This paper will serve as a manual of best software development practices, emphasize the principles of flexibility and reusability of information systems development components.

Key words: *ATM, design patterns, exchange rate, software engineering, system, OOP.*

Introduction

Information systems have significantly improved the productivity of human activity from the moment of their appearance until today, when it is difficult to imagine any area of life without software systems that make everything easier. Nowadays, software systems have been perfected so much that with their help, processes that were impossible to carry out a few decades ago, nowadays can easily be carried out. All this thanks to the improvement of software development techniques, driven by the need to provide solutions for problems of complex natures because, the more complex the problem, the more complex is the provision of adequate solutions. As each problem is specific in its nature and requires a specific software solution in order for the information system to meet expectations.

The science that deals with the study and analysis of requirements for a certain software development problem, the design and development of solutions is known as software engineering. In software engineering "Design patterns are descriptions of communicating objects and classes that are customized to solve a general design problem in a specific context." (Gamma, Helm, Johnson, & Vlissides, 1994).

Design patterns enable us to solve complex problems in a fast and efficient way with the guarantee that the solution has been tested and proven thousands of times throughout the history of software development and has proven to be efficient.

> "Each pattern describes a problem that occurs repeatedly in our environment, and then describes the essence of the solution to this problem, in such a way that this

solution can be used a million times, without ever doing it the same way twice." (Alexander, 1977)

A design pattern names, abstracts, and identifies key aspects of a common design structure, making it useful for creating reusable object-oriented designs. These models identify the participating classes and instances, the role and relationships between them, and the distribution of responsibilities. In this paper, in addition to the theoretical elaboration of how design patterns work and how developers benefit from them, an information system based on design patterns, specifically "The Observer" pattern and "The Chain of responsibility" pattern, will be developed.

1.1 Purpose

Some of the goals of this paper are:

- Introducing an information system based on the principles of object-oriented programming. More specifically, the aim is to build an ATM cash machine system, using "The Chain of responsibility" pattern, and "Observer" design patterns.
- To represent the facts which state that design patterns are the optimal solution for the problems that are often encountered in an information system and to implementation them correctly.
- By enabling an efficient interaction for the user, the ATM and the system administrator, I will aim to develop a service that is characterized by: transparency, security and simplicity of use of the solution provided.

1.2 Objectives

The objectives that I intend to reach at the end of this paper are:

- Creating a model through the application of a set of explanatory and heuristic principles.
- Understanding a problem in details in terms of its processes and concepts, through the study and application of the information and techniques presented.
- Creation of an information system using design patterns, such as Observer and Chain of Responsibility patterns.
- Designing a solid solution through the use of Object-Oriented Programming principles.

1.3 Methodology and results

The software industry is a fast-paced field and innovation is at a crazy pace compared to other industries. As in any other industry, the key to success is the ability to innovate before anyone else. Based on these indications, we assume that a commercial bank intends to launch ATMs that, in addition to playing the role of cashier, also provide information and functionality related to the exchange rate. Such a system, which is offered for the first time, should be able to provide citizens with real-time teller services that include managing their money, withdrawing money at any time as well as information on the exchange rate in every moment. The system must be flexible, update data in real time with exchange rate changes in international exchanges market, as well as support the removal or addition of new currencies, the creation and updating of accounts in different currencies, and updating the current status of ATMs.

Elaboration

2.1 Development of the model by means of UML

Based on the requirements, the system will be developed using abstraction to define the concepts that participate in the ATM Exchange system. The design of UML software model has been carried out too. The purpose of using the model is to get a good understanding of the system that will be built. The three models that were used to design the ATM Exchange software are Use case diagram, Class diagram and Sequence diagram. Model development, in addition to creating a visualization of the developing system, helps us generate a significant amount of code correctly.

2.2 Use Case Diagram

To create a software product, the developer must know how the user will interact with the application. This is where we need the Use Case diagram. A Use Case Diagram summarizes the details of the users and their interaction with the system. It specifies the expected behavior of the system (what?), rather than the exact method of achieving it (how?). A key concept of using these diagrams is that they help create a system from the end user's perspective. It is an effective technique for communicating system behavior in user terms by specifying all externally visible system behavior.

