British Journal of Applied Science & Technology
15(3): 1-12, 2016, Article no.BJAST.24860
ISSN: 2231-0843, NLM ID: 101664541

SCIENCEDOMAIN *international*
www.sciencedomain.org

Automated Test Case Generation Using UML Class & Sequence Diagram

Syed Asad Ali Shah[1], Raja Khaim Shahzad[1], Syed Shafique Ali Bukhari[1] and Mamoona Humayun[1*]

[1]University Institute of Information Technology, PMAS Arid Agriculture University, Rawalpindi, Pakistan.

Authors' contributions

Author SAAS worked on complete UML diagrams, XML, implementation of code in C#, designed framework for the proposed approach and wrote the first draft of the manuscript. Authors RKS and SSAB performed the statistical analysis, wrote the protocol, managed the analyses of the study and literature searches. Author MH supervised the complete study. All authors read and approved the final manuscript.

Article Information

DOI: 10.9734/BJAST/2016/24860
Editor(s):
(1) Wei Wu, Applied Mathematics Department, Dalian University of Technology, China.
Reviewers:
(1) M. Bhanu Sridhar, GVP College of Engg. for Women, Vizag, India.
(2) S. K. Srivatsa, Prithyusha Engineering College, Chennai, India.
(3) Anonymous, Rzeszow University of Technology, Poland.
Complete Peer review History: http://sciencedomain.org/review-history/13897

Original Research Article

Received 4th February 2016
Accepted 10th March 2016
Published 28th March 2016

ABSTRACT

It is identically significant in today's vastly dynamic background with fluctuating requirements to improve test plans at each phase of the Software Development Life Cycle. Testing helps to make sure that the final software product works according to user needs and requirements. Various testing techniques are being used in order to test software according to its nature and complexity. Research demonstrations that the numeral software fails on account of such variations because appropriate testing is not possible on outmoded requirements. Some researchers have tried to figure out the ways to make automatic tools that generates test cases but there is small amount of studies done in achievement of this approach through UML Class diagram. Further, the growing complication of the projects mark manual testing infeasible. It demands for automatic testing of requirements to keep check on the variations. Many of the former approaches use intermediate forms for testing software that makes automation difficult. In this paper, we have highlighted the gaps and flaws in previous work. Based on identified gaps, a simple and optimal approach to

**Corresponding author: E-mail: ali.naqvi1989@gmail.com, mamoona@uaar.edu.pk;*

generate test cases by extracting class diagram along with sequence diagram is proposed. In this approach, a simple and easily understandable framework is developed that uses UML class and sequence diagrams to generate test cases precisely.

Keywords: Unified Modeling Language (UML); Model Based Testing (MBT); Software Development Life Cycle (SDLC); Class Diagram; Object Oriented Language (OCL).

1. INTRODUCTION

Testing is considered as the most important phase in software development life cycle [1]. It plays an important part in guaranteeing the worthiness and consistency of software. The testing spell depends upon the scope and intricacy of the software. It has been observed that half of the time during software development is expended by testing [2]. Testing actions include designing test cases that are orders of particular inputs, executing the program code through test cases, and then observing the results created by the execution. Test case formation is one of the toughest steps in testing [3]. Therefore, automated generation of test cases is one of the significant challenges in software testing, as manual testing is laborious and error-prone [4]. Test case generation in exercise is mostly commenced manually since automated test case generation techniques need formal or semi-formal requirement to select test case to distinguish faults in the code enactment. So designing a huge figure of test cases and testing is highly labor intensive and time-consuming task. Test case generation automatically can diminish the progress cost by eradicating manual test case design struggle and support reliability through augmented test coverage [5]. Test case generation from design has much more importance because it helps in figuring out the errors in design due to which reliability of software increases. Model Based Testing (MBT) is considered as extra effectual and operative than code-based methodology as it is the miscellaneous approach of specification requirements and source code in order to test the software [6].

UML (Unified Modeling Language) is the modeling language, seeking great attention as the engineering de-facto standard for modeling object-oriented software systems in the field of testing. UML is a design language for imagining, stipulating, creating, and detailing the relics of a software-intensive system. The three significant aims for using design model in object oriented program testing are: (1) outmoded testing of software techniques are the only static interpretation of the code that is insufficient to test dynamic aspects of the object-oriented system; (2) Usage of code/program in order to test an object-oriented system is difficult and time-consuming assignment. In distinction, models aid software testers to recognize systems in a better way and to find out test data by performing simple processing as related to code (3) Generating test cases using model-based approach is intended at the primary phase of the SDLC, permitting to carry out coding and testing in parallel [7].

Among UML diagrams, one of the very common diagrams is a Class diagram. It can be used for various purposes and at different times in the development phases. Class diagrams are frequently applied to analyze the application domain and to pin down the terminology to be used. They are usually taken as a basis for discussing things with the domain experts, who cannot be anticipated either no programming or computer background at all; therefore, they remain relatively simple. A class diagram being a part of UML structural diagrams is a static model providing a platform for the dynamic model also comprises of interface information with properties. Every class contains three portions that are name, characteristics and methods [8]. Classes signify the association's model and semantic relationships among problematic concepts. Generalization/Specialization in the class diagram defines a classification from the bottom up approach. The class defining mutual concepts will be known as the generalization of subclasses. The aggregation in class diagram is also a type of association [9]. The vital unit of testing an object-oriented application consists of a class and class-testing efforts are centered on functional testing [10]. In the diagram, classes are represented with boxes, which contain three parts:

1. The top part contains the name of the class. It is printed in bold and centered, and the first letter is capitalized.
2. The middle part contains the attributes of the class. They are left aligned and the first letter is lowercase.

