Automated Regression Test Case Generation for Web Application: A Survey

NISHANT GUPTA, MGM College of Engineering & Technology, Noida, India
VIBHASH YADAV, Rajkiya Engineering College, Banda, India
MAYANK SINGH, University of KwaZulu-Natal, Durban, South Africa

Testing is one of the most important phases in the development of any product or software. Various types of software testing exist that have to be done to meet the need of the software. Regression testing is one of the crucial phases of testing where testing of a program is done for the original test build along with the modifications. In this article, various studies proposed by the authors have been analysed focusing on test cases generation and their approach toward web application. A detailed study was conducted on Regression Test Case Generation and its approaches toward web application. From our detailed study, we have found that very few approaches and methodologies have been found that provide the real tool for test case generation. There is a need of an automated regression testing tool to generate the regression test cases directly based on user requirements. These test cases have to be generated and implemented by the tool so that the reduction in the overall effort and cost can be achieved. From our study, we have also found that regression testing for web applications was not investigated much, but in today's scenario web applications are an integral part of our daily life and so that needs to be tested for regression testing.

CCS Concepts: • **Software and its engineering** → **Software testing and debugging**;

Additional Key Words and Phrases: Regression testing, Test Case Generation, Test case Prioritization, Web Application

ACM Reference format:
Nishant Gupta, Vibhash Yadav, and Mayank Singh. 2018. Automated Regression Test Case Generation for Web Application: A Survey. *ACM Comput. Surv.* 51, 4, Article 87 (August 2018), 25 pages.
https://doi.org/10.1145/3232520

1 INTRODUCTION

Software is modified/changed due to customer modifications in the requirement, improvement, defects corrections, and change in environment. Software programs should perform the same way as specified by the customers after changes are incorporated in them. Regression testing is done whenever there are modifications introduced in the software, and it is done to ensure that the whole system is performing the same operations as it was doing before without any defect. It is performed on the subset of the program on which change is applied. The quality of any software product depends on its testing. This type of testing is done when the test build is available for

Authors' addresses: N. Gupta, Dept. of Computer Science & Engineering, Mahatma Gandhi Mission's College of Engineering & Technology, A-09, Sector 62, Noida (U.P), 201301, India; email: nishantlira@gmail.com; V. Yadav, Dept. of Information Technology, Rajkiya Engineering College, Banda (U.P), 210201, India; email: vibhashds10@yahoo.com; M. Singh, Dept. of EECE, Howard College, University of KwaZulu-Natal, Durban 4041, South Africa; email: dr.mayank.singh@ieee.org.

https://doi.org/10.1145/3232520

reuse (Graves et al. 2001). Any change in the specifications of the software has to be tested for verifying its quality. The software development cost increases after the changes have been incorporated, as generating test case becomes difficult. It costs around 80% of the budget of testing and approximately one-third of overall software development cost (Chen 2002).

The web application requires special attention because of its security concerns, dynamic in nature, and because it is heterogeneous (Zarrad 2015). As every application is moving to the web, its security requires a proper testing. As E-commerce is widely growing on the web, the feature of an online payment system, which is the current demand of any organization and its business, is also increasing. Electronic payment has completely changed the transaction processing system by reducing the efforts and cost of resources, therefore the efficient testing of the web application is a needful requirement for secure transactions (Mounica 2016). The web application still requires considerable work for regression test case generation, as more detailed evaluation of the approaches will be required in the future (Qiu 2015).

Regression testing is the time taken and costly testing for the organizations, which is performed to test the modifications. The purpose of regression testing can be summarized in the following points (Vokolos 1997):

- to identify the bugs due to the changes or modification in the program,
- to find out that the changes incorporated will not produce new bugs,
- to find out that the changes done will not change the actual specification defined by the user.

The implementation of regression testing on any software program can be described in three types of phases: Test case Generation, Prioritization, and Execution (Khandai et al. 2011). In Regression test case generation, suitable test cases have to be generated. Prioritizations of these test cases are done so that the testing can be done in a minimal number of test cases and overall efforts and cost can be reduced.

The minimization of the test cases is required for minimizing the testing cost and the best approach has to be adopted so that maximum coverage is possible. The comparative study has been shown by Baradhi (1997), who showed that out of five techniques—slicing, genetic, incremental, simulated annealing, and firewall—incremental algorithms are faster than other algorithms for retesting the programs.

