A Review of Class Based Test Case Generation Techniques

Syed Asad Ali Shah*, Raja Khaim Shahzad, Syed Shafique Ali Bukhari, Nasir Mehmood Minhas, Mamoona Humayun

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

* Corresponding author. Email: asadalishah_islamabad@yahoo.com
Manuscript submitted January 13, 2015; accepted March 13, 2015.
doi: 10.17706/jsw.11.5.464-480

Abstract: One of the important phases in software development is testing, due to advancement in software development in recent years, testing has also grabbed great attention and various testing techniques have been devised in order to test the software. Researchers are rapidly moving towards model based testing due to its advantages. UML class diagram has been used alone or with some hybrid approaches in order to achieve concentrated level of test case generation. However, before progression towards new approaches, there is a need to analyze and synthesize existing evidence based techniques in the area of software test case generation. In this paper, we have discussed various approaches that exercises class diagram to support their test case generation activities. Systematic Literature Review (SLR) was conducted to collect related papers to examine the relevant research. The outcome of this study shows analysis of class based test case generation techniques. In the end, we elaborated the usage and significance of class diagram in generating test cases with the help of evidences and propose thinkable direction and practices for further research.

Key words: Model based testing (MBT), unified modeling language (UML), systematic literature review (SLR), class diagram.

1. Introduction

Testing is considered as the most important phase in software development life cycle [1]. It plays an important role in guaranteeing the quality and reliability of software. The testing time is identified as an important factor that depends upon the scope and complexity of the software. It has been observed that half of the time in software development is consumed by testing [2]. Testing activities include designing test cases that are sequences of inputs, running the code with test cases, and exploring the results produced by the execution.

Process of software testing is used to verify and validate system under test by executing system. Intent of testing is to find errors in specific system or a part of system with the ultimate objective of providing assurance that software meets its requirements. Testing can be performed either automatically or manually. In manual approach, testing can be achieved by executing system code manually and then compare obtained output with specified behavior to records the observations [3]. Manual process of testing does not provide the suitable outcome that leads to inefficiency and ineffectiveness of the system. The finest approach is to test the software automatically to raise the usefulness, effectiveness and fulfilling coverage criteria of software testing. Automated testing practices means to test the system which involves no speculation of human and only investment of money and resources are desired. If we compare it with

manual testing, automated testing provides precise and quicker results. The most corporate mean to test the software is to generate the test cases and then test automatically by some automated tool. Afterward, results are compared with best cases to find optimality.

Model Based Testing (MBT) is attaining its attractiveness in academia and in industry as well due to its features. As systems are growing in complication, more systems execute mission-critical functions and steadiness requirements such as security, consistency, accessibility and safety factors are becoming important for *the* users. Model Based Testing is more capable and operational than code based methodology as it is the mixed approach of source code and describing needs in order to test the software [4]. The competitive marketplace is compelling corporations to describe or implement new approaches to lessen the time to market as well as the development cost of critical systems. Much focus has been sited on front-end development exertions without understanding the fact that testing accounts for 40 to 60 percent of the lifetime development and maintenance costs [5], [6]. Testing is conventionally accomplished after the completion of development but market-driven agendas frequently demand organizations to release products before they are satisfactorily tested. Model-based development tools are rapidly growing in routine because they are famous to deliver noticeable paybacks by aiding simulation and code generation along with the standard design and analysis tasks. These tools aid users to improve requirement and design models of specified systems. The key challenge is to interpret development oriented modeling languages into a form that is appropriate for automated test path generation, condition based test coverage examination, constraints to test traceability, and design-to-test traceability.

UML (Unified Modeling Language) is the modeling language that is achieving great attention due to industrial de-facto standard for modeling object-oriented software systems in testing. The Unified Modeling Language (UML) is a design language for imagining, stipulating, creating and documenting the artifacts of a software-intensive system. Major three reasons for testing design models in object oriented programs are: (1) Outmoded software testing techniques use only static interpretation of code which is insufficient to test dynamic aspects of object-oriented system. (2) Usage of code/program in an order to test an object-oriented system is difficult and boring task. In distinction, models aid software testers to recognize systems in a better way and to find out test data by performing simple processing as compared to code. (3) Generating test cases using model based approach is planned at an early stage of the SDLC that allows carrying out coding and testing in parallel [7].

Among UML diagrams, one of the very common diagrams is Class diagram. It can be helpful for several determinations and at multiple intervals in the development life cycle. Class diagrams are frequently used to examine the application domain and to identify the language to be used. They are generally taken as a source for deliberating things with the domain specialists, who cannot be anticipated to have any programming *knowledge* and computer background at all; therefore, they remain comparatively simple. A class diagram is a static model which provides an environment for the dynamic model also includes interface information and property guards. Every class consists of the three parts that are name, attribute and functions [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 elementary unit of testing an object-oriented presentation is a class and class-testing efforts are mostly based upon functional testing [10]. In class diagram, classes are signified with rectangles consisting of three parts: (1) the topmost rectangular part comprises the name of the class. It is written in bold with centered alignment and its first letter is capital (2) the middle part contains the attributes of the class that are aligned left and first letter is in lowercase (3) the bottom part contains methods the class can execute. Methods start with first lowercase

letter observing left aligned format.

For this *particular* study, the exploration of related work is mostly discovered about the concerns of test case generation techniques. Later on, more investigation was carried out for generating test cases using class diagram and only specific and relevant literature has been taken for deep study.

This paper presents the systematic review of class based test case generation techniques. It has been investigated *through* existing studies that there is no specific comprehensive survey conducted so far which only considered test case generation through class diagram. Systematic literature review (SLR) methodology has been used for the supposed inquiries and a comprehensive analysis has been performed.

The *remaining* paper is organized as: Section 2 presents the related work, Section 3 presents the methodology, Section 4 presents the results and analysis, Section 5 presents the discussion and Section 6 presents conclusion.

