ISSN: 3048-5320 (Online)

CSIBER International Journal - CIJ

Vol 1, Issue 1, Oct - 2023

KOLHAPUR

MULTIDISCIPLIANRY JOURNAL

Published By
CSIBER Press, Central Library Building
CSIBER Campus, University Road,Kolhapur-416004, Maharashtra State, INDIA.

Find the Journal Online @
www.siberindia.edu.in

CSIBER International Journal – CIJ

CONTENT

Realizing the Use Cases through Classes, Objects and Their Relationship

Dr. Ajay D. Shinde
Associate Professor,
Chhatrapati Shahu Institute of Business Educations and Research, Kolhapur,
ajshinde@siberindia.edu.in

Abstract: Since its inception, the Unified Modelling Language (UML) has proven to be a valuable tool for software developers in modelling the system at hand. The use case diagram serves to provide developers with an understanding of the functions that external users expect from the system. On the other hand, the class diagram focuses on how these expected functions can be implemented in the system through the use of classes, attributes, and operations. This necessitates that the classes interact and collaborate with one another in order to fulfil these functions. The class diagram visually represents the relationships between classes and objects. It is important to note that the class diagram is reliant on the accuracy and comprehensiveness of the use case diagram, as any errors or omissions in the latter will have significant implications for the former. This paper aims to explore the relationship between these two crucial diagrams.

Keywords: Unified Modelling Language, Use Case Model, Class Diagram, Requirement modelling, Data model.

I. INTRODUCTION:

The software development process commences with requirement analysis, during which software developers strive to comprehend the desired functions and features that end users expect from the system. The objective is to enhance the user's daily work by providing them with a system that is more efficient and effective. Subsequently, requirement modelling is undertaken, involving the creation of various diagrams to gain insight into the functioning of the existing system in terms of inputs, processes, outputs, data storage, and data flow. Requirement engineering is a challenging task due to the fact that requirements exist within the problem domain, while software objects exist within the solution space (Betty H.C. et al 2007). As software is applicable across various domains and its complexity continues to grow, the gathering, analysis, and modelling of requirements becomes increasingly arduous (Mona Batra et al 2020). Consequently, many organizations are placing greater emphasis on requirement engineering, recognizing that the quality of the software is contingent upon the clarity, completeness, and consistency of the requirements (Kanishka Gopal et al 2016). This underscores the significance of thorough and comprehensive requirement analysis within the software development process.

Requirement Engineering encompasses the process of requirement modelling, in which software developers utilize specific methods to visually represent user requirements, typically through the use of diagrams and images. Diagrams, such as Data Flow Diagrams (DFD) and Use Case Diagrams, are the preferred means of modelling requirements due to their enhanced communicative capabilities compared to textual representations. The primary focus during requirement modelling is on determining what needs to be accomplished, rather than how it should be achieved. In the subsequent design phase, the emphasis shifts towards determining how user requirements can be translated into software components and architecture.

For object-oriented systems, the fundamental building block is the class diagram, which illustrates the classes necessary to fulfil user requirements and the relationships between them. This is accomplished by specifying attributes and methods for each class. Consequently, each requirement can be traced back to the corresponding classes, attributes, and methods. It is important to note that a single class may not be sufficient to implement a user requirement;

multiple classes may be required. In order to fulfil the requirements, these classes must interact and collaborate, enabling the system to deliver the desired functionality to the user.

II. RESEARCH PROBLEM:

Each phase within the software development life cycle relies on the preceding phase, as the output of the preceding phase serves as input for the subsequent phase. This demonstrates the interdependence of phases within the software development life cycle, indicating that they cannot be viewed in isolation. The work product of each phase is directly linked to the work product of the following phase. In the context of object-oriented software engineering, the requirement analysis phase yields a use case diagram, which is utilized to define the relationship between system requirements and users. These requirements must then be mapped to classes, attributes, and methods to ensure that the final system fulfils the necessary functionality for the user. The purpose of this paper is to comprehend, establish, and substantiate the correlation between use case diagrams, which are the output of the requirement analysis phase, and class diagrams, which are the output of the design phase. The research conducted by Kmalrudin concludes that the most crucial diagram or modeling tool in object-oriented programming is the use case diagram, particularly in the context of requirements validation (M. Kamalrudin and others 2015).