A study showed that early prioritization of test cases during the development process of software yields less effort for further investigation by the mangers and software testing team (Kahtibsyarbini 2017). For agile testing, prioritization can be achieved using clustering of the user stories based on similar modules, which has been already covered (Kandil 2017)

The cost of regression testing has to be minimized by automation of test case generation. Binkley (1995) has tried to reduce the cost by an algorithm based on semantic test case selection. The algorithm was using already executed test cases and the results for cost reduction. These previously executed test cases are reused in selecting new test cases. Automation requires a specific skill set, environment, and the automation tool (Hoffman 1999). Various cost benefit analysis can be adopted to find out the automation cost and its benefits. Return on Investment, ratio of efficiency of automated regression tests and the return on investments has been adapted to analysis the automation. Time Complexity, Total Error Detection, and Time to Deploy (Kadry 2011) were the main factors to find out the effectiveness of regression testing in terms of its cost.

A study on efficient regression testing (Eric 1997) showed a tradeoff between selection of the regression test cases, organization cost, and budget. They have proposed a hybrid-based technique where test cases have been generated based on modifications and then selection of these test cases using prioritization technique. The purpose is to find out those test cases that impact the

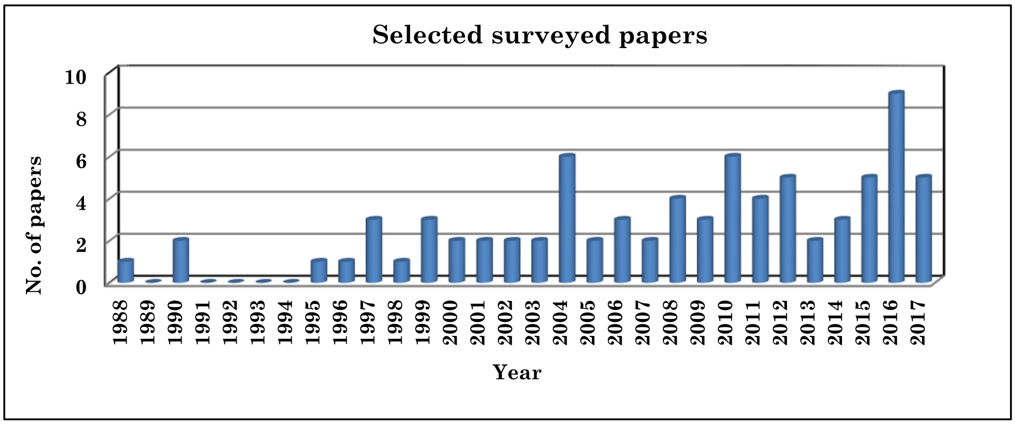

Fig. 1. Selected publications on Regression Test Case Generation and its approach to Software and Web applications (1988–2017).

output. The cost of regression testing depends on the characteristics that its test build possesses. Continuous changes in the Regression test data demands a lot of effort and cost. The continuous development and deployment is a new trend of software development. The cost of Software testing targets about 50% cost of overall software development (Gupta 2015). The agile development of software increases the business value but also increases the cost of development (Dingsoyr 2016). Based on the study of 61 projects, the efforts of using regression test cases has been analyzed in terms of cost and benefit (Labuschange 2017). Using Regression test cases is beneficial, but it may increase the overall cost of the development. Cost of the regression test cases may be reduced by using Adaptive test prioritization strategies (Schwartz 2016).

2 SELECTION OF PAPERS AND SCOPE

The complete survey has been done from 1988 to 2017 related to regression test case generation and its approach to web application. The selection of the papers for this survey is based on the regression test case generation either at requirement level, design level, or at the level of coding. This article considers the techniques and approaches that are related to the Regression test case generation for software applications and web applications. The motivation of this survey and the papers included in it has been chosen in keeping with their wide scope in software industries. There are various tools available for testing the applications and software system, but still the test cases are identified and generated manually, which is later implemented by the available tools. Initially, more than 486 papers were collected for providing a complete survey. The papers collected were, in general, related to regression testing. Out of these papers, 79 papers have been selected for this literature, based on the Regression test case generation for software applications and for web application. Papers have been searched in the main technical repositories, such as IEEE Explore, ACM Portal, Science Direct, Springer, Wiley Databases, and Google Scholar. Figure 1 represents the number of papers selected for their contribution in test case generation and their approach to web application.

The rest of the article is presented as follows. Section 3 of this study discusses an overview of regression testing. Discussion of regression test case generation and their approaches has been given in Section 4. Section 5 discusses test case Generation for Web Applications and the cost-effectiveness of regression testing using an automated tool. The gap in the existing literature that motivates the future work is given in Section 6. Section 7 includes Future Scope.