2. Related Work

This section discusses the related work done on test case generation techniques that used class diagram completely or partially. Various techniques are considered along with their approaches on class based test case generation.

Karambir [11] conducted a review on test case generation techniques. After studying various techniques, they classified them into three main categories (i.e.) test case generation using GA, using random testing technique and using model based testing for generating test cases by analyzing the dynamic behavior of objects. In the end, their study discussed that defects in the design model can be detected during the analysis of the model itself. So, the weaknesses can be eliminated as prompt as possible that results in decreasing the cost of defect removal.

Kaur [12] performed review for test case generation techniques based upon hybrid approach by analyzing the dynamic behavior of UML diagram using evolutionary algorithm. In the end, they proposed a technique by using sequence diagram with multi objective GA to generate test cases.

Pahwa [13] presented various techniques for test case generation that were based upon UML diagrams. Their focus was on effective use of UML techniques and test cases generated from them in order to make suitable executions.

Tahiliani [14] conducted a survey that identified different techniques used for deriving test cases from use case based approaches and UML Diagrams. In their brief analysis, they discussed the advantages and drawbacks of both approaches and none of the techniques describe complete process with their stated algorithm.

Kaur [15] performed comprehensive survey for test case generation techniques using UML diagrams. At the end, they classified those techniques on the basis of number of activities performed, tool used and coverage criteria for automated test case generation.

Hooda [16] presented a review based upon different test case generation techniques for minimization of test cases, selection, prioritization and estimation techniques. Their main focus was on those various techniques which help the test engineers to schedule and rank the test cases to reduce the total effort, time and the cost.

Prasanna, M., *et al*. [17] presented a survey on automatic test case generation using UML diagram. They classified techniques in three main categories: (1) specification based (2) model based and other (3) hybrid approaches. After that, they identified issues in usage of those existing techniques such as a random test case generator may create numerous test statistics but might fail to identify test case to fulfil requirements. A path oriented approach identifies path for which test case has to be generated but the path might be infeasible and the test data generator might fail to find out input that traverse through the path. An

intelligent approach generates test case quickly but is quite complex, so, the model based testing is a valuable one, since it creates flexible and useful test automation from practically first day of development.

Anand, S., *et al*. [18] presented a survey of some most prominent techniques of automated test data generation containing figurative execution, combinatorial, model-based, adaptive random and search-based testing. Their main goal was to give an initial, up-to-date and short overview of research in automatic test case generation while ensuring comprehensiveness and validity.

Ingle and Mahamune [19] conducted a literature survey on automatic test case generation using UML diagram. They synthesized their literature survey on the basis of two main categories naming specification base test case generation and model based test case generation.

Dias Neto., *et al.* [20] conducted a systematic review based upon model based testing (MBT) approaches. A selection criteria was used to narrow down the initially identified papers. Their detailed study shows where MBT approaches have been applied, their characteristics and limitations. Comparison measures comprise on representing models, provision tools, test coverage criteria, level of automation, intermediate models and the complexity.

Khandai, M., *et al.* [21] presented a literature survey on various methodologies available for generating test cases from UML designs for parallel as well as non-concurrent systems. Summary of the work done in that field and its advantages/disadvantages have been presented. Literature survey helps the researcher attentive in the ground of test case generation from UML model to find out what work has been done in their interested field.

From above literature, it has been spotted that in the majority of surveys, the prime focus was on reviewing automated test case generation based on UML diagrams. After conducting reviews, most of the studies classified test case generation techniques in different categories. Some of those categories are model based test case generation, specification based testing, random testing techniques and UML diagram using evolutionary algorithms. The studies presented generic surveys on UML based test case generation techniques without considering a specific UML diagram.

3. Methodology

In this study, our research methodology is Systematic Literature Review (SLR). A systematic literature review is a mean of recognizing, estimating and understanding entire accessible research related to a certain research query, topic domain or case of attention. SLR includes activities such as *planning phase, conducting phase and reporting phase* [22] whole process shown in (Fig.1). Distinct studies aiding to a systematic review are called primary studies; but a systematic review is a type a secondary study.

The need for this systematic review (Step 1), the most common reasons are:

1) To summarize the existing literature evidence relevant to generating test cases through class diagram.
2) To find out any holes in present research to improve suggested parts for additional investigation.
3) To suggest an analysis in order to suitably place innovative research activities.

Though, systematic reviews can also be performed to inspect the level to which experimental proof supports/contradicts theoretical assumptions or even to support the generation of new hypotheses. An experimental search was piloted while keeping the following search string in Google Scholar, IEEE Xplore, and Springer Link and ACM digital library. The papers extracted through the search string were helpful in finding a direction for the development and validation of the desired search terms and the wanted protocol.
((“Test Case” OR “Test Case Generation” OR “Test Case Generation Techniques”) AND (“Class Diagram” OR “Class Based Testing” OR “UML Diagrams” OR “UML Class Diagram”) AND (“issues” OR “problems”)).

The research questions (Step 2) in section (3.1) indicate what should be extracted from the selected publications.