3. The bottom part contains the methods the class can execute. They are also left aligned and the first letter is lowercase.

Below is the example of a class diagram as shown in (Fig. 1), TRAIN JOURNEY class with its attributes and methods:

The UML sequence diagram is a behavioral diagram used for modeling the behavior of an object through its sequence of order [11]. The Sequence Diagram represents the association of objects based upon messages and time sequence. It demonstrates how the objects interrelate with others in a specific situation of a use case. The diagrams are mainly well appropriate for object-oriented software, where they signify the movement of control while interaction between object [12]. The diagram may also cover extra info about the flow of control throughout the interactions, such as conditions (e.g. “if condition is equal to c then sends particular message m else send message n”) and iteration (e.g. “send message m several times”) or state-dependent conduct [13].

A sequence diagram demonstrations parallel vertical lines (lifelines), various processes or objects that live concurrently and horizontal arrows, the messages switched between them, in the order in which they occur. This permits the description of simple runtime scenarios in a graphical mode. Below is the example of simple sequence diagram as shown in (Fig. 2).

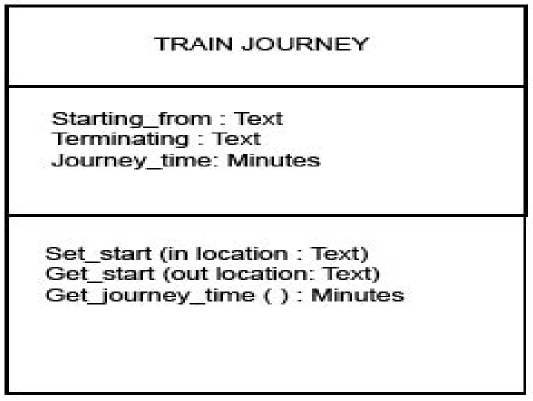

TRAIN JOURNEY
Starting_from : Text
Terminating : Text
Journey_time: Minutes
Set_start (in location : Text)
Get_start (out location: Text)
Get_journey_time ( ) : Minutes

Fig. 1. Example of single class

Above a structural diagram, class diagram can be used to extract the static elements of like methods and objects, similarly sequence diagram being a behavioral diagram will provide us with sequence of message. Therefore, our focus is on UML class and sequence diagram for generation of test cases.

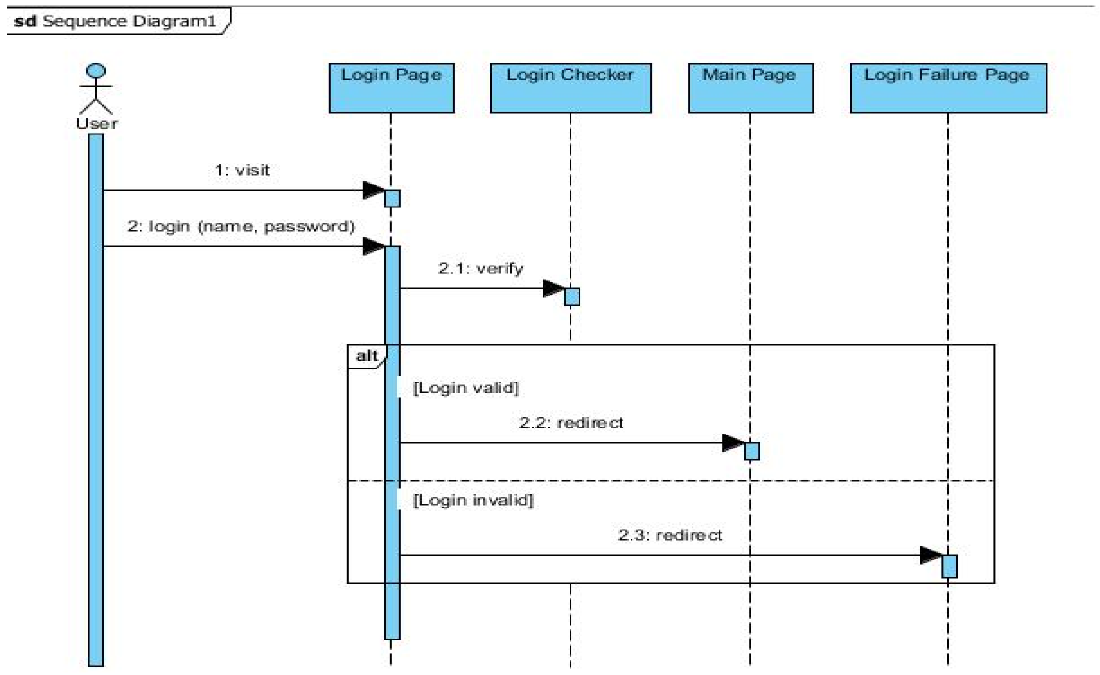


Fig. 2. Example of sequence diagram

2. LITERATURE REVIEW

The studied literature shows that there are various integrated methods described by numerous researchers for automated test case generation from UML class diagram.