"Use Cases represent only the functional requirements of the system. Other requirements, such as business rules, quality of service requirements, and implementation constraints should be represented separately, with other UML diagrams." (What is Use Case Diagram?, 2022).

USE CASE DIAGRAM

The ATM Exchange software system contains two main actors: the user and the administrator. The user or rather the client has the functionality of logging into the system and then he can perform a series of operations. These operations include: checking the balance in real time, withdrawing money and obtaining information on the exchange rate. Another important actor is that of authentication. The authentication actor plays an important role in validating the credentials that are inserted into the system. This functionality serves both the client and the administrator and is a key factor that ensures the security of the information system.

The administrator is another actor, who has associated several use cases in the ATM Exchange system. The administrator can log into the system and after his data goes through the authentication process, he has a series of menus available, which include account management, ATM management and exchange rate management.

Account management involves adding new accounts to the system for new customers, updating account information of existing customers, and finally deleting the accounts of customers who are no longer served.

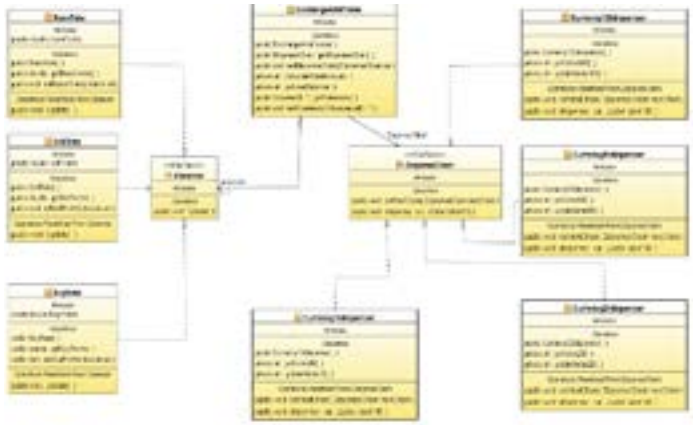
ATM management includes adding new currencies, updating existing currencies and deleting currencies that are no longer serviced by the system. Another use case of the administrator, which contains several sub-cases, is currency management. This use case includes adding a currency rate, updating a currency rate, and deleting a currency rate.

2.3 Class Diagram

A class diagram is used to address the static view of a system, the relationships between classes, and how concepts relate to each other to form the whole, which is the system being built.

> "In software engineering, a Class Diagram in UML represents a type of structural diagram that describes the structure of a system by showing the system's classes, their attributes, methods, and relationships between objects." (What is Class Diagram?, 2022)

CLASS DIAGRAM



There are eight classes and two interfaces that participate in the information system, which have been defined based on the analysis of abstraction. The main class in the UML diagram of the system, which plays the role of the main class in the information system, is the ExchangeAtmFrame class, which has navigable association with the two interfaces, the Observer and the DispenseChain. The Observer interface is implemented by three classes BaseRate, BuyRate and SellRate. In this class diagram we have two pure uses of the principle of polymorphism implemented through the use of interfaces, which is a very powerful feature of object-oriented programming and the Java language in particular. The three classes BaseRate, BuyRate and SellRate realize a polymorphic implementation of the Update() method defined in the interface. But polymorphism is not limited to the implementation of a single method. The DispenseChain interface is implemented by four classes that are Currency100dispenser, Currency50dispenser, Currency20dispenser, Currency10dispenser, which implement the two methods defined in the DispenseChain interface: setNextChain() and dispense().