III. REQUIREMENT ENGINEERING AND DESIGN:

The software's quality is contingent upon its ability to meet the user's requirements. This underscores the significance of the requirement engineering process in software development. It is imperative that the requirement engineering process effectively gathers and documents all user requirements and desired functionalities. These requirements can be categorized as either functional or non-functional, with functional requirements being implemented as system functions and non-functional requirements encompassing features expected by the user.

In order to gather the necessary requirements, developers employ various tools and techniques known as fact-finding techniques, which include information gathering and integration techniques(M. Christel et al 1992). The requirement engineering process involves selecting from a collection of proposed requirements, prioritizing them, determining system boundaries, resolving conflicts, establishing objective acceptance criteria, and so on (Betty H.C. et al 2007). (Gareth Rogers 2016) has identified eight common requirement problems: missing requirements, hidden stakeholder needs, inadequate or ambiguous requirements, conflicting requirements, lack of testability and measurability, challenges in communicating requirements, changing requirements, and over-specification. These problems make requirement engineering challenging for developers.

In their paper (Geshwaree Huzooree and others 2015), emphasize the negative effects of poor and incomplete requirement engineering. Software development often encounters problems such as schedule delays, cost overruns, low customer satisfaction, failure to meet expectations, errors leading to poor quality deliverables, and increased maintenance costs due to rework, all of which can be attributed to improperly defined requirements. This discussion underscores the importance of the requirement engineering process, as it is the foundation upon which all other activities in software development depend.

Following requirement engineering, the design phase involves transforming the requirements into actual classes and establishing relationships between them. It is often necessary to have interaction and collaboration between classes in order to bring the requirements to fruition. This is achieved through the creation of a class model, which allows developers to consider and design the classes needed to implement the desired functionality. In his thesis (Dong Liu 2004) , the author proposes a method for semi-automating the conversion of functional requirements into a class model. The process begins with the use case diagram, which provides insight into

 ISSN: 3048-5320 (Online)

the functional requirements that developers must consider when identifying the necessary classes for implementation.

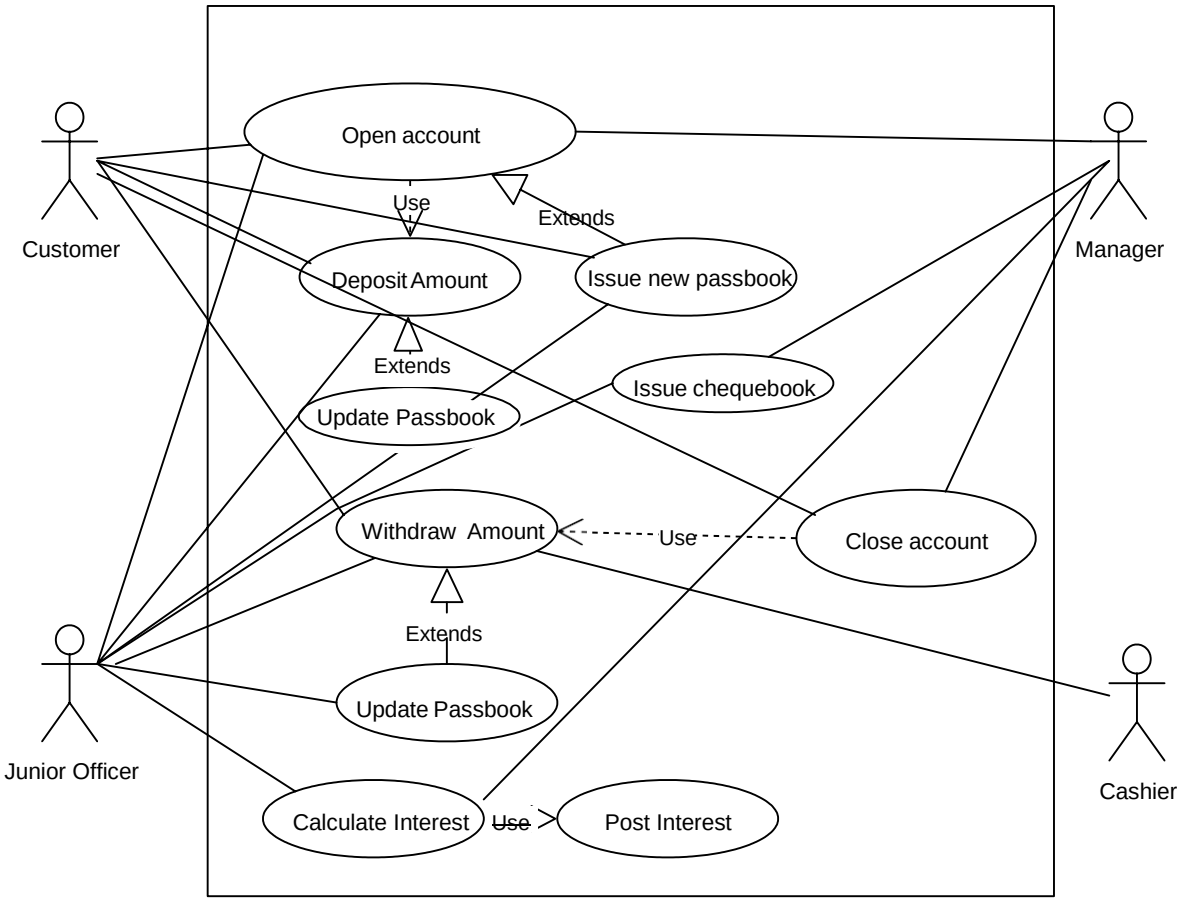
The classes serve as fundamental components in an object-oriented system, however, a single class may not be adequate for implementing the necessary functionality. It is essential for the classes within the system to interact and cooperate with one another in order to execute the desired functionality. This necessitates the establishment of relationships between the classes, which can be visually represented through a class diagram. While the use case diagram is a result of requirement engineering, the class diagram is a product of the design process.