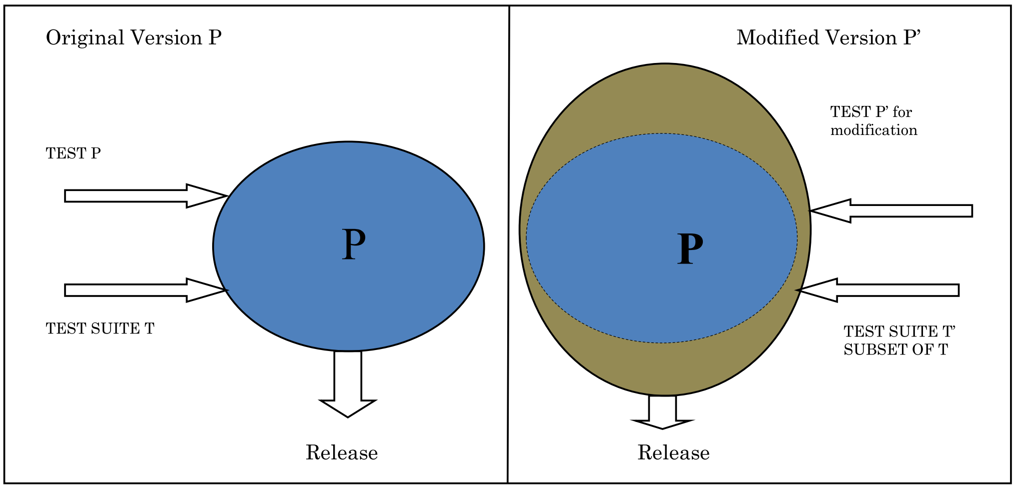

Fig. 2. Regression testing: A diagrammatic representation.

3 REGRESSION TESTING: AN OVERVIEW

A lot of cost is involved in regression testing. The testing is performed to confirm that the modified program behaves the same way as it did before the change is incorporated, and the actual requirements of the user remain unchanged. It is a difficult process, because we have to test the new build as well as the old build, keeping in mind the actual specifications. Regression testing can be understood in a simpler manner in Figure 2.

Rothermel et al. (1996) has described the problem of Regression test case generation as, "Given a set of original test suite T for original program P, Select T′ subset of Test T executed on Program P′, where P′ is the modified program."

In Figure 2, the test has to done with the subset of T on program P′ so that the modified program can be tested along with the confidence to confirm the original specifications. The prioritization problem is to find out a T′′, for new specifications or modifications to test the program P′, such that $T'' \neq T'$.

Regression testing can be better understood in Table 1, and in Tables 2 and 3, where the requirements and the tests cases for the initial as well as modified phases are shown in detail.

In Table 1, the requirements of the original and modified version of the program are shown. In the original program, we are adding two integers, but later, in the modified program, we have added one more requirement to multiply the integers.

In Table 2, test cases for the original program are shown. In Table 3, the actual scenario of regression testing has been shown where the test cases have been generated for original program along with the modified program.

The first test case here shows the old test case to add two integers. The second test case shows the test case related to the new feature, while the third test case shows the multiple operations on the integers. Table 3 shows the actual regression testing of a program in TC_002 and TC_003, which has to take care of the new feature along with the old features of the program.

The test cases for the initial phase are shown in Table 2.

The test cases have been created after the modification in the application along with test cases developed for the original phases and are shown in the Table 3.

The various test case generation techniques that have been proposed are Unified Modeling Language (UML) state diagrams, collaboration diagrams (communication diagram with UML 2.x),

Table 1. Requirements at Original and Modified Version

Requirements					
Initial Requirement v1.0					
S. No.	Requirements Title	Requirement Summary	Requirement Description	Business Rule No.	Business Rule
1	REQ_001	Add two integers.	Provide two integer values and on clicking "=" button will lead to sum of the integers.	BR_001	Integer Number with maximum addition limit: 2147483647 and minimum addition limit: - 2147483648
Updated Requirement v2.0					
S. No.	Requirements Title	Requirement Summary	Requirement Description	Business Rule No.	Business Rule
1	REQ_001	Add two integers.	Provide one integer value, choose "+" and then provide another integer value. On clicking "=" button will lead to sum of the integers.	BR_001	Integer Number with maximum addition limit: 2147483647 and minimum addition limit: - 2147483648
2	REQ_002	Multiply two integers.	Provide one integer value, choose "*" and then provide another integer value. On clicking "=" button will lead to multiplication of the integers.	BR_001	Integer Number with maximum multiplication limit: 2147483647 and minimum multiplication limit: -2147483648