Wang and Zheng [9] presented a methodology to produce the test cases from a design level class diagram with interaction diagram. It is considered to be in the category of model-based testing. It is supposed that the given model was accurate and the objective of the testing was to check whether the implementation imitates to the given model. They used a car rental example to demonstrate test case generation. The test competence criteria used in this paper was the coverage for the model elements that are also called the building blocks in the class and interaction diagrams. That criteria was based on the similar principle for the primary code testing criteria to aid defined testing objectives. Their method cannot check the correctness of the model.

Prasanna, Chandran and Suberi [10] presented the class diagram by using data flow approach. In their approach, data variables and member methods were extracted from UML class diagram and they also used data flow technique to generate test cases. They discussed that applying this approach may leads to intra-class data flow inconsistencies; they had removed those anomalies which leads to feasible test cases for testing. They created a directed flow graph which helps to express use pair approach and to detect the anomalies. After that, infeasible sequences were erased from the data flow sequences and specific valid test cased were created. Their approach cannot work for Nested State Charts and in result, couldn’t achieve maximum coverage.

Anbunathan and Basu [14] proposed a novel approach for making structural test cases through UML Class diagram and corresponding UML State diagram. Traditional testing practices like DD path, basis path and DU path testing procedures were considered in order to generate test cases. Their technique is not compatible with complex class diagram.

Alhroob, Dahal, and Hossain [15] proposed an approach using class diagram and Object Constraint Language (OCL). In their approach, class diagram and Object Constraint Language (OCL) were used to signify specification for each classification and connected classes in the software specification is represented by names and their attributes in the class diagram. To ensure that associations are reliable, an automatic methodology was proposed to capture and hold the class relationships in an organized way.

Weißleder and Sokenou [16] presented an approach in which they used Class diagram, state machines and OCL expressions and for producing test cases. OCL pre-conditions and post-conditions of actions and guard circumstances of state machines were taken to automatically find out the test data input panels. At the end, their results obtained from the judgements to commercial tools using mutual analysis, solitary holds for examples with some loops and a few conditions relied upon the repetition of executing those loops.

Li and Maibaum [17] presented their technique in which they used integration testing against object-oriented codes. They projected a methodical approach in order to test object-oriented codes at the integration level. In their approach, they produced test cases through UML class and sequence diagrams and executed the test cases with the aid of coordination contracts that is considered as a source of automated test execution.

Verma [18] projected a technique for generating test cases by taking four diagrams that are Class Diagram, Sequence diagram, State chart Diagram and Use case Diagram. According to the authors, test case generation based upon the behaviour is uncommon. In this approach, Petal files were formed for Sequence diagram, Class Diagram, Use case Diagram and State chart Diagram independently. Temporary buffers were produced in which information was stored that was extracted for the diagram for example, there was a distinct buffer for class name buffer, class attribute buffer etc and is same for other diagrams as well. For every diagram, there created a file to store the information extracted out from buffers. Then those files were stored in the database in the tuples and test cases were generated from this table with the help of matching strings.

Albert et al. [19] proposed a method (formalized as a M2M transformation using ATL) which makes a set of simple procedures for initial clearly static conceptual schema. Procedures created by their scheme served to fulfil all basic alteration progressions (inserting/updating/

deleting) towards the method under progress. The total quantity and behaviour of the actions were presumed from the features of the structural components (classes, relations and so forth) in the input plan. Their building procedure assures that none of the immaterial operations were defined and all created processes were powerfully executable with respect to the utmost mutual structural possessions in CSs (like multiplicity property). With regard to future work, they considered prolonging their technique to deal with extra possessions and limitations that were not addressed in their work, for example association between classes. So, their work cannot deal with the relationship between classes that is association.

Sawant and Shah [20] presented the paper related to a novel technique that creates test cases from UML models. In their technique, the UML diagrams such as Use Case Diagram, Sequence Diagram & Class Diagram of some application were considered for making the test cases. A graph was generated to accumulation the essential information that could be extracted from mentioned diagrams & data dictionary stated in OCL for the similar application. The graph was lastly scanned to produce the test cases that were suitable for system testing.

Sharma and Singh [21] proposed an approach for generating test cases and their approach also determines server hitting cost using UML modelling diagram. Initially, they drew a UML sequence diagram and then converted it into the particular control flow graph (CFG) for each test case. CFG nodes amplified with various information that was necessary to constitute test vectors. It is individuality of their research that it can easily distinguish the testing cost by attaining the server hit cost. This indicates that their approach can work with large design i.e. scalable, suitable and well-organized.

2.1 Critical Review of Some Approaches

Existing approaches for test case generation are compared on the basis of parameters there are given below which are considered as developer's viewpoint while scripting algorithm for system:

(P1) Testing level provide by the technique.
(P2) Number of diagrams used for test case generation.
(P3) Use of Intermediate forms during test case generation.
(P4) Number of steps used by particular technique for test case generation.

It has been seen that approaches that used only class diagram need some kind of intermediate forms for test adequacy. Intermediate form makes test case automation difficult. Number of steps while generating test cases also affects the level of automation; small number of steps increases the level of automation whereas large number of steps decreases automation level. By analyzing the existing approaches, we have originated some desired parameters which need be consider while developing test case generation tool. Those are:

1) Suitable footsteps need be taken in order to make automation easier.
2) Accurate reading of diagrams should be available.
3) Need to use less complex intermediate form or no intermediate form.
4) Less steps in order to generate test cases.