IV. USE CASE DIAGRAM

We present a methodology for implementing use cases using a class diagram. To illustrate the process, we will consider the design and development of a savings bank account system. The actors involved in this system include the customer, manager, cashier, and junior officer. The desired functionalities for these users encompass opening an account, issuing a passbook and chequebook, withdrawing and depositing funds, calculating and posting interest, updating the passbook, and closing an account. By utilizing the gathered information, we can construct a use case diagram that visually represents the actors and their respective functions of interest.

Figure 1 : Use case diagram for Savings Bank Account System

The use case diagram depicted in Figure 1 illustrates the functionalities expected by the actors within the savings bank account system. Each use case represents a specific function within the



proposed system. It is important to note that the implementation of these functions necessitates the utilization of multiple classes and objects, as a single class alone is insufficient to achieve complete functionality. The successful realization of these interactions and collaborations

between classes and objects can be achieved through the utilization of class diagrams, as well as interaction diagrams such as sequence and collaboration diagrams.

The use case relationships and descriptions play a crucial role in understanding and documenting the functionality of a system. The use case diagram, which depicts the relationships between different use cases, provides a high-level overview of the system's functionality (Sa'adillah & Ali, 2022). However, it does not comprehensively understand system functions and the specific interactions between the different use cases. To address this limitation, it is necessary to provide detailed use case descriptions that provide a more in-depth explanation of each use case. These descriptions follow a template provided by the Unified Modeling Language and serve as a comprehensive guide to understanding the behaviour and interactions within the system. This approach is supported by various authors and studies. For instance, the Use Process approach combines Business Process Modelling Notation and UML Use Case Diagrams to model requirements (Bochicchio et al., 2012). This approach emphasizes the involvement of customers in the requirements definition process, leading to a successful project outcome. Furthermore, the use case descriptions also serve as a bridge between the use case diagram and the actual implementation of the system. They provide a clear understanding of the specific functionality, actors involved, and the relationships between use cases. The use case relationships, such as "use" and "extend," indicate the nature of the interactions between different use cases.

V. USE CASE DESCRIPTION

The use case description has five sections objectives, initiation of use case, flow of messages, alternative flow of messages, special/supplementary requirements, and how the use case finishes the functionHans-Erik Eriksson et al. The use case description has to be written for each use identified in use case modeling, but for this paper, we will consider only one use case description.

Let us consider a use case “Withdrawal of amount”

A. Objectives of use case:
   i) To record transactions permanently.
   ii) To update the balance in the account.
   iii) To make payment to the customer.