Table 2. Test Cases: At Original Phase

Test Case						
S. No.	Title	Precondition	Description	Req._Trace	Test Case Steps	
Test Cases v 1.0					Test Steps	Expected Result
TC_001	Addition of Two Integers	1. Calculator must be displayed. 2. No value must be preselected.	Provide two integer values and on clicking "=" button will lead to sum of the integers.	REQ_001	1. Provide first integer value. 2. Choose "+" operator button. 3. Provide second integer value. 4. Click on "=" button.	1. First integer value is displayed in the display screen. 2. "+" operator button is selected. 3. Second integer value is displayed in the display screen. 4. The system will return sum of both the integers.

activity diagram, and sequence diagrams, as shown in Table 4. As Regression testing is widely implemented and used in industry, most of the test cases that we have studied for regression testing are based on UML, impacted area of new changes or new build, and well-defined path.

Test Case Generation for regression testing can be done for black box or white box methods (Sapna 2015). Black box methods target the change related to the functionality, which has to be tested, but white box method covers the code that has been affected due to the modifications. There is a method known as Gray method (Linzhang 2004), which is the combination of Black and

Table 3. Test Cases for Modified Version

Test Case						
S. No.	Title	Precondition	Description	Req._Trace	Test Case Steps	
Test Cases v 2.0						
TC_001	Addition of Two Integers	1. Calculator must be displayed. 2. No value must be preselected.	Provide one integer value, choose "+" and then provide another integer value. On clicking "=" button will lead to sum of the integers.	REQ_001	1. Provide first integer value. 2. Choose "+" operator button. 3. Provide second integer value. 4. Click on "=" button.	1. First integer value is displayed in the display screen. 2. "+" operator button is selected. 3. Second integer value is displayed in the display screen. 4. The system will return sum of both the integers.
TC_002	Multiplication of Two Integers	1. Calculator must be displayed. 2. No value must be preselected.	Provide one integer value, choose "*" and then provide another integer value. On clicking "=" button will lead to multiplication of the integers.	REQ_002	1. Provide first integer value. 2. Choose "*" operator button. 3. Provide second integer value. 4. Click on "=" button.	1. First integer value is displayed in the display screen. 2. "*" operator button is selected. 3. Second integer value is displayed in the display screen. 4. The system will return sum of both the integers.
TC_003	Multiple Operation on Integers	1. Calculator must be displayed. 2. No value must be preselected.	Apply several operations, i.e., sum and multiplication between numbers. On clicking "=" button will lead to final result.	REQ_002	1. Provide first integer value (say - 2) 2. Choose "+" operator button. 3. Provide second integer value (say - 3). 4. Choose "*" operator button. 5. Provide third integer value (say - 4). 6. Click on "=" button.	1. First integer value is displayed in the display screen. 2. "+" operator button is selected. 3. Second integer value is displayed in the display screen. 4. "*" operator button is selected and sum of the first two integers is displayed. 5. Third integer value is displayed in the display screen. 6. The system will return sum of both the integers.

White box approach. It covers coverage criteria and generates test cases for all the paths and then generates test cases to satisfy the path conditions and functionality for the black box method.

The problem of minimization of test case generation, their selection, and prioritization has been studied by Yoo (2012). They have covered all three aspects, which are useful for providing an efficient regression testing. They have given a complete study about minimizing the available test suite so that redundant test cases will not be executed again, resulting in the smaller test case suite. The selection of the test cases from modified test suite is having a similar purpose as minimization of test suite. The authors have also focused on prioritization of the test cases to order the test cases as per their priority of implementation.

Abdurazik et al. (2005) showed in their experiments that test data can be generated efficiently with the help of model-based testing. The tests have been generated using UML state charts and sequence diagrams. Both techniques have their advantages in finding out the faults. UML state charts have been shown to be good in finding faults at the unit level and sequence diagrams at the integration level.

4 TEST CASE GENERATION

A lot was proposed in the area of generating test cases for regression testing. Various techniques and methodologies have been proposed by the authors to generate test cases. Most of the work is being done using Unified Modeling Language, which commonly uses either activity diagrams or state diagrams. The following subsections describe the different techniques and approaches proposed by the authors.