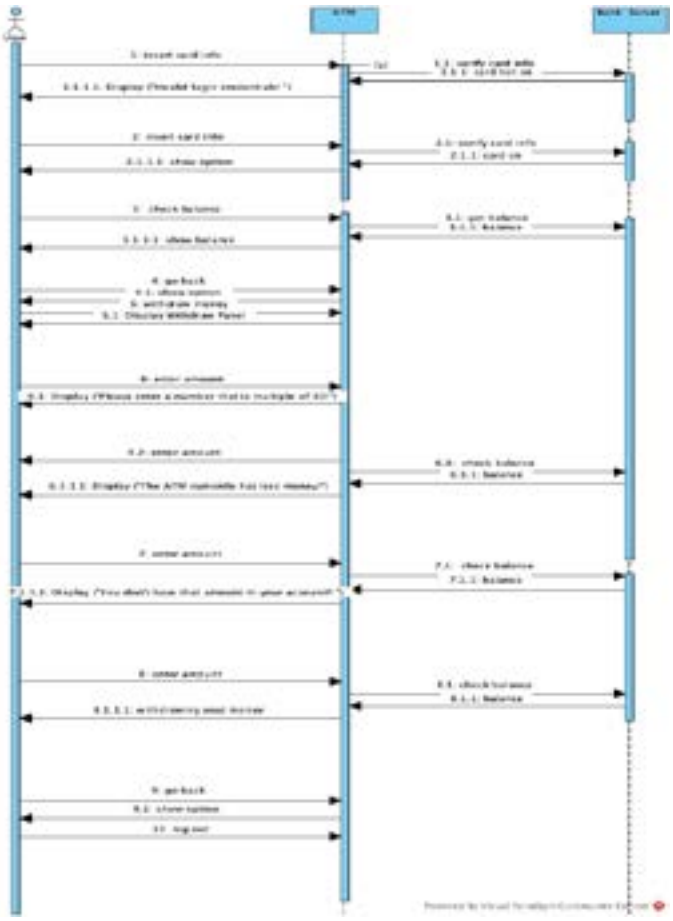
2.4 Sequence Diagram

The sequential diagram shows the execution of the system based on user cases, but in chronological order. In simple words, the sequence diagram represents user cases by the time they occur.

> "A sequence diagram represents the objects that participate in the interaction in a timely manner. The timing of when messages are sent to objects is important and changing this order can lead to unexpected results." (Petraq J. Papajorgji, Software Engineering Techniques Applied to Agricultural Systems, An Object-Oriented and UML Approach, 2014)

Figure below shows the sequence diagram of the ATM Exchange software.

SEQUENCE DIAGRAM



2.5 Design Patterns used in system development

The development of the ATM Exchange system is based on object-oriented programming technology. This object-based approach presents an information system as a community of objects that are in constant communication with each other with the goal of fulfilling the functions that the system has. The interaction between objects is realized by the behavior of each object and through the impact that the behavior of each object has on itself or on other objects, it affects the state of the object. The interaction of objects between each other adheres to a structure or a pattern. From here we come to the concept of design patterns, which are structures that represent the way objects interact with each other to provide solutions to a general problem in a specific context. So, if the information system is a bunch of objects that communicate together, a design pattern describes the flow of execution of this communication that coincides with the solution of a certain problem. In the development of the ATM Exchange system, two general design problems were identified, which were solved using the Chain of responsibility pattern and the Observer pattern. In the next subsection, it will be explained how these patterns are implemented in the system and why exactly these patterns were chosen.