B. How the use case is initiated:
   The customer submits a filled withdrawal slip to a junior officer in the bank.

C. The flow of messages:
   The customer submits a filled withdrawal slip to the junior officer.
   Junior officer verifies and validates information on withdrawal slips.
   The junior officer checks the balance in the account of the customer.
   Junior officer records withdrawal transactions in books of account.
   The junior officer updates the balance in the account.
   The junior officer issues a token to the customer by writing the token number on the withdrawal slip.
   Junior officer records transaction number on withdrawal slip.
   The slip is sent to the manager.
   The manager verifies and validates transactions.
   The manager signs the slip and sends it to the cashier.
   The cashier announces the token number from the slip.
   The customer hands over the token to the cashier.
   The cashier verifies the token and counts currency notes to match the amount.
   The cashier records currency details on the backside of the slip.
   The cashier makes payment to the customer.

 ISSN: 3048-5320 (Online)

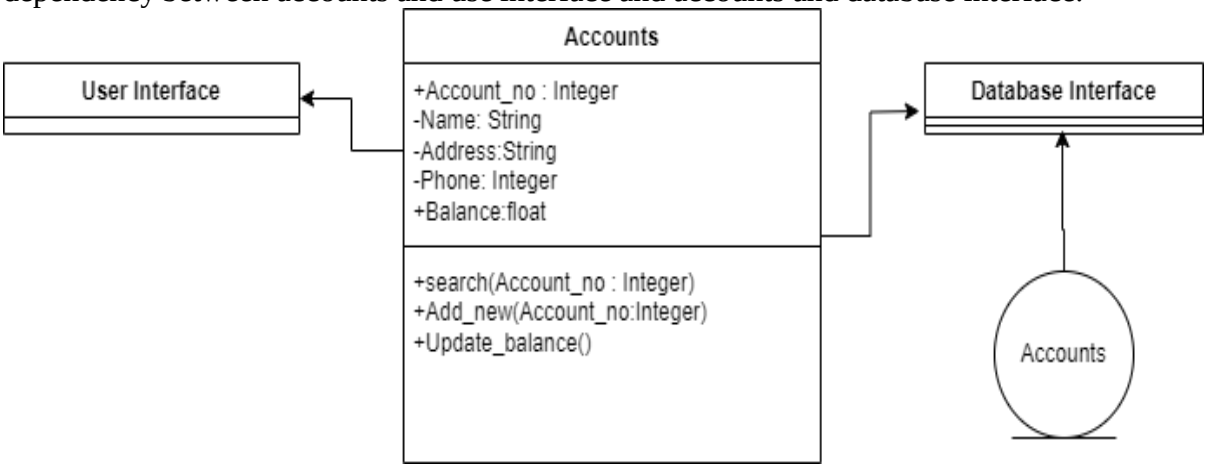
D. Alternative flow of messages:
i. If the account number written on the withdrawal slip is incorrect
Inform the customer to correct it.
ii. If the amount written on the withdrawal slip is greater than the balance in the account + minimum balance
Inform customer of “Insufficient balance”.

E. How use case finishes:
The use case finishes by recording the transaction in the books of account, updating the balance in the account, and making payment to the customer.

Together the use case diagram and use case description make the use case model complete and clear. This will provide all stakeholders with the information about who are the users of the system? What functions are they expecting from the system? How use cases are related to each other? How the work is carried out in the system?. The information obtained here is critical for software development process and it will initiate realization of use cases in to classes, objects and relationship between them.

VI. REALIZING USE CASES

In order to create a class diagram, it is necessary to carefully select each use case and use case description to determine the corresponding classes needed for implementation. We will begin the process of realizing use cases with the help of classes objects and relationship between them. Where each class and object can be divided in to three compartments the first compartment is used to name the class appropriately, the second compartment is used to define attributes of the class and the third compartment is used to declare methods to be implemented by the class.
Let us begin the process of use case realization by selecting "open account" use case depicted in figure 1. To implement this use case, two classes need to be designed: an interface class and a customer class, where interface class will implement user interface with the system that will enable the user to interact with system and provide necessary input. The customer class will define attributes of customer that are necessary for the function. Also the methods should be identified for the class that are must for carrying out the required function in the system.
Additionally, a decision must be made regarding the handling of database utilities. Should a common class be created to manage the interface between the database management system and the application classes, or should the code be written within each class? If the decision is to create a common class, three classes will be required to implement the "open account" use case. As the data required by accounts class is accepted and provided by use interface class and to store the date accounts class takes the help from database interface class the relationship is dependency between accounts and use interface and accounts and database interface.



ISSN: 3048-5320 (Online)

Figure 2 : Class diagram Showing Classes and relationship between them to successfully carry out open account use case.