4.1 Model-Based Coverage Approach

A model-based approach is a technique that derives test cases from the software models. It is a lightweight method used for the program correctness when generally incomplete approaches are available (Ananda 2013).

A model-based approach using collaboration diagram has been given by Abdurazik (2000). Test cases have been generated from the software design, and collaboration diagrams have used both static and dynamic testing in specification level and instance level. The focus of the authors was on the study of the behaviors of a set of interacting objects. The static checking has been referred to the testing of the source code, while the dynamic testing has been done by using collaboration diagram and the test criterion is defined as follows:

"For each collaboration diagram in the specification, there must be at least one test case *t* such that when software is executed using *t*, the software that implements the message sequence path of the collaboration diagram must be executed." Further study is required and proposed to implement software based on collaboration diagrams and to generate test cases automatically by UML collaboration diagrams.

Data-oriented Boundary coverage for test case generation has been proposed by Kosmatov (2004). The algorithm uses BZ testing tool for generating test cases from UML specifications, and test case is designed in such a way that it satisfies the given criteria.

Cavarra (2002) presented an architecture using UML for the model-based testing. The model that has been presented is compiled into the Intermediate format (IF). The IF is then used to generate the test cases. The work has been done for scalable methods for automation in test case generation. The first step toward the architecture was to make the system model that consists of class, object, and state diagram. The second phase is to form the test directive that consists of object and state diagram. Both system model and test directive are written in UML. The extended interacting state machine was produced by the compiler. The target language, IF is used for all the operations and data types. Three test directives were proposed by the authors: test constraints, coverage criteria, and test purposes, which have been used in the combination form to make the test directives. The AGEDIS test generation tool has been used for test case generation. The authors have used this tool because of its capability of combining all the test directives.

Riebish (2003) presented an approach to generate the test cases that was based on use case models. Further, they have been elaborated by state diagrams. The usage model has been created from the model to define usage and system behavior. The XML-based tool has been used for transforming into usage model. The author used the interactive software development process model, so that both source code and the architecture can be changed to cover the changes and constraints in the requirement. The complete approach can be defined in different steps as follows:

- Initially with the help of use cases, complete scenario of the required functionality is developed.
- In the next process, the refined use cases in the form of templates are converted into the state diagrams.
- In the next step, the state diagrams have been converted into usage graphs.

- After getting the usage graph, the usage model is obtained.
- Now the test cases can be generated using usage model.

The author in this approach has combined both scenario-based requirement and system behavior description for generating the test cases. The benefit presented by the author is reduction in testing efforts and support in requirement engineering.

The test case generation with the help of state charts and sequence diagrams has been given by Abdurazik (2004), which was experimentally evaluated for the fault detection. The software of standard cell phone has been modeled and taken for the experiment. The test cases were generated from the sequence diagram dependent on message sequence path coverage criteria. The full predicate coverage was used to create the state charts. Two software tools have been used to represent the cell phone system and implement the same in java: The TogetherControl Centre software and JBuilder tool, respectively. Experiment showed that state chart is better to find out the defects at unit level faults as compare to sequence diagram, but sequence diagram showed a better performance at integration level faults as compare to state charts. To generalize the results, more test sets and applications are required, and concurrency is needed in the execution to automate the testing process. The author in his future work plans to combine specifications with the Object Constraints Language (OCL) to give better test performance.

The approach for model-based test case generation has been proposed by Gnesi (2004). The formal test cases have been generated for conformance testing and UMLSCs (UML state charts). The approach used Input/Output Labeled Transition States (IOLTS) to develop a semantic model for the specifications and implementations. The algorithm designed for test case generation has been written with the combination of processed algebra and lambda calculus. The algorithm proposed is a non-deterministic algorithm. The proper strategies for the test selection have to be required for implementing the test case generation algorithm. The design of automatic tool in combination with the test case generation tool has been proposed by the author in his future work.

Bertolino et al. (2004) in his work proposed a coherent approach of test case generation. A reasonable reference model has been generated where sequence diagram and state diagrams were used for the purpose of generating test cases. The approach is divided into various steps:

- Generate a complete scenario specification “SEQ’’ using sequence and state diagram.
- Produce State machine “ST” from “SEQ.”
- Generate general scenario detection along with the resolution process.
- Produce reasonably complete model.

The approach given by the author is focused on low effort and more accuracy of testing. The author has tried to develop a more informative model, thus producing more accurate test cases.