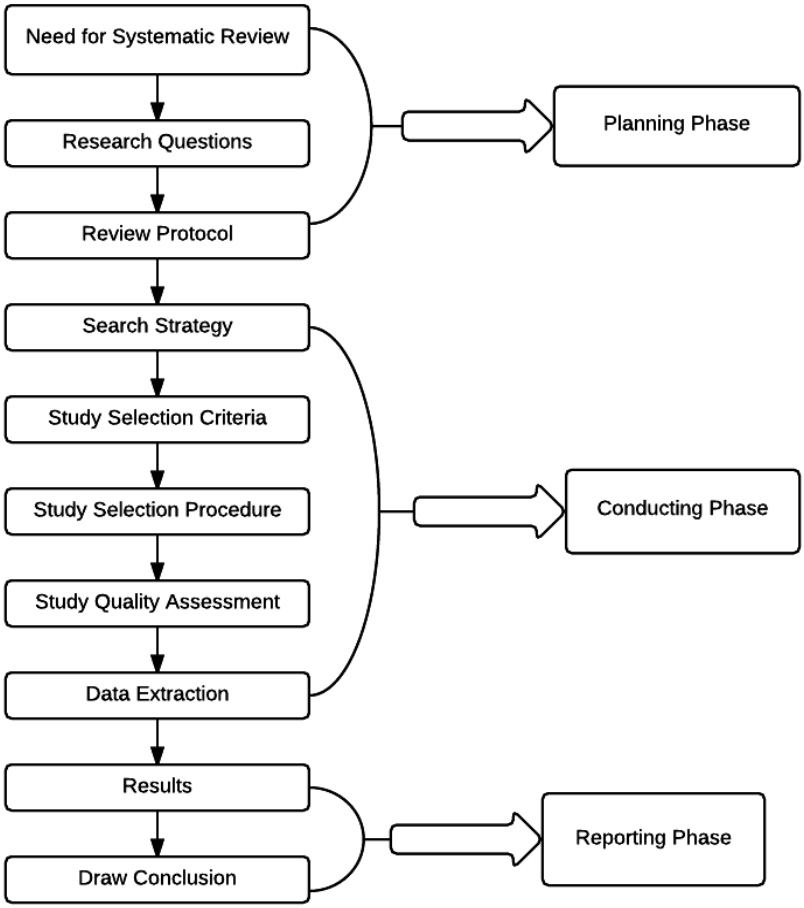


Fig. 1. SLR process.

3.1. Research Questions

The following search conditions inspired by (Tore and Torgeir, 2008) have been used to guarantee the excellence of the papers and to discount unrelated research papers. The research questions addressed through our study are:

RQ1. What kind of techniques has been proposed while generating test case from Class diagram?

RQ2. Are those techniques suitable for fruitful test case generation? What are the deficiencies to be worked upon in order to improve better test case generation?

The aim of the review protocol (Step 3) was to reduce potential researcher bias and to permit a replication of the review in the future (Kitchenham, 2007) [23]. The protocol was evaluated (Step 4) by scholars with the help of practice in performing systematic reviews. Depending upon feedback and our own gathered experiences during the progression, we iteratively upgraded the design of the review. The summary of the final protocol is given in sections 3.2 to 3.5.

3.2. Search Strategy

We followed the process shows in (Fig. 2) for the identification of papers. From our research questions, we derived the keywords for the exploration. So, as to authenticate the excellence of the search string, we piloted a trial search on Google Scholar, Science Direct and IEEE Xplore.

3.3. Study Selection Criteria

The key principle of primary study was to demonstrate the data that includes publications which are relevant to our keywords matching with defined search strings in order to answer the research questions

showing in (Step 2) (see Table 1). So, all papers of test case generation techniques using class diagram has been included. We excluded text that was not in English data, books and presentations. We neglected information that was not found against our search string. Also, data, which was not related to class based testing, study that did not satisfy test case generation techniques and studies that were not based on expert opinions are excluded.

Table 1. Study Selection Criteria

Selection of Study	Inclusion Criteria	No. of Papers left
Based on search string	Only English papers, Only published papers, 2000 to onwards	168
Based on title	No slides and chapters, no duplicates	98
	Part of title or fully	70
Based on abstract	Emphasize on class based testing, Test case using class diagram	51
Based on complete text	Investigate based on class diagram, Class based test case generation techniques	38

3.4. Study Selection Procedure

The systematic review technique was first introduced (Step 5) conducted in order to collect a similar analysis of the selection criteria among the three researchers which conducted the review. The selection criteria was applied on the title and abstract and if required on the full text of the publication of relative domain. For the pilot, we individually assessed 98 haphazardly nominated publications from a search undertaken in IEEE Xplore, ACM, Springer and Google Scholar.

We recognized the vague description of the selection principles and research questions on which the decision for inclusion was entirely based upon. After applying search string, we found total 38 papers having data related to class based test case generation techniques (see Graph 1). We discarded papers that were having focus on some other aspects and were not related to our domain. We collected typical required part from collected research papers to support our decisions towards success in generating test case techniques (see Fig.3). Furthermore, from selected papers we again read them and ensured that the selected papers are completely valid for evidences for test case generating from class diagram (see Table.2) as number of results per source and more results of evidence elaborated (see Graph.2).

3.5. Study Quality Assessment

The study quality assessment (Step 6) shows the quality level of our work. For all the questions, we found hardly very relevant work that totally supports our work. But through collected data, we facilitated our decisions and analysis. With QA1, we evaluated and performed analysis to support our findings. With QA2, we examined if the study provides sufficient material to provide the presented research.

3.6. Data Extraction

Similar to the assortment of study, we assigned the work between us. Data extraction (Step 7) was performed in an iterative way. Based on the experiences reported by [22], we expected that it would be difficult to establish a priority an exhaustive set of values for all the possessions. An initial extraction form was prepared with the properties like research method, context which shows also the mapping to the

respective research questions answered by the property (see Table 3).