Similarly, for the "withdraw amount" use case, an interface class must be designed to accept details about the withdrawal operation. Furthermore, when accepting the details of the withdrawal transaction, the account number of the customer needs to be validated, which is an attribute of the account class. This implies that the account class must implement search operation by accepting account number and if the account number is found it must return the account details required. If the account number is not present the account class must display a message on the interface indicating it is not found. Additionally, a table has to be designed in the database to record the data permanently. This makes a compulsion on designer that in order to handle the data a class has to be designed that is responsible for storing and retrieving data to and from the table. The scenario depicted here can be represented using a class diagram shown in fig. 3. Fig. 3 shows classes and relationship between then in order to implement withdraw amount use case successfully. The transaction class can be designed as a base class and withdrawal and deposit classes can be inherited from transaction class. Between accounts and transaction class the relationship is association. If a class is designed for implementing common database utilities the account and transaction classes will be dependent on this class for storage and retrieval of data from the database.

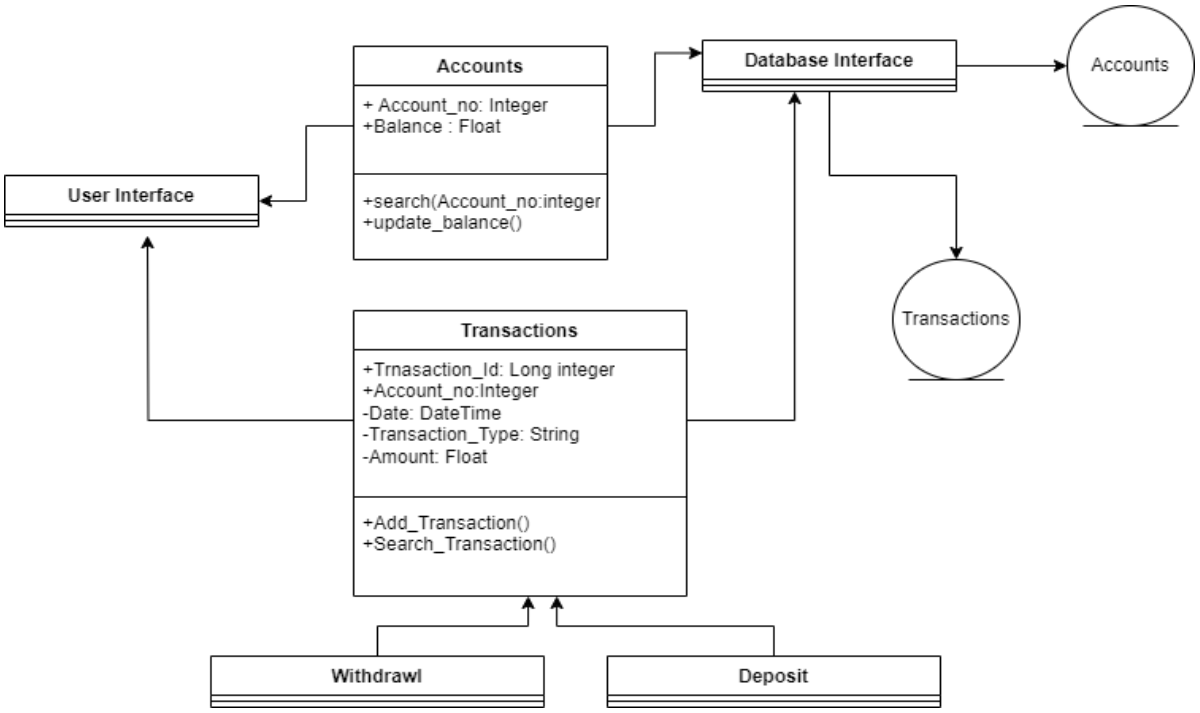


Figure 3 : Class diagram Showing Classes and relationship between them to carry out Withdraw amount and Deposit Amount use case.

The successful execution of the "deposit amount" use case necessitates the retrieval of information from the customer table through the customer class, as well as the insertion of a new transaction into the transaction table using the transaction class. To facilitate this use case, an interface class must be designed, which will accept the necessary information from the user. Furthermore, it will require an interface with the database management system. The relationship is association between accounts and transaction class and dependency between accounts and user interface and database interface.