Hartmann et al. (2004) proposed and focused on test case generation tools. The authors have focused on coupling the proposed techniques with the suitable tools. The test case generation has been done for the black box technique and based on UML diagrams. The approach is divided into two steps: component behavior modeling and test case generation. In the first step of modeling, state charts have been developed to show the behavior of the components. In test case generation, the test case has been generated using global behavioral model, where the transitions between and within components is considered as a default coverage. The test case generation has been generated using Test Generation Tool on the basis of data variations defined in the modeling phase. The authors tried to increase the effectiveness by using the efficient coverage criteria. Further improvement in the tool and empirical study has been planned by the authors.

The Model-based technique of Test case generation based on Labeled transition system has been given by Cartaxo (2007). The test cases were generated using all the paths obtained from labeled transition system. The labeled transition system has been made from the UML sequence diagram. The depth first search method was used to obtain the paths used for generation of test cases. The case study on mobile phone application has been illustrated for the proposed approach and showed that all functionalities have been tested under the coverage criteria. The quality of a model used has to be considered for the effectiveness of the approach, since the faults in UML sequence diagrams may affect the test cases.

For object-oriented programs, Swaina et al. (2012) have given a test case generation technique designed based on State and Activity models (SAD). Test cases are generated to focus state activity coverage of SAD by constructing state activity diagram. The technique given considers the test data selection manually by the testers. A prototype tool, Model JUnit, an open source tools has been used by Swain et al. (2012) to produce the UML design artifacts. A state chart graph is generated to find out the predicated. These predicated was transformed to source code so that Extended Finite State Machine (EFSM) can be constructed. Based on EFSM, test cases were generated.

Considering Use case diagram and sequence diagram, a technique for test case generation have been proposed by Mohapatra et al. (2010). The main focus was on the artifacts of the analysis stage and the specifications of the sequential constraints. Use case dependency analysis and test requirements were used to generate test cases. The technique proposed used XML for exchanging of UML models.

The Input/Output Explicit Activity Diagram (IOAD) model has been proposed by Kim (2007). In this approach, generation and minimization of test case is based on UML activity diagram and only external inputs and its outputs is exposed. The IOAD model has been used and converted in directed graph, so that the various test scenarios can be find out along with their test cases.

Another approach based on UML activity diagram is given by Kundu et al. (2009). The test cases are generated from an activity graph, which is derived from an activity diagram based on the specifications. They considered various coverage criteria like basic path, simple path and activity path for the test case generation. The proposed approach was capable of detecting location of fault in the implementation phase. The proposed technique only considered the activity diagram for one use case at a time.

Boghdady et al. (2011) proposed a technique based on Activity Diagram in which an Activity dependency graph and an intermediate table are generated for each activity diagram. For traversing depth first search has been and find out all the possible paths. Test cases are generated using test paths. A cyclomatic complexity technique is adopted for the validation.

Model driven and UML activity diagram-based approach has been proposed by Thanki et al. (2014). The technique focused on the duplicate test cases and aims to generate and minimize the test suite for the testing. The complete approach has been divided into three parts: test case generation for different domains, model transformation, and minimization of the test build. The test optimal tool has been used to implement the proposed approach using activity diagram. The steps for the proposed algorithm have been stated as follows:

- Editing of the UML diagram for the modifications and changes.
- The test cases have been generated as per the structural and semantic changes and categorized as changed and unchanged test cases.
- Filtration of Unchanged test cases.

Genetic algorithm has been used to generate test cases using techniques of UML Object diagrams (Prasanna et al. 2009). Test cases are generated with Genetic algorithm using model-based technique where object behavioral aspect is considered. Results have shown that the faults can be found at the unit level and integration level both.

The model-based GUI testing approach based on annotated use cases has been given by Navarro et al. (2011). GUI test cases has been automatically generated and validated from the annotated use cases. The approach does not rely on a complete model but instead it models test cases and annotated elements and these elements are defined as:

- A set of Use cases: behavior of the GUI is described.
- A set of Annotated Elements: to represent the GUI elements with the variable values.

The execution was done on an application of fixed-term deposit calculator and the validation was performed based on assert oracles and state oracles. The process generates the test case for every possible path.

The test case generation UML Models using class diagram, use case diagram, and sequence diagram has been proposed by Sawant (2011). The sequence diagram graph has been created that stores the necessary information so that the test cases can be generated. The UML diagram has been converted to XML format to create the graph and the scenarios. The approach has been implemented using the case study of Bank ATM system.