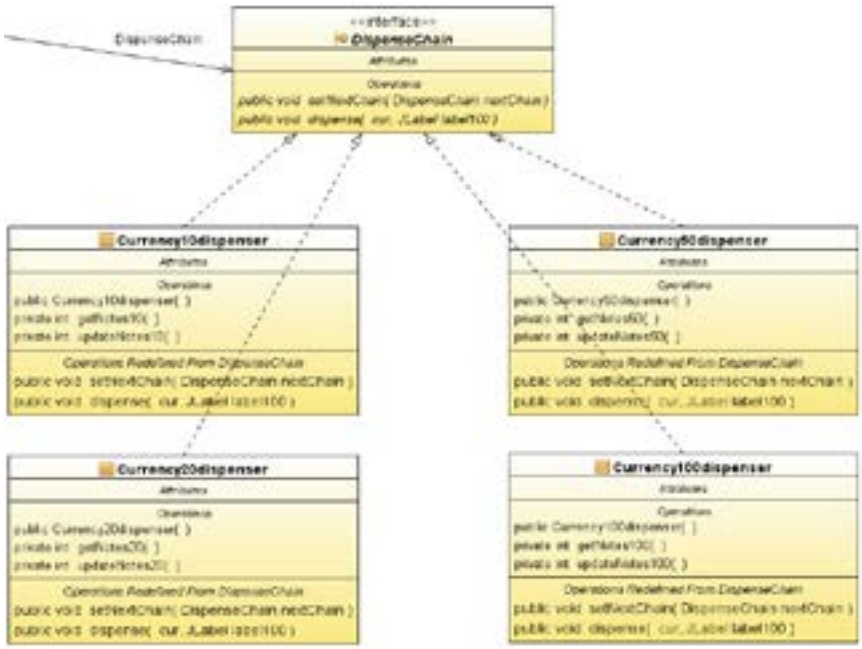
2.6 Chain of responsibility as a pattern for ATM functionalities

One of the functional requirements of the software we are developing is the functionality of an ATM cash machine. But what is the role of an ATM cash machine? Most of us, have had the opportunity to use an ATM to withdraw money or check the balance in our account. The balance check process is a simple read-only process. While ATM withdrawal is a functionality which enables the user to provide an amount to withdraw and withdraw it. But since it is an automatic service, there are some restrictions, the two main ones being that the amount must be a multiple of ten and the withdrawal is made in regular denominations that include tens, twenties, fifties and hundreds. The system calculates the amount and determines the deductions the user will receive. The selection of denominations is determined based on the amount given by the user and the availability of denominations in the ATM. This determination occurs through the design pattern of the chain of responsibility.

> "Chain of responsibility avoids piling up requests to one receiver by giving more than one object the opportunity to manage the request. It creates a chain between receiving objects and passes the request along the chain until an object handles it." (Erich Gamma, 1994).

Figure below shows the UML diagram of the "Chain of Responsibility" design pattern for the ATM Exchange system. This diagram consists of an interface that has two methods defined, one for passing request management from one object of the chain to another, and another that performs request management.

CLASS DIAGRAM OF THE CHAIN OF RESPONSIBILITY" PATTERN



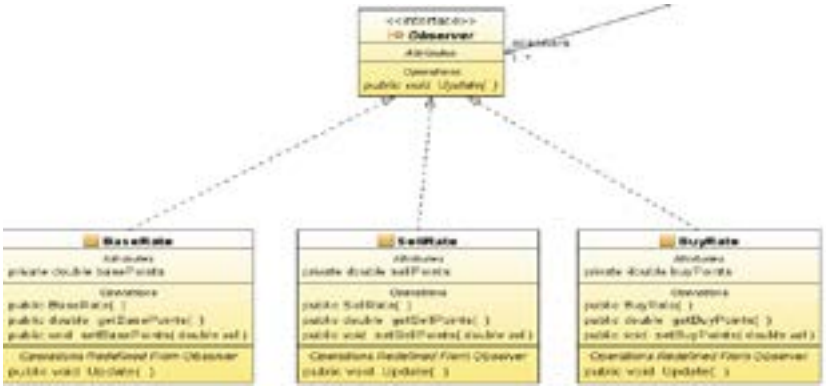
As shown in the figure, the chain consists of four objects, which are responsible for managing requests to the object chain. In this case we have a chain, which has one object for each denomination. Now, the user makes a withdrawal request with a certain amount, if the amount is above the value of 100, the chain starts from 100 and passes in turn to the objects of other values until the withdrawal amount is met and the chain ends.