While presenting our approach we have taken care of all these points. Our technique uses Class diagram then from class diagram fully loaded sequence diagram which does not use any intermediate form .This marks our technique totally distinct from the previous approaches as explained in Table 1.

Table 1. Review of techniques

Ref#	P1	P2	P3	P4
[9]	Integration testing	Class diagram, interaction diagram	No	3
[10]	System testing	Class diagram	Directed flow graph	4
[14]	Integration testing	Class diagram, state diagram	DD graph	5
[16]	Integration testing	Class diagram, state machine	Control flow graph	6
[18]	Regression testing	Class diagram, sequence diagram and state chart diagram and use case diagram	Petal file	5
[20]	Integration testing	Class diagram, sequence diagram and use case diagram	Sequence diagram graph	6
[23]	Integration testing	Class diagram, object diagram	No	6

3. PROPOSED APPROACH

After analysis and studying research papers, we have conducted a small case study. The proposed approach has been introduced to generate automated test cases. To make the approach easily understandable, we are using step wise explanation. Below are number of steps that describe complete procedure of our approach in (Fig. 3).

3.1 Creating Class Diagram

Visual Paradigm (Community Edition) tool is used to create class diagram as this tool provides many features to make diagrams easily. Class diagram will be constructed carefully according to requirements as it is using as the base diagram in the whole approach.

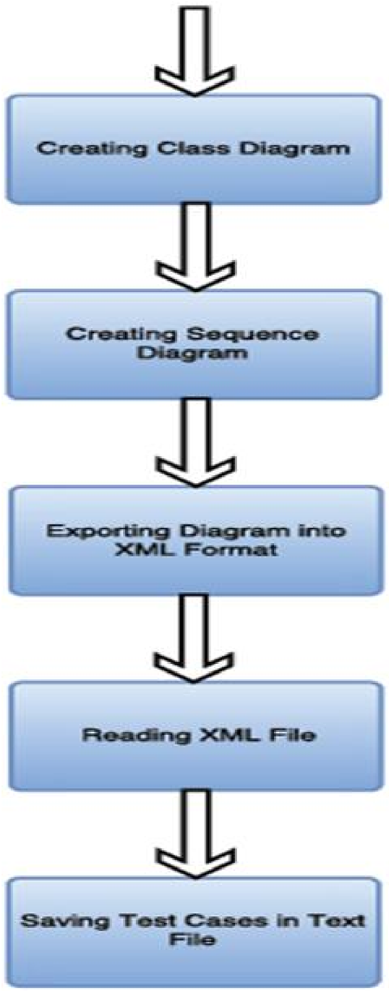

Creating Class Diagram
Creating Sequence Diagram
Exporting Diagram into XML Format
Reading XML File
Saving Test Cases in Text File

Fig. 3. Steps of proposed approach

3.2 Creating Sequence Diagram

Again same tool will be used to create sequence diagram. Methods and class name will be extracted from Class diagram and then used in sequence diagram while developing it. Through this approach, we can obtain links between all classes and static members of classes that will be used in sequence diagram and helpful in keeping a connection between both diagrams.

3.3 Exporting Diagram into XML Format

The developed diagrams will then be exported into XML format. XML files are widely used in IT for storing information and some programming code is needed to read those files. XML file saves information in the form to tags. In our approach, it stores information related to complete Sequence diagram. Visual Paradigm provides export to XML feature and by using this feature, we get XML file for sequence diagram.

3.4 Reading XML File

For reading required XML tags, we write code in C# that can identify and extract required tags in an order and stores it in a Data Table. Data table in C# is used to store information in tabular form on temporary basis. Code will be written with the intent of covering all test paths and generating minimum teat cases using different checks and correct logics in code.

3.5 Saving Test Cases in Text File

After getting all required information in Data Tables, we again use C# code to write test cases in some text file to keep record permanently. In the end, we get text file containing test cases for specific diagram.

To elaborate the concept of proposed approach, we design two diagrams. First, we draw a class diagram of simple standard registration system (see Fig. 4). Both diagrams are designed in Visual Paradigm. Visual Paradigm is a design tool that provides XML exporting feature as well. As we need XML file of desired diagram to generate test cases, we use this tool. Below is the class diagram explaining the system itself.

3.5.1 Class diagram

There are three classes in above diagram having different links between them. **RegistrationUI** class has simple association with **RegistrationController** class and then **RegistrationController** is depending on **User** class. **RegistrationUI** is a class that shows interface through which user interacts with the system. Now one can easily get the mappings or interactions between the classes after having a look on above example. Now let's proceed towards sequence diagram while keeping in mind the above class diagram. What we get is (see Fig. 5).

3.5.2 Sequence diagram

We can see that all the classes in class diagram are transformed into objects denoting each class individually. As **RegistrationUI** has registrationUI object on first lifeline and other class have same representation as shown in (Fig. 5). We have a method CreateUser() having some inputs between two objects that are **RegistrationUI** and **RegistrationController** and then between **RegistrationController** and **User** objects, we have **setName()** and **setPassword().** We have noticed that one can easily find the flow of sequence diagram with the help of extracting links and attributes from class diagram. Using static attributes from class diagram can help us in identifying mappings between static classes and those classes act as objects to carry out the flow of messages between them. Below is the model of the proposed approach as shown in (Fig. 6).