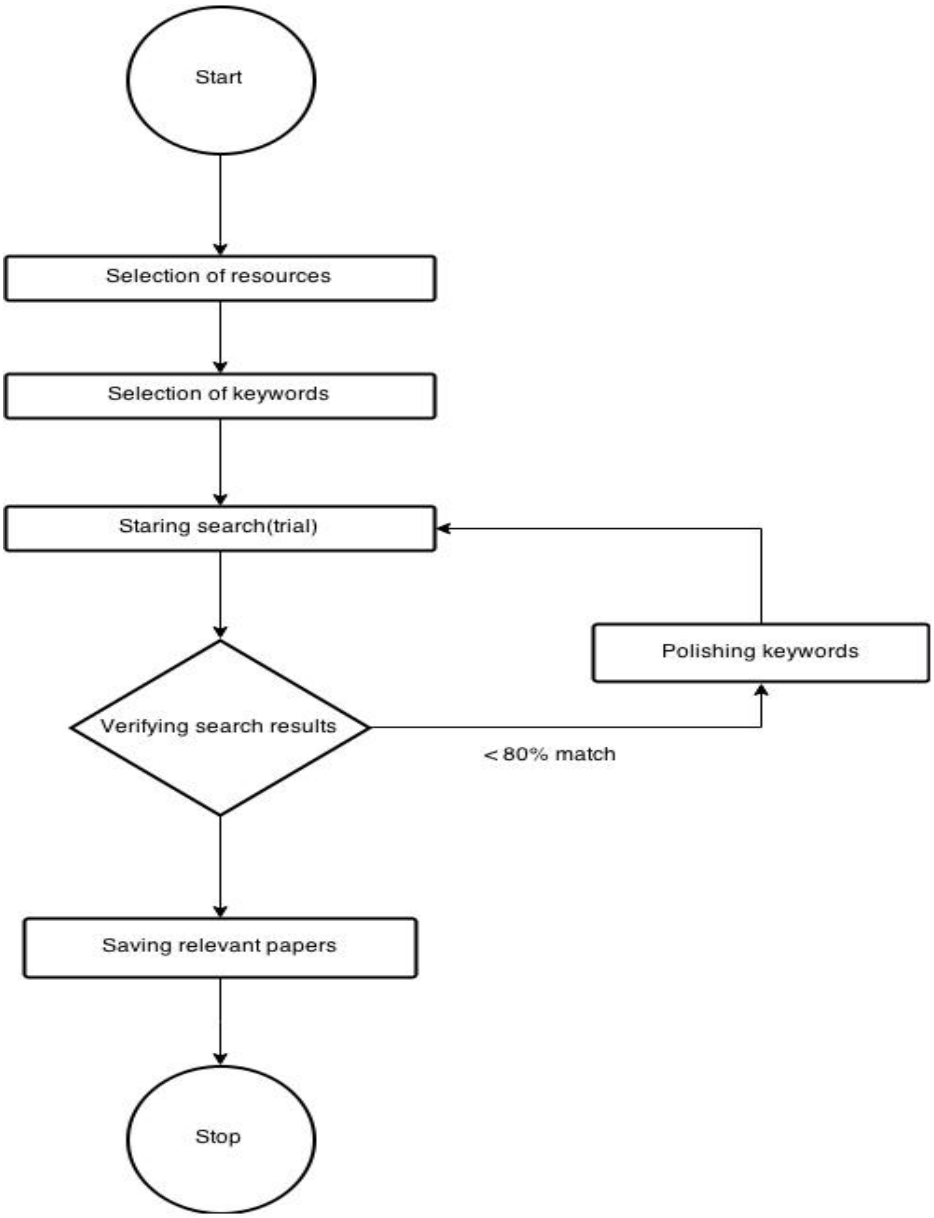


Fig. 2. Search strategy.

Table 2. Number of Results Per Sources

	ACM	**IEEE**	**Springer**	**Science Direct**	**Google Scholar**
Total Found	49	29	12	7	71
Candidate Studies	26	19	8	4	41
Primary Studies	9	6	2	1	20