2.7 Observer as a pattern for exchange rate charts

Another functionality of the ATM Exchange system is the real-time display of the exchange rate. The exchange rate represents the trading percentages between different currencies for the value of each currency to be translated from one type of currency to another. As we know, the exchange rate changes constantly even within a day and often even within an hour. Given that we use different objects to display the exchange rate, which in our system are responsible for displaying the graphs with the exchange rate data, whenever there is a change in the exchange rate, this

change must be reflected in each of the objects. If this process were to be done manually, it would take a lot of time and would not be done immediately, and this would conflict with the functional requirements of the system, which include real-time information about the foreign exchange rate. Fortunately, there is a general solution known as The Observer design pattern for this problem. "Observer pattern defines a one-to-many dependency relationship between objects, such that when an object changes state, all its dependents are automatically notified and updated." (Erich Gamma, 1994).

CLASS DIAGRAM OF THE OBSERVER PATTERN



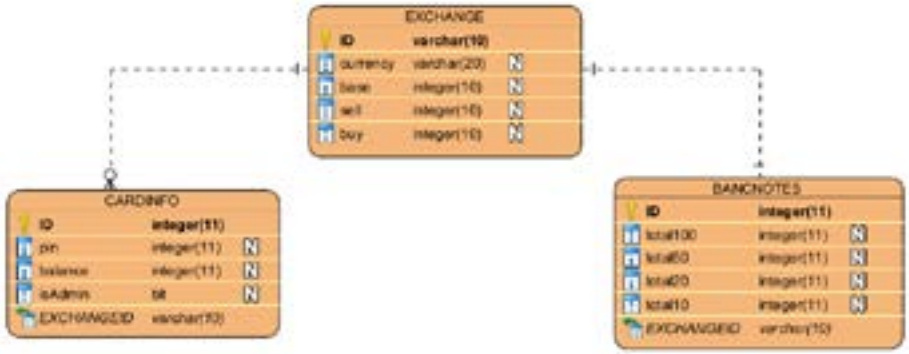
The figure above shows the Class diagram of the Observer pattern. This diagram contains an interface and three classes that implement the interface. The observer interface has a defined Update () method, which is declared in each of the classes that implement the interface, as the principle of polymorphism imposes. Each of these methods provides a different implementation of the same method for each of the exchange rate types, so we say that a polymorphic implementation of the Update () method is realized.

2.8 Entity relationship diagram

The database is the backbone of any information system. This is because the main function of a software is the storage and processing of data based on the requirements that users have. In the ATM Exchange prototype system, the database stores important information in terms of user data, exchange rate data and ATM status data. The entity relationship diagram of the ATM Exchange system has been built to visualize the database. This diagram consists of three entities which are "Cardinfo", "Exchange" and "Bancnotes". "Cardinfo" stores the attributes of card ID, primary key, pin, balance and "isAdmin or not" status. "Exchange" stores the data as the ID which is the primary key of the table, the currency type and the

three values of the exchange rate, the base rate, the selling rate and the buying rate. "Bancnotes" store the ID attribute values as the primary key, the total number of denomination of hundreds, fifties, twenties and tens.

ER DIAGRAM



As also shown in the ER diagram, "Exchange" with "Cardinfo" has a 1 x N relationship, because a currency can have many cards and a card belongs to only one currency type. On the other hand, the connection between "Exchange" and "Bancnotes" is a 1 x 1 connection since denominations in an ATM can only be in one currency and one currency has only one list of denominations. The ER diagram presented above only presents the basic functionalities of the system, since in this paper the goal is to explore the potential of providing solutions based on design patterns and not to design complex databases that store huge amounts of information.

2.9 Algorithm comments

After analyzing the functional requirements, defining the design patterns that would be needed to solve some of the key requirements of the system, and building the UML models of ATM Exchange, we are ready to develop the system. Since the developed system is a desktop application analogous to an ATM cash machine, the process of placing the card in the ATM is simulated by a login form.

The customer must have a personal account to access the system. The account is composed by the personal card number and the user's password. If any of the credentials are incorrect, the user is shown an error message asking them to enter the correct details. In this way, an increased security of the system is enabled.

After successfully logging into the account, the user is presented with a menu from which he can choose between two options: check the account balance or perform a currency withdrawal.