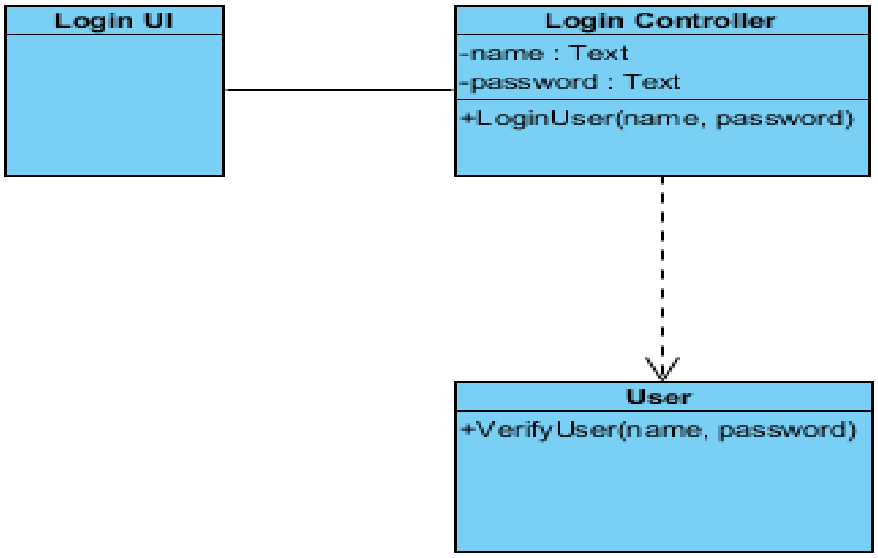


Fig. 4. Class diagram

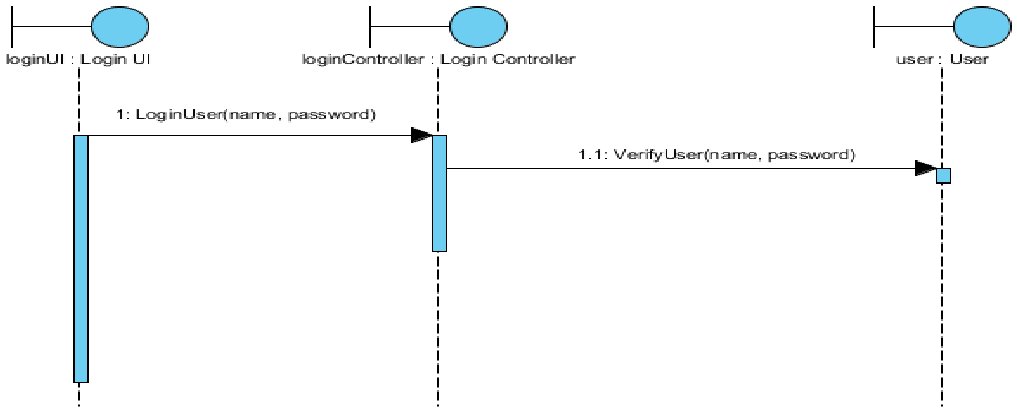

loginUI : Login UI
loginController : Login Controller
user : User
1: LoginUser(name, password)
1.1: VerifyUser(name, password)