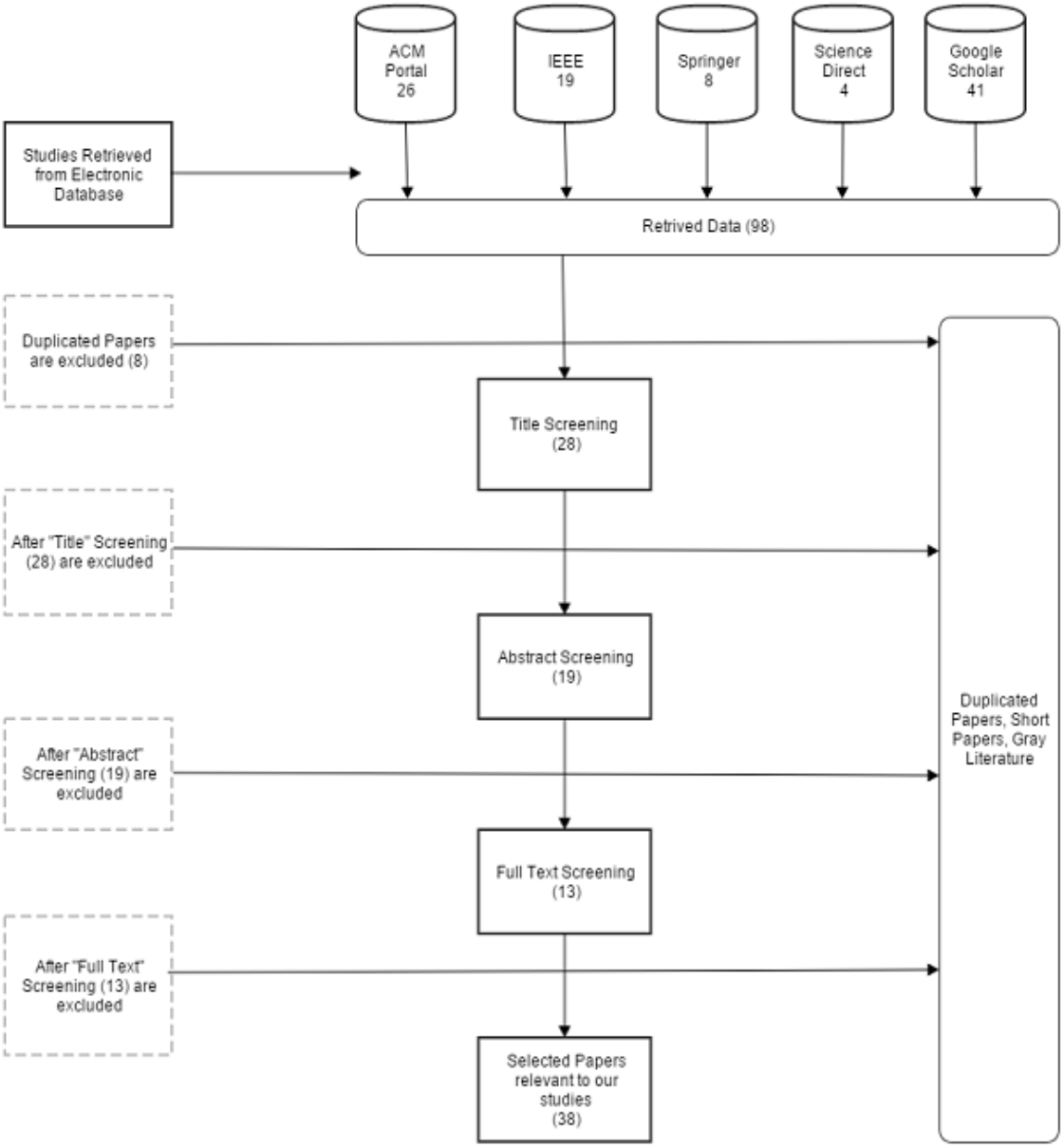
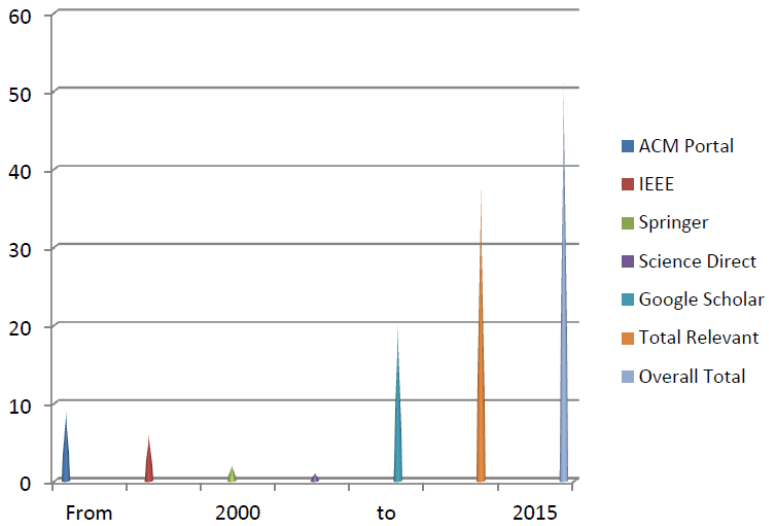
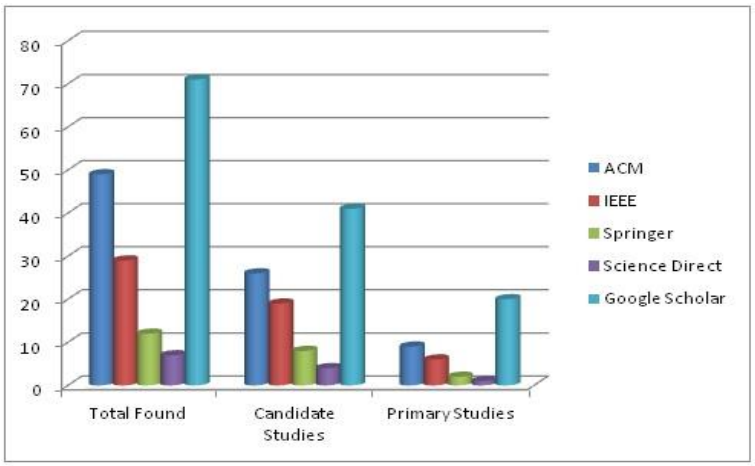


Fig. 3. Primary study selection.



Graph 1. Selected papers.



Graph 2. Number of results per sources.

Table 3. Data Extraction

#	Attributes	Research Question
1	Author / title /year / type of articles	Overview of study
2	Context	Overview of study
3	Research Methodology	SLR
4	Proposed techniques for generating test case using Class diagram	RQ1
5	Existing techniques useful for test case those generation	RQ2

4. Results and Analysis

RQ1. What kind of techniques has been proposed while generating test case from class diagram?

In order to address RQ1, we performed analysis to support our findings (see Table 4). Parameters of analysis were extracted from different studies and existing approaches were compared on the basis of following parameters while analyzing test case generation techniques. (1) Testing level [14], [15], and [20]. (2) UML model used [14], [16]. (3) Use of Intermediate forms [13], [25]. (4) Methodology used in a particular paper. (5) Provision of tool for test case generation technique [15], [20]. (6) Number of steps used by technique for test case generation [15], [16]. While generating test cases, use of intermediate form makes automation challenging, secondly, number of step while guaranteeing test cases also affect the level of automation that means smaller number of steps increase the level of automation whereas larger number of steps decreases automation level. Similarly, number of diagrams used for test acceptability surely increases the number of steps for test case generation. By analyzing the existing approaches, we have initiated some desired parameters which need to be considered while developing test case generation tool. That are: (1) Suitable path needs to be taken in an order to make automation easier. (2) Accurate reading of diagrams should be available. (3) Need to use minimum complex intermediate form or no intermediate form. (4) Small number of steps should be undertaken to generate test cases. (5) Use of no or minimum number of diagrams with class diagram while generating test cases.