A model-based approach has been proposed by Wang (2012), in which the test cases have been generated from design level class diagram. Also, the interaction diagrams are used in the methodology, which includes collaboration and sequence diagram. The test case generation has been illustrated using a case study of a car rental example. The class diagram has been generated for the car rental and based on Association-end-Multiplicity criteria; the test cases have been created.

In the study done by Asad (2015), various approaches have been studied that exercises class diagram to support their test case generation activities. The study was focused on the test case generation using class diagram. They have showed that only 76% of the techniques are using intermediate form while generating the test cases.

Test cases are generated has been generated using Meta model with the help of test model and structural model (Saifan 2016). Content can be dynamically modified whenever required using the meta model. Lack of experimental evaluation to validate the approach in finding the defects. UML with finite state machine (FSM) has been used by Patil (2017), where it covers both paths and conditions by using both UML and FSM. The technique required an approach that can find our defects in less time and effort. A proper automated tool support and case study-based evaluation is required. Sahoo (2017) proposed an approach based on honey bee colony algorithm. The idea is to optimize the test cases and finding out the test paths using combinational system graph. Test cases optimization may not be benefitted for the purpose of testing the components.

4.2 Specification Based Approach

Offut et al. (1999) proposed a state-specification-based technique for the generation of test inputs and the test cases. These test cases and the specifications are generated from UML State-Charts. The techniques covered were "full predicate test, transition pair tests, transition coverage, and sequence coverage." A tool for test case generation "UML TEST" was used along with the Rational Rose and a case study was also implemented for Cruise Control System. The coverage for each technique is defined as:

- In a transition coverage level, the transition coverage is defined as "the test set T, which should satisfy each transition in the specification graph SG."
- The full predicate coverage is defined as "for every predicate P for each transition, the test set T should have the tests that reflect each clause c in P to the result in a pair of outcomes where the values of P is directly correlated with the value of c."

- In a transition-pair coverage level, it is stated and defined as "for each pair of adjacent transitions S_i: S_j and S_j: $S_{k,}$in SG, T consist of test that traverse the pair of transitions in sequence."
- In a complete sequence level, the meaningful sequences of transitions have to be defined by the test engineer. In a further work, they have planned to compare each testing technique with multiple programs and cover all UML aspects.

Based on coverage criteria in UML state diagrams, two approaches, control flow and data flow, are used for generating test cases (Kim and et al. 1999). Control flow is formed using an Extended Finite State machine is made (EFSMs) and data flow is formed by converting EFSMs to flow graph. Limitation of this approach is that it can be used for unit testing and needed an automated tool to support the class testing.

Chen (2002) worked on a model used for stating requirements depending on the customer behavior and activity diagram. The approach was based on two basis regression tests, "targeted tests and safety tests." Targeted tests were used to ensure that all the current features are also supported by the new release and safety test are performed to cover the most risk associated problems. The cost has been evaluated on two different parameters related to both customer and vendor: effect of loss in market and high maintenance respectively. Test cases with high risk probability and risk exposure have been selected. The results of risk-based approach showed improvement over manual test cases. The authors divided the approach into four parts:

- Find out the cost of the requirements attributes.
- Derive high intensity probability for every test case.
- Find out risk exposure for every test case.
- Select Safety test.

The validity was done on the factors of effectiveness, cost efficiency, and sensitivity to risk.

An approach based on specifications is given by Alhroob et al. (2010). Software specifications are designed using unified Modeling Language. A class diagram has been used for representing the classification and its classes mentioned in the specifications. The aim is to reduce the redundancy in the test cases by the restructured algorithm. Integrated classification tree methodology has been used and improved to discover specifications. These specifications are then used for generation of test cases.

Sanj (2012) have proposed a specification-based approach for functional test case generation using UML Activity diagram and use cases specification. Test cases have been also prioritized based on the software risk. The test case generation approach was based on the Activity Diagram (AD) and the AD is defined as:

"G = (N, T) where
N= NA U ND U NF U NJ U {start point, end point}
where NA= {graph nodes representing Activities in the AD},
    ND= {graph nodes representing Decisions in the AD},
    NF = {graph nodes representing Forks in the AD},
    NJ = {graph nodes representing Joins in the AD} and
T (Transitions in the AD) = (edge, guard)"

The experiment was done on DVD collection software application with four defects that was intentionally inserted. The test case generation method has been validated by 70 software experts of different profiles. A tool for automation has been also developed using XML file and Eclipse Environment.