 ISSN: 3048-5320 (Online)

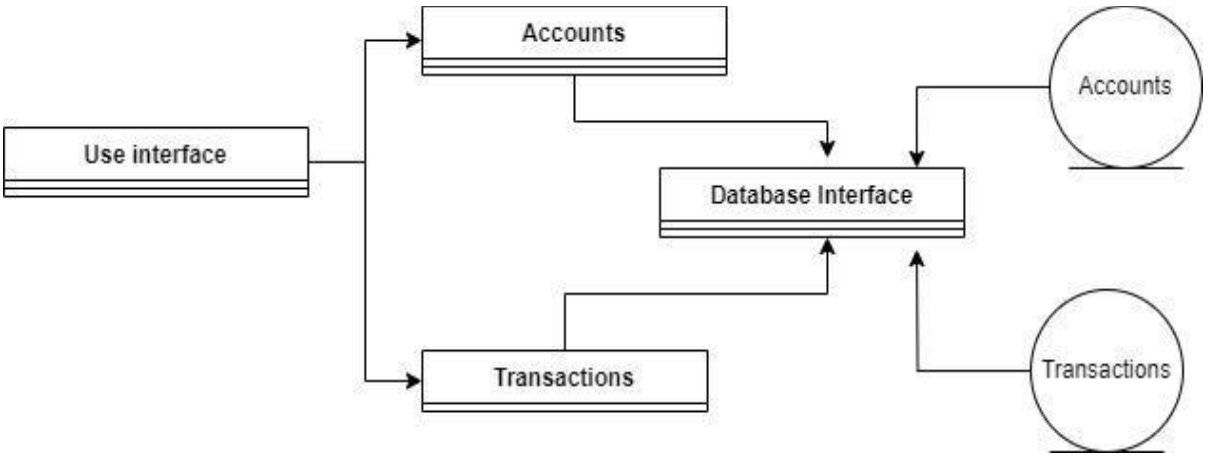


Figure 4: Update passbook use case

The update passbook use case involves retrieving data from the customer and transaction tables and subsequently generating a printed record in the passbook. However, in order to facilitate this process, it is necessary to design and develop an interface class that can accept account information for the purpose of updating the passbook.

The preceding discussion highlights the importance of creating a new class and utilizing existing classes as a reference point in order to effectively implement each use case. This emphasizes the relationship between the use cases depicted in the use case diagram and the classes depicted in the class diagram. If any use cases are omitted from the use case diagram, the class diagram will not accurately represent the necessary classes for implementing the use case. As a result, the system may fail to meet all user requirements and compromise the overall quality of the system. Therefore, it is recommended to follow a strategy that ensures the development of comprehensive and clear requirement specifications and a class model, which will ultimately enhance the quality of the system.

VII. THE PROPOSED STRATEGY:

Based on the experience and observation we recommend following strategy to enhance the quality of the proposed system. If the steps recommended are followed it can improve the overall system quality considerably.

1. Engage with the user to ascertain the functional requirements.
2. Build the list of use cases and user roles.
3. Construct a use case diagram that illustrates all the functions and user roles.
4. Write use case description for each use shown in diagram.
5. Present the diagram to both users and team members for verification and validation.
6. Refine the list of use cases and user roles.
7. Incorporate any necessary modifications to the model following the verification and validation process.
8. Analyse each use case depicted in the use case diagram along with the use case description and consider the classes needed for successful implementation.
9. Build the list of classes.
10. Identify the relationships between classes, such as association, inheritance, aggregation, and interface.
11. Validate the class diagram by cross-referencing it with the use case diagram to ensure that all required classes are accurately represented.
12. Seek verification and validation of the class diagram from members of the software development team.

13. Refine the list of classes and make necessary changes to class diagram if required.

As the quality of the system is always dependent on how many user requirements are fulfilled by the system, the strategy discussed above involves users, customers and team members we will be able to obtain their suggestions that can improve the quality of the system. This will lead to a complete and correct solution to the user problem.

VIII. SUMMARY:

The study delves into the correlation between use case diagrams and class diagrams in software engineering. It illustrates how use case diagrams clearly define the expected functions of a system from a user's perspective, which class diagrams then depict through the formulation of classes and their interconnections. The paper underscores the importance of accurate and comprehensive use case diagrams for the development of class diagrams. It also highlights the critical role of requirement engineering in gathering and modelling requirements during software development. The paper concludes by emphasizing that requirement engineering is an integral phase in software development, as it aids in meeting user expectations, thereby mitigating potential problems such as time and cost overruns. The researchers also propose a method for actualizing use cases through class diagrams, which includes real-life examples.