The Java login code has logic such that if a user with the given credentials exists in the database, in the Cardinfo table, a check is made to read the value of the isAdmin column for that record. If the isAdmin value is false, the user will be redirected to the client panel. In the other case if the user has the value "isAdmin true", it will be redirected to the administrator panel.

Selecting the balance check option enables the panel that presents information about the account status of the logged-in user. The information relates to the personal number of the card, the balance of the account and the corresponding currency.

The second option for withdrawing currencies, displays a panel where a simple form is presented in which the customer determines the amount he wants to withdraw from the ATM. On the other hand, three graphs show the exchange rate, named as: "The base rate", "The purchase rate", and "The sale rate" of different currencies.

The system is built based on some constraints, so that the model approximates the real model as much as possible. The amount that can be withdrawn from the ATM must be a multiple of ten. Also, the customer cannot withdraw an amount beyond the total that he has in his personal account. Another limitation is related to the total balance of the ATM. In case the customer requests to withdraw a larger amount compared to the ATM balance, there will be shown an informational message, just like in the two cases mentioned above.

Meanwhile, in case of success, the system prints a message showing to customer the banknote denominations of the completed transaction.

The second user entity is the administrator, who can also log into the system using personal credentials. His role is related to the management of the entire ATM system, where he can manage customer accounts, ATM card balances and exchange rates.

The administrator has the ability to manage the personal accounts of users, adding new customers, updating their data and deleting them from the system.

Since the fields are mandatory, if any of them is empty, an error message appears that all fields must be filled. Otherwise, it continues with the declaration of the query that inserts the data. Next, the method that makes the connection to the database is called. Communication with the database is carried out by means of a prepared statement, passing the values as parameters and not as values directly in the query. This is done for security reasons, since prepared statements are more protected against SQL injections and cyber-attacks.

In the ATM management option, the possible functions that he can perform are related to the maintenance of the state of the system in terms of its overall balance, through the addition, deletion and updating of banknotes that the customer can withdraw through his account.

Furthermore, exchange rate fluctuations, directly related to each client's personal accounts, are also managed by the admin. In this way, the interaction between the two actors is realized, enabling a simple system in use and maintenance.

Conclusions

Based on the theoretical analysis and the development of the information system designed using design patterns, the following conclusions were reached:

- Software development based on design patterns provides fast, well-proven and reliable solutions to complex development problems. Using them properly allow us to create powerful software systems that meet end-user requirements and have a well-defined architecture.
- System modeling, which includes UML, use cases and sequence diagrams, in addition to enabling software visualization, improves the development process by accurately and quickly generating a significant amount of code directly from the model.
- When talking about a successful software application, it is not only about how it performs the function for which it was developed, but also about the efforts that have been made for its development, testing and maintenance. If this is not done correctly, the cost of development will result in an app that no one wants.
- Design principles enable writing better code. Also following these principles helps create a clean and modular design that is easier to test, debug and maintain in the future. Furthermore, features should be implemented when necessary and code duplication should be avoided. It should not be forgotten that a clean code is easier to understand and certainly saves time at the time of changes or new implementations.
- Object-oriented programming is a powerful and natural paradigm for creating programs that survive the inevitable changes that accompany the life cycle of any large software project.
- In the future, I would suggest raising the awareness of the developer community about the importance of studying and implementing design patterns as among the best development practices.
- Trends in banking and foreign exchange developments will continue to evolve as long as technology advances. Artificial intelligence will likely continue to advance data analysis and improve the way these market players interact with each other.

Bibliography

1.Alexander, C. (1977). *A Pattern Language.* Oxfordd University Press.

What is Class Diagram? (2022). Retrieved from visual-paradigm: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-class-diagram/

2. Erich Gamma, R. H. (1994). *Design Patterns: Elements of Reusable Object Oriented Software.* Addison-Wesley Professional.

3. Papajorgji, P. J. (2014). *Software Engineering Techniques Applied to Agricultural Systems, An Object-Oriented and UML Approach.* Springer.

3. What is Use Case Diagram? (2022). Retrieved from visual-paradigm: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-use-case-diagram/