Fig. 5. Sequence diagram

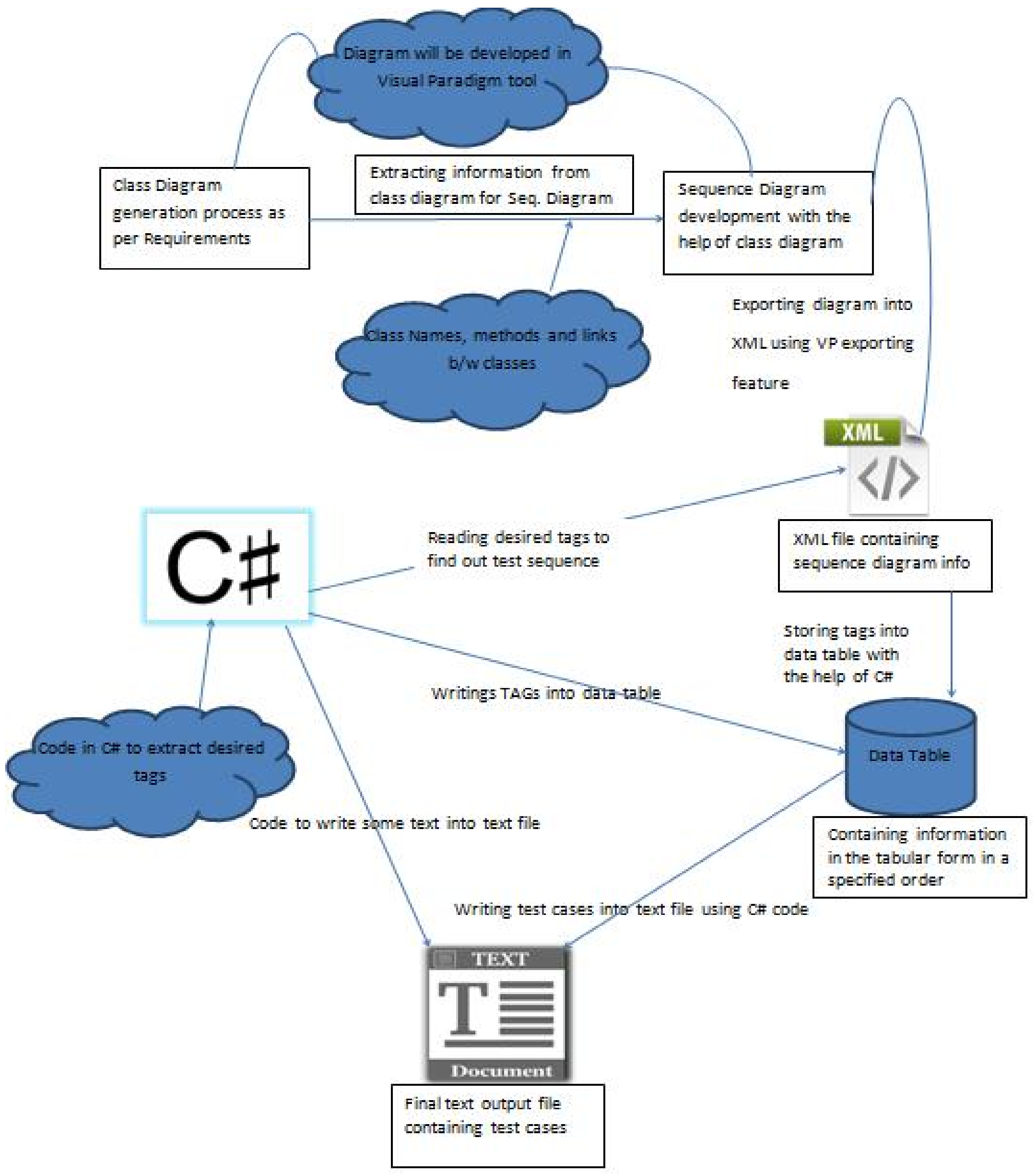


Diagram will be developed in Visual Paradigm tool
Class Diagram generation process as per Requirements
Extracting information from class diagram for Seq. Diagram
Sequence Diagram development with the help of class diagram
Class Names, methods and links b/w classes
Exporting diagram into XML using VP exporting feature
XML
XML file containing sequence diagram info
C#
Reading desired tags to find out test sequence
Storing tags into data table with the help of C#
Writings TAGs into data table
Code in C# to extract desired tags
Data Table
Containing information in the tabular form in a specified order
Code to write some text into text file
Writing test cases into text file using C# code
TEXT
Document
Final text output file containing test cases

Fig. 6. Proposed model

4. IMPLEMENTATION

A class diagram first develops in Visual Paradigm tool that is very easy to use and provides extensive features. After completing class diagram, we use 'New Diagram' option from generated class diagram. Visual Paradigm provides the feature to draw various diagram from Class diagram to establish a link between them. By using this feature, we get a complete Sequence diagram containing link, attributes and classes from class diagram. The purpose of using this feature is to creep the test case generation process towards automation. The generated sequence diagram has complete data of class diagram and we can proceed towards next step that is converting Sequence diagram into XML. XML is widely used in IT for various purposes. In this case, it will store all the information related to generated class diagram in the form of tags. Next step is to read that XML file with some coding language to extract required tags/data. C# code has been used by us to read XML file. As VP tool also exports a lot of further information in XML file that is useless for us in this scenario, we only find required set of tags by filtering the XML file with the help of C# code. All of the required tags will be store in the form of data tables. The core information that is required was Test Case No., Class Name, Message from Class, and Message to Class, Message and its Parameters. Mentioned information was kept in an order as XML stores classes in an order as per order found in the diagram. Step by step implementation of tool is described below.

Table 2. Class names with objects

Class	Object
Login UI	loginUI
Login Controller	loginController
User	user

Above Table 2 is showing extracted Class names along with their objects. **Login UI** is the name of class having object name **loginUI** which is in the next column to class. Each Class are represented with their objects respectively.

Table 3. Class names with methods

ClassName	Method
Login Controller	LoginUser
User	VerifyUser

Above Table.3 represents Class name with its method. It can be seen that **Login Controller** is the class with the method **Login User** and same is the case with next class.