IX. CONCLUSION:

In conclusion, the paper firmly establishes the symbiotic relationship between use case diagrams and class diagrams in the sphere of software development. It sheds light on how use case diagrams facilitate the understanding of external user expectations, whereas class diagrams aid in realizing these expectations through classes, attributes, and operations. The paper underscores that the accuracy and completeness of the use case diagram is vital for the correct formation of the class diagram. The significant role of requirement engineering in software development cannot be refuted, as it is pivotal in designing a fool proof system, and circumventing issues such as missing or conflicting requirements. Failures in the requirement engineering process can lead to various detrimental effects such as missed schedules and unsatisfied customers. The paper proposes a method to implement use cases through class diagrams effectively, thereby advocating for strong interaction and collaboration between classes. The paper successfully presents the connection between use case and class diagrams and reiterates the pivotal role of requirement engineering in successful software development.

X. FUTURE WORK:

The current research paper can serve as a foundation for future work in the area of software development, with specific focus on requirement engineering. The proposed methods for translating use cases into class diagrams could be refined or other techniques could be explored. Since one of the main challenges identified in requirement engineering is the issue of missing or conflicting requirements, future research could focus on innovative strategies to ensure the accuracy and completeness of the requirements to reduce related problems. Furthermore, it would be beneficial to develop automated methods when converting functional requirements into class models, as proposed by the authors could be a very fruitful area for further investigation.

This research has its limitations, including a need for empirical evidence to substantiate the proposed method. The usage of a savings bank account system as only example also restricts the generalizability of findings to other software development scenarios. To mitigate these limitations, future research should include empirical studies to validate the relationship between use case and class diagrams, as well as examine diverse software development contexts. Multiple examples sourced from various genres of software development would provide comprehensive insights into this relationship and how it can be utilized for efficient software development.

REFERENCES:

1. Bochicchio, M., Bruno, A. and Longo, A. (2012). Supporting Continuous Improvement in Care Management with BPM. International Journal of Software Engineering, 1(2), pp.32–38. doi:https://doi.org/10.5923/j.se.20110102.04.
2. Cheng, B.H.C. and Atlee, J.M. (2007) 'Research Directions in Requirements Engineering,' IEEE Computer Society Conference [Preprint]. https://doi.org/10.1109/fose.2007.17.
3. Christel, M.G. and Kang, K.C. (1992) Issues in requirements elicitation. https://doi.org/10.21236/ada258932.
4. Eriksson, H.-E., Magnus Penker, Lyons, B. and Fado, D. (2003). UML 2 Toolkit. John Wiley & Sons.
5. Gareth Rogers (2016) 'RE in Agile Projects: Survey Results' – Requirements Engineering Magazine. [online] Available at: https://re-magazine.ireb.org/articles/re-in-agile-projects-survey-results [Accessed 4 Oct. 2023].
6. Gopal, K. et al. (2016) 'Software quality problems in requirement engineering and proposed solutions for an organization in Mauritius,' International Journal of Computer Applications, 137(2), pp. 23–31. https://doi.org/10.5120/ijca2016908698.
7. Huzoree, G. and Ramdoo, D. (2015) 'A systematic study on requirement engineering processes and practices in Mauritius,' Computer Science and Software Engineering, 5(2). https://espace.curtin.edu.au/handle/20.500.11937/16937.
8. Kamalrudin, M. and Sidek, S. (2015) 'A review on software requirements validation and consistency management,' International Journal of Software Engineering and Its Applications, 9(10), pp. 39–58. https://doi.org/10.14257/ijseia.2015.9.10.05.
9. Liu, D. (2004) 'Automating transition from use-cases to class model,' Masters Thesis [Preprint]. https://doi.org/10.1109/ccece.2003.1226023.
10. Mona Batra1, Dr Archana Bhatnagar2, (2020) "A Requirements Engineering Process Model for Software Development", International Research Journal of Engineering and Technology (IRJET), Volume: 07 Issue: 07 | July 2020 e-ISSN: 2395-0056, p-ISSN: 2395-0072.
11. Sa'adillah, M. and Ali, R. (2022). Logical framework of information technology: Systematization of software development research. Telfor Journal, 14(1), pp.26–32. doi:https://doi.org/10.5937/telfor2201026s.