Table 4. Analysis of Selected Studies

Title	Year of publication	Testing level [14], [15] [20]	UML Model used [14], [16]	Intermediate form [13], [25]	Methodology	Provision of Tools [15], [20]	No. of activities performed [15], [16]
Test Case	2012	Integration	Class diagram,	No	Case study	No	3

Generation from UML Models [9]		testing	interaction diagram				
Automatic Test Case Generation for UML Class Diagram using Data Flow Approach [10]	2009	System Testing	Class diagram	Directed flow graph	Case study	Yes	4
Automatic Test Case Generation from UML Models and OCL Expressions [24]	2007	Integration testing	Class diagram, state machine	Control flow graph	Case Study	No	6
A Novel Approach to Generate Test Cases Using Class And Sequence Diagrams [25]	2010	Integration testing	Class diagram, sequence diagram	No	Case study	Yes	4
AUTOMATED TEST CASES GENERATION FOR OBJECT ORIENTED SOFTWARE [26]	2011	System testing	Class diagram	Binary tree	Exploratory	Yes	5
Automated Test case generation using UML diagrams based on behavior [27]	2014	Regression Testing	Class Diagram, Sequence diagram and State chart Diagram and Use case Diagram.	Petal file	Case study	Yes	5
Approaches for Generating Test Cases Automatically to Test the Software [28]	2013	Integration testing	Class Diagram, Sequence Diagram	Tree, Binary trees.	Exploratory	No	5
Automatic Generation of	2011	Integration testing	Class Diagram,	Sequence diagram graph	Case Study	No	6

Test Cases from UML Models [29]			Sequence Diagram, Use case Diagram				
Generation Of Automated Test Cases Using UML Modeling [30]	2013	Integration testing	Class Diagram, Sequence Diagram, Use case Diagram	Control Flow graph	Exploratory	No	10
An Approach to Integration Testing of Object-Oriented Programs [31]	2007	System testing	Class Diagram, Sequence Diagram	No	Case Study	Yes	4
Automatically Generating Test Cases Using UML Structure Diagrams [32]	2009	Integration testing	Class Diagram, Object Diagram	No	Exploratory	Yes	6
Automated Test Data Generation Using Fuzzy Logic-Genetic Algorithm Hybridization System for Class Testing Of Object Oriented Programming [33]	2013	System testing	Class Diagram	Object Tree	Case Study	Yes	6
Dataflow test case generation from UML Class diagrams [34]	2013	Integration testing	Class Diagram, State Diagram	DD Graph	Case Study	Yes	5
A Novel Approach for Scenario-Based Test Case Generation[35]	2008	Integration testing	Activity Diagram, Sequence Diagram, Class Diagram	Class and Sequence diagram	Case Study	No	6

Deriving Input Partitions from UML Models for Automatic Test Generation [36]	2007	Integration testing	State Diagram, Class Diagram, OCL expression	Test Case Tree	Exploratory	Yes	4
Using UML for Automatic Test Generation [37]	2002	Regression testing	Class diagram, object diagram, and state diagrams	State machine	Case Study	Yes	5
A Systematic Approach to Generate Inputs to Test UML Design Models [38]	2006	System testing	Class diagram, Sequence diagram	Variable Assignment Graph	Case Study	Yes	5

In order to perform SLR, we have selected 17 studies from which only 3 used solely class diagram, 8 of them used class diagram and one supporting diagrams (like sequence, interaction, use case, activity) and 6 used more than one diagrams along with class diagram for test case generation. Similarly, out of the total number of studies, majority developed a tool for their framework and few of the studies did not develop any tool to support their framework. It has been observed that out of 17 studies, only 4 did not use any intermediate form while rest of them used some kind of intermediate form (Control flow graph, Binary tree, Directed flow graph etc). While generating test cases, among total number of studies, 11 papers used case studies as their methodology while rest of 6 were exploratory level studies. Integration level testing was performed by 10 studies while 5 performed system level testing and rest of two used regression testing. 5 papers used 6 or more number of steps, 6 papers used 5 steps, 4 papers used 4 steps and only 1 study used 3 Step in order to complete test case generation process.

RQ2. Are those techniques suitable for fruitful test case generation? What are the deficiencies to be worked upon in order to improve better test case generation?

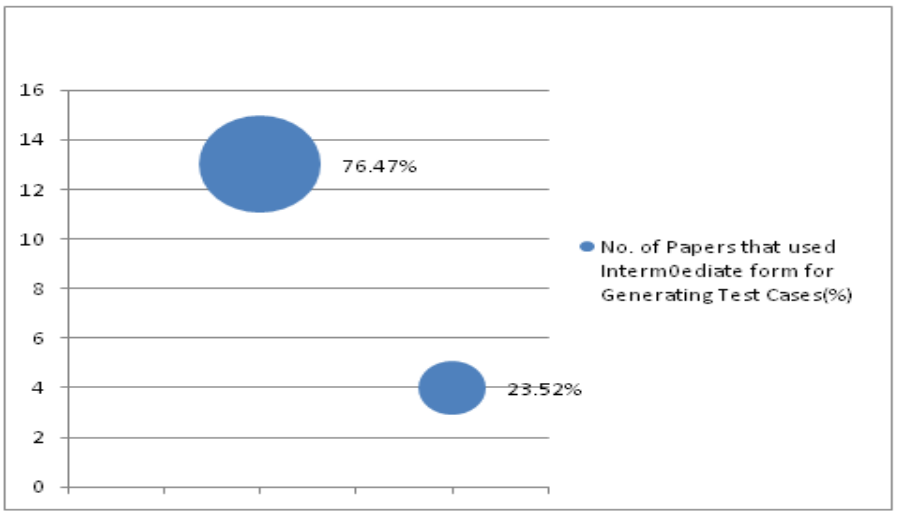
With the aim to answer RQ2, we briefly discussed the deficiencies of techniques in our study. We found the answer of this question after getting SLR results. Researchers used a lot of techniques to generate test cases for their feasibility and usage with the support of Class diagram. No doubt, their techniques are useful but there were some weaknesses found in their approaches that still need to be more refined. No one has proposed such a solution that claims to cover test case generation process efficiently with maximum accuracy. By using Class diagram, certain level of test case generation has been achieved but cannot be declared as an ultimate solution for the desired purpose.