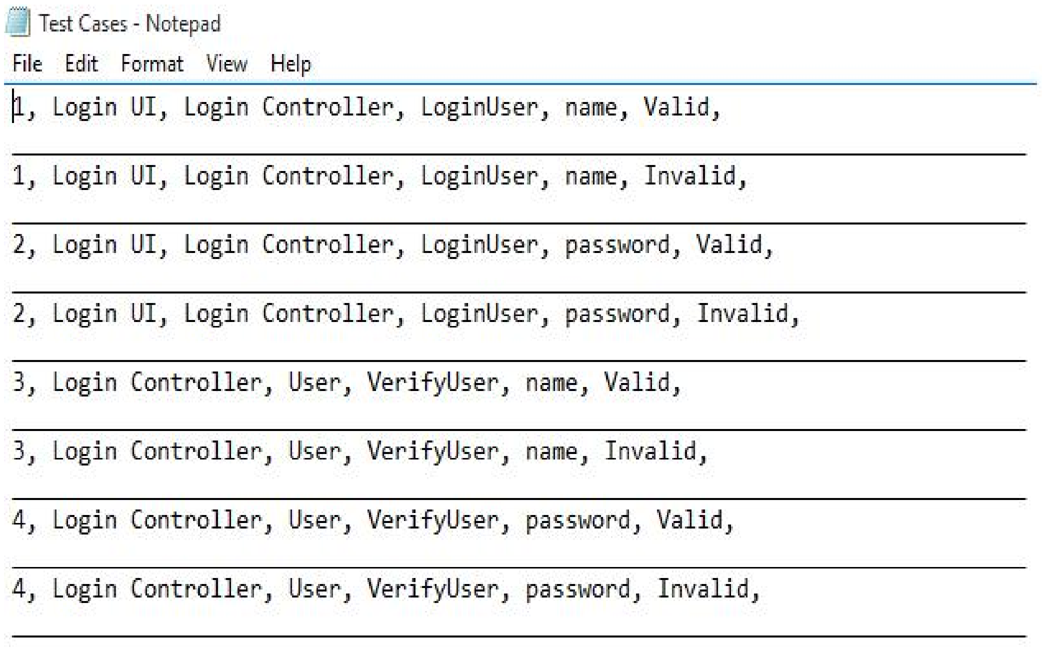
Table 4. Method names with parameters

Method	Parameter
LoginUser	Name
LoginUser	Password
VerifyUser	Name
VerifyUser	Password

Above mentioned Table 4 now shows methods with their respective parameters. In above case, there are two parameter of each method for e.g. **LoginUser** is a method with two parameters **name** and **password.** After getting information in above three Tables (2, 3 and 4), we formalize that extracted data into a sequence with the help of C# code. After building sequence, we get information in the form of test cases.

In generated test cases, we can see that for every parameter, we have two test cases i-e **Valid** or **Invalid.** Generated test cases can be clearly seen in the Table 5. The last step is storing the above tabular information in text file.

Below text file is showing generated test cases that have been permanently stored (see Fig. 7).



```
1, Login UI, Login Controller, LoginUser, name, Valid,
1, Login UI, Login Controller, LoginUser, name, Invalid,
2, Login UI, Login Controller, LoginUser, password, Valid,
2, Login UI, Login Controller, LoginUser, password, Invalid,
3, Login Controller, User, VerifyUser, name, Valid,
3, Login Controller, User, VerifyUser, name, Invalid,
4, Login Controller, User, VerifyUser, password, Valid,
4, Login Controller, User, VerifyUser, password, Invalid,
```

Fig. 7. Screen shot of text file

Table 5. Generated test cases

Test case no.	From class	To class	Methods	Parameter	Result
1	Login UI	Login controller	LoginUser	name	Valid
1	Login UI	Login controller	LoginUser	name	Invalid
2	Login UI	Login controller	LoginUser	password	Valid
2	Login UI	Login controller	LoginUser	password	Invalid
3	Login controller	User	VerifyUser	name	Valid
3	Login controller	User	VerifyUser	name	Invalid
4	Login controller	User	VerifyUser	password	Valid
4	Login controller	User	VerifyUser	password	Invalid

5. RESULTS

While comparing proposed approach with other approaches, it has been observed that most of the approaches used class diagram in cooperation with sequence and other UML diagrams. Data from class, sequence and other diagrams are extracted to generate test cases and mentioned approaches have used some kind of intermediate forms in order to generate test cases. In our approach sequence diagram is created from class diagram’s attributes and then test cases are generated from that sequence diagram. Secondly, our proposed approach does not use any intermediate form. Similarly, some approaches [10,22,24] also used a single diagram (class diagram) but these approaches also used intermediate forms. Thus it has been observed from above comparison that our approach uses only sequence diagram without using intermediate in order to increase the level of automation in test case generation domain.

Table 6. Comparison of approaches

References	Using only class diagram	Intermediate form	Provision of Tools
Our approach	✓	×	✓
[10]	✓	✓	✓
[16]	×	✓	×
[17]	×	×	✓
[18]	×	✓	✓
[20]	×	✓	×
[21]	×	✓	×
[22]	✓	✓	✓
[24]	✓	✓	✓
[25]	×	×	✓
[26]	×	✓.	×
[27]	×	✓	×
[28]	×	✓	✓

6. DISCUSSION

From the literature, it can be seen that various studies have been proposed to generate test cases from UML behavioral diagram along with class diagram. Most of the studies used intermediate forms, huge number of steps, various kinds of graphs or trees in order to complete test case generation process. Rare studies have been spotted which used XML instead of XMI in order to retrieve required data. Some studies proposed complex solutions that are not easily understandable to everyone. The theme of this study is to deliver a very simple and easily understandable approach that follows few steps to accomplish test case generation process without using any intermediate form and some database. This approach is a step forward towards quick test case generation before the coding phase begins. Keeping in mind the above issues, this study aimed to minimize mentioned drawback up to some extent by avoiding them. Test cases have been clearly displayed for better understanding of results of proposed technique. All steps along with figures have been shown in order to make it explainable at its best. Fig. 6 has been presented to describe complete details of proposed model in a comprehensible way.

7. CONCLUSION

In today's growing needs, change in requirements needed with the change in complexity of systems, it becomes mandatory for software testing team to test the software before the start of coding phase. Manually, it is not easy to test the system and the only solution is automated testing. This study presented a simple tool that can generate automated test cases using class and sequence diagram without any intermediate form. Sequence diagram was generated from class diagram's that includes class name, methods, parameters and connection between classes. Then exported to XML file that has been read by C# code for test case generation. Based upon the data extracted from XML, test cases were generated.

In future, complex class and sequence diagrams can be taken for huge systems. Furthermore, maximum coverage and more optimal results would be considered in order to improve results.