4.3 Functional Based Approach

A Gray-Box method-based approach by Linzhang (2004) has been proposed for generation of test cases using UML Activity Diagram. The DFS algorithm has been used to traverse each activity states and transitions of every path. All the test scenarios were generated using activity diagram. The Gray method used can describe the external system behavior as well as the code structure to generate the test cases.

Steiner tree algorithm has been used by Sapna (2015) that can generate test cases from UML diagram to check the functionality of the system. The specifications are represented by UML. The method is benefitted as the changed nodes can be added at the terminal end. The method has been tested for the smaller functionality, hence needed to test its efficiency on bigger one. A case study of the vending machine has been taken into consideration to produce graph CFG, G. The minimal path is being found after applying the Steiner tree algorithm.

4.4 Slicing Based Approach

Regression test case generation based on Component Slicing was given by Lalchandani (2008). Software architecture is transformed into Architecture Component Dependence Graph (ACDG) and the algorithm works on in service and out service edges of ACDG. A prototype tool, SRTWA is used to implement the algorithm.

Dynamic Slicing for cluster level testing has been implemented by Samuel et al. (2008). The technique is dependent on Dependence graph, which is generated from UML sequence diagrams. Test cases have been generated from slices that were created using Edge marking Dynamic method. An approach based on Dynamic Slicing has been proposed by Korel (1988). The technique requires an input for computing the slices and generating the path to make relations between Test Control and Data-Data. The technique is not dependent on CFG instead it works and implemented on the program path. The dynamic slicing here produces smaller slices as compared to the static slicing. Another approach based on dynamic slicing has been given by Agrawal (1990). The approach is based on Dynamic Dependence Graph where it produces a node for every statement occurred previously including already executed statements.

Horwitz et al. (1990) have given the technique based on System dependence graph and computes forward slices for interprocedural program. Another approach based on Conditioned program slicing has been given by Canfora et al. (1998). The conditioned program slicing works as an interface between static and dynamic approaches. The technique works on predefined initial condition. It is defined as "a triple, (p, V, n) where p is some initial conditions and (V, n) are the two elements of the static slicing criterion."

4.5 Some other Approaches

An approach using Tabu Search Algorithm has been given by Shanthi et al. (2012) to generate test path using Unified Modeling Language. The purpose is to generate test paths using Activity Dependence table. Test paths have been prioritized by Tabu Search Algorithm and are capable of finding faults in loops along with their location in implementation. The test suite generation based on metamorphic relations and fuzzy logic has been implemented by Bandaru et al. (2016). The purpose of the technique was to focus on the oracle problem by covering all the abstract states.

Based on Business Process Execution Language (BPEL), an approach is proposed by Li (2016). The test cases that are covered in regression testing have been identified by the following steps.

- The data flow has been analyzed with the help of Extended Control Flow Graph model (XCFG).

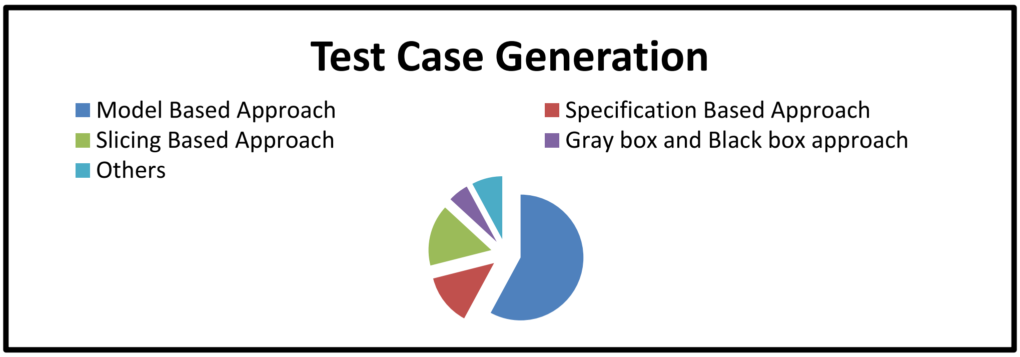

Fig. 3. Various Approaches Surveyed for Test Case Generation.

- The impacted data flow pair identification, which is affected by process, binding, and interface change.
- Make complete XCFG paths.
- Select Regression tests cases with the help of path condition analysis.

The approach is given to find out the affected data flow, which can be due to process, binding, and interface change. Test case generation still has to be done, which is limited to single version in the given approach.

4.6 Summary

Test Cases for regression