Solely, class diagram is not enough for test case generation because it is a structural diagram. Behavioral diagrams have been used along with Class diagram to achieve test case generation phenomenon (see Table 5). It has been observed that most of the studies used an intermediate form to achieve automated test case generation that lessens the test case generation process. In this study, we have showed the number of papers that used intermediate form and also those which did not use any intermediate form (in percentage

form) while generating test cases (see Graph 3). Similarly, from the total number of studies, majority developed a tool for supporting their framework (see Table 6). Most of the researchers have gone through four or more numbers of activities to perform test case generation that is a time consuming and challenging activity, those studies can be seen along with references (see Table 7). Every activity takes time to get it done and acts as an input for the next activity. Decreasing the number of activities results in saving time and efforts. It should be kept in mind that decreasing the number of steps doesn't affect the performance of test case generation process.

Table 5. Diagrams for Generating Test Cases

Name of Diagrams	No. of Papers	References
Class	3	[10], [26], [33]
Class diagram and one supporting diagram	8	[9],[24],[25],[28],[31],[32],[34], [38]
Class diagram and more than one supporting diagram	6	[27],[29], [30], [35], [36], [37]



Graph 3. Number of studies that used intermediate form.

Table 6. No of Techniques Provided Tools

No. of Studies with References	Supported Tools	Percentage (%)
[10], [25], [26], [27], [31], [32], [33], [34], [36], [37], [38]	Yes	65%
[9], [24], [28], [29], [30], [35]	No	35%

Table 7. Number of Studies That Used Number of Activities for Test Cases

No. of Papers with references	No. of Activities for generating test cases
[9]	3
[10], [25], [31], [36]	4
[26], [27], [28], [34], [37], [38]	5
[24], [29], [32], [33], [35]	6
[30]	10

5. Discussion

In the literature, we have reviewed and spotted that 76% of the techniques used intermediate form for generating test cases and 23% of the proposed techniques did not use any intermediate form. Approaches, that didn't use intermediate form couldn't provide any clear picture about their generated test cases. In most of the techniques, it's been observed that their number of activities ranges from 4 to 6 for generating test cases. It has also been perceived that most of the techniques used one or more supporting diagrams with class diagram in order to generate test cases. Similarly, out of total number of studies, 65 % studies developed a tool for their framework and 35% did not provide any tool support for their studies.

From the literature survey, it has been analyzed that use of one or more intermediate forms, use of one or more supporting diagram with class diagram and large number of steps/activities has challenged the level of automation in a fruitful test case generation process. So far, there is no technique that claims to generate test cases in an optimal way and still, there is a rich space available for researchers to work in this area. If anyone can address above mentioned challenges or triggers down their intensity, then it would be a way forward step towards improvement in automated test case generation.

6. Conclusion

Unified Modeling Language (UML) has gained significant importance in the field of software testing due to its importance and features. Keeping in mind the UML class diagram, detailed analysis has been performed. Whole study was aimed to address two research questions and both are concluding respectively.

As far as RQ1 is concerned, various studies have been selected and precisely discussed according to literature. It was desired to identify techniques that considered class diagram to achieve focused test case generation. It's been spotted after passing through different approaches that UML diagrams like activity, use case, sequence and state chart have been used along with class diagram for generating test cases by using any of the testing types such as integration level testing, system level testing and regression level testing.

While addressing RQ2, it has been observed that studies proposed various techniques to express their ideas . After thorough investigation, we came to know that researchers used different approaches in order to generate test cases with the support of Class diagram. RQ2 aimed to investigate limitations of existing techniques so that future directions can be indicated for the research purpose. Some complications have also been highlighted such as maximum usage of intermediate form results in decreasing the level of automation. Secondly, huge number of steps/activities is also an overhead as it takes much time to perform whole test case generation process. It has also been observed that few techniques proposed a tool that used class diagram to generate test cases.

In future, we are planning to develop an automated tool for generating test cases through class diagram in order to minimize the issues mentioned in this paper. That tool will generate test cases directly from class diagram with fewer steps and to obtain better results, which will fulfill the objective of consuming less time, effort and improve quality with no intermediate form.

References

[1] Mall, R. (2013). Fundamentals of software engineering. *International Journal of Computer Science and Informatics*, *3(2)*, 14-21.

[2] Swain., *et al.* (2013). Generation of test cases using activity diagram. *International Journal of Computer Science and Informatics*, *3(2)*, 1-10.

[3] Shanthi, A. V. K., & Kumar, G. M. (2012). Automated test cases generation from UML sequence diagram.

International Journal of Computer Science and Application, *41*, 83-89.

[4] Ali., *et al.* (2007). A state based approach to integration testing based on UML models. *Journal of* information *and Software Technology*, *49*, 1087-1106.

[5] Gourlay, J. S. (1984). Introduction to the formal treatment of testing, software validation. *Proceedings of the Symposium on Software Validation* (pp. 67-72).

[6] Beizer, B. (2003). *Software Testing Techniques*. ACM.

[7] Shukla, S. G., & Chandel, G. S. (2012). A systematic approach for generate test cases using UML activity diagrams. *International Journal of Research in Management and Technology*, *2*, 469-475.

[8] Shirole, M., & Kumar, R. (2013). UML behavioral model based test case generation: A survey. *Software Engineering Notes*, *38*, 1-13.

[9] Wang, Y., & Zheng, M. (2012). Test case generation from UML model. *Proceedings of the 45th Annual Midwest Instruction and Computing Symposium* (pp. 1-8).

[10] Prasanna., *et al.* (2011). Automatic test case generation for UML class diagram using data flow approach. *Academia Education*, 1-7.

[11] Karambir, & Kaur, K. (2013). Survey of software test case generation techniques. *International Journal of Advanced Research in Computer Science and Software Engineering, 3*, 937-942.

[12] Kaur, K., & Chopra, V. (2014). Review of automatic test case generation from UML diagram using evolutionary algorithm. *International Journal of Inventive Engineering and Sciences*, *2*, 17-20.

[13] Pahwa, N., & Solanki, K. (2014). UML based test case generation methods: A review. *International Journal of Computer Applications*, *95*, 1-6.

[14] Tahiliani, S., & Pandit, P. (2012). A survey of UML-based approaches to testing. *International Journal of Computational Engineering Research, 2*, 1396-1401.

[15] Kaur, A., & Vig, V. (2012). Systematic review of automatic test case generation by UML diagrams. *International Journal of Engineering Research and Technology*, *1*, 1-17.

[16] Hooda, I., & Chhillar, R. A review: Study of test case generation techniques. *International Journal of* Computer *Applications* (pp. 33-37).

[17] Prasanna, M., *et al*. (2005). A survey on automatic test case generation. *Academic Open Internet Journal*, *15*, 1-6.

[18] Anand, S., *et al*. (2013). An orchestrated survey of methodologies for automated software test case generation. *Journal of Systems and Software*, *86*.

[19] Ingle, S. E., & Mahamune M. R. (2015). An UML based ssoftware automatic test case generation: Survey. *International Research Journal of Engineering and Technology*.

[20] Dias, N., *et al.* (2007). A survey on model-based testing approaches: A systematic review.

[21] Khandai, M., *et al*. (2011). A survey on test case generation from UML model. *International Journal of Computer Science and Information Technologies*, *2*, 1164-1171.

[22] Staples, M., & Niazi, M. (2007). Experiences using systematic review guidelines. *J. Systems and Software*, *80*, 1425-1437.

[23] Kitchenham, B. A. (2007). Guidelines for performing systematic literature reviews in software engineering. *Software Eng. Group.* Keele Univ. and Dept. of Computer Science, Univ. of Durham, UK.

[24] Weißleder, S., & Sokenou, D. (2007). Automatic test case generation from UML models and OCL expressions.

[25] Asthana, S., Tripathi, S., & Singh, S. K. (2010). *A Novel Approach to Generate Test Cases Using Class and Sequence Diagrams*. Springer-Verlag Berlin Heidelberg.

[26] Shanthi, A. V. K. (2011). Automated test cases generation for object oriented software. *Indian Journal of Computer Science and Engineering*.

[27] Verma, A., & Dutta, M. (2014). Automated test case generation using UML diagrams based on behavior. *International Journal of Innovations in Engineering and Technology.*

[28] Kaur, P., & Kaur, R. (2013). Approaches for generating test cases automatically to test the software. *International Journal of Engineering and Advanced Technology*.

[29] Sawant, V., & Shah, K. (2011). Automatic generation of test cases from UML models. *Proceedings of the International Conference on Technology Systems and Management*.

[30] Sharma, A., & Singh, M. (2013). Generation of automated test cases using UML modeling. *International Journal of Engineering Research and Technology*.

[31] Li, Z., & Maibaum, T. (2007). An approach to integration testing of object-oriented programs. *Proceedings of the Seventh International Conference on Quality Software.*

[32] Name, W. (2009). *Automatically Generating Test Cases Using UML Structure Diagram*. University of Delaware, Newark, DE, USA.

[33] Mondal, S. K., & Tahbildar, H. (2013). 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*.

[34] Anbunathan, R., & Basu, A. (2013). Data flow test case generation from UML Class diagrams. *Proceedings of the IEEE International Conference on Computational Intelligence and Computing Research* (pp. 1-9).

[35] Biswal, B. N., Nanda, P., & Mohapatra, D. P. (2008). A novel approach for scenario-based test case generation. *Proceedings of the International Conference on Information Technology (ICIT)*.

[36] Weißleder, S., & Schlingloff, B. H. (2007). Deriving input partitions from UML models for automatic test generation. *Proceedings of the Workshops and Symposia at Models.*

[37] Cavarra, A., Jeron, T., & Hartman, A. (2002). Using UML for automatic test generation**.** *ISSTA*. Italy.

[38] Dinh-Trong, T. T., Ghosh, S., & France, R. B. (2006). A systematic approach to generate inputs to test UML design models. *Proceedings of the 17th International Symposium on Software Reliability Engineering*.

Syed Asad Ali Shah was born in Islamabad, Pakistan. He is a research student. He received his BS (CS) from PMAS Arid Agriculture University Rawalpindi, Pakistan in 2012. He is doing a MS (CS) from same University. His research area is software engineering and software development.

Raja Khaim Shahzad is a research student. He received his BS (CS) from PMAS Arid Agriculture University Rawalpindi, Pakistan in 2012, He is doing a MS (CS) from the same University. His research area is software engineering, requirement engineering and agile development.

Syed Shafique Ali Bukhari is a research student. He received his BS (CS) from PMAS Arid Agriculture University Rawalpindi, Pakistan in 2012, Now, He is doing a MS (CS) from the same University. His research area is software engineering and web development.

Nasir Mehmood Minhas received his MS in computer science from COMSTATS Institute of Information Technology, Islamabad Pakistan, in 2006. He is a Ph.D candidate from Capital University of Science & Technology, Islamabad Pakistan. He is an assistant professor at University Institute of Information Technology, PMAS-Arid Agriculture University Rawalpindi, Pakistan. His research interests are software requirement engineering, software process/software process improvement, global/distributed software development and model based testing.

Mamoona Humayun has done her Ph.D. in computer science from Harbin Institute of Technology, Harbin China, in 2014. She is an assistant professor at University Institute of Information Technology, PMAS-Arid Agriculture University Rawalpindi, Pakistan. Her research interests are global software development, requirement engineering, and knowledge management and web application security vulnerabilities.