COMPETING INTERESTS

Authors have declared that no competing interests exist.

REFERENCES

1. Mall R. Fundamentals of software engineering. International Journal of Computer Science and Informatics. 2013;3(2):14-21.
2. SWAIN, et al. Generation of test cases using activity diagram. International Journal of Computer Science and Informatics. 2013;3(2):1-10.
3. Abdurazik A, Offutt J. Using UML collaboration diagrams for static checking and test generation. In Proceedings of the 3rd international Conference on the UML. Lecture Notes in Computer Science, Springer-Verlag GmbH. 2000;1939:383-395.
4. Linzhang, et al. Generating test cases from UML activity diagram based on gray-box method. Proceedings of the 11th Asia-Pacific Software Engineering Conference (APSEC'04), IEEE. 2004;284-291.
5. Jena, et al. A novel approach for test case generation from UML activity diagram. International Conference on Issues and Challenges in Intelligent Computing Techniques (ICICT), IEEE. 2014;621-629.
6. Ali, et al. A state based approach to integration testing based on UML models. Journal of Information and Software Technology. 2007;1087-1106.
7. Shukla SG, Chandel GS. a systematic approach for generate test cases using UML activity diagrams. IRACST-International Journal of Research in Management and Technology (IJRMT). 2012;2:469-475.
8. Shirole M, Kumar R. UML behavioral model based test case generation: A survey. ACM SIGSOFT Software Engineering Notes. 2013;38:1-13.
9. Wang Y, Zheng M. Test case generation from UML model. 45th Annual Midwest Instruction and Computing Symposium. 2012;1-8.
10. Prasanna M, Chandran KR, Suberi DB. Automatic test case generation for UML class diagram using data flow approach. Academia Education. 2011;1-7.
11. Panthi V, Mohapatra DP. Automatic test case generation using sequence diagram. Proceedings of ICAdC, AISC 174, Springer Berlin Heidelberg. 2013;277-284.
12. Panthi V, Mohapatra DP. Automatic test case generation using sequence diagram. International Journal of Applied Information Systems (IJAIS), Foundation of Computer Science FCS. 2012;2(4):22-29.
13. Fraikin F, Leonhardt T. SeDiTeC – Testing based on sequence diagrams. ASE '02 Proceedings of the 17th IEEE International Conference on Automated Software Engineering. 2000;261-266.
14. Anbunathan R, Basu A. Dataflow test case generation from UML class diagrams. Computational Intelligence and Computing Research (ICCIC), IEEE International Conference. 2013;1-9.
15. Alhroob A, Dahal K, Hossain A. Automatic test cases generation from software specifications modules. Journal of e-Informatica Software Engineering. 2010; 4(1):109-121.
16. Weißleder S, Sokenou D. Automatic test case generation from UML models and OCL expressions. Conference on Software Engineering in München; 2008.
    Available:http://citeseerx.ist.psu.edu/viewdoc/summary?
    DOI: 10.1.1.541.485
17. Li Z, Maibaum T. An approach to integration testing of object-oriented programs. Seventh International Conference on Quality Software (QSIC '07), IEEE. 2007;268-273.
18. Verma A. Automated Test case generation using UML diagrams based on behavior. International Journal of Innovations in Engineering and Technology (IJIET). 2014;4(1):31-39.
19. Albert, et al. Generating operation specifications from UML class diagrams: A model transformation approach. Data and Knowledge Engineering, ELSEVIER. 2011;70:365-389.
20. Sawant V, Shah K. Automatic generation of test cases from UML models. International Conference on Technology Systems and Management (ICTSM), Proceedings published by International Journal of Computer Applications (IJCA) IEEE society. 2011;7-10.
21. Sharma A, Singh M. Generation of automated test cases using UML modeling. International Journal of Engineering

Research and Technology. 2013;2(4): 1833-1835.

22. Shanthi AVK, Mohankumar DRG. Automated test cases generation for object oriented software. Indian Journal of Computer Science and Engineering (IJCSE); 2011. Available:http://citeseerx.ist.psu.edu/viewdoc/download? DOI: 10.1.1.300.9789&rep=rep1&type=pdf
23. Name Withheld. Automatically generating test cases using uml structure diagram. University of Delaware, Newark, USA. 2009;1-7. Available:http://citeseerx.ist.psu.edu/viewdoc/summary? DOI: 10.1.1.170.1937
24. Mondal SK, Tahbildar H. Automated test data generation using fuzzy logic-genetic algorithm hybridization system for class testing of object oriented programming. International Journal of Soft Computing and Engineering (IJSCE). 2013;3(5).
25. Asthana S, Tripathi S, Singh SK. A novel approach to generate test cases using class and sequence diagrams. Springer-Verlag Berlin Heidelberg Contemporary Computing Communications in Computer and Information Science. 2010;95:155-167.
26. Kaur P, Kaur R. Approaches for generating test cases automatically to test the software. International Journal of Engineering and Advanced Technology (IJEAT). 2013;2(3):191-193.
27. Biswal BN, Nanda P, Mohapatra DP. A novel approach for scenario-based test case generation. International Conference on Information Technology (ICIT), IEEE Society. 2008;244-247.
28. Dinh-Trong TT, Ghosh S, France RB. A systematic approach to generate inputs to test UML design models. Software Reliability Engineering, ISSRE '06. 17th International Symposium, IEEE Society. 2006;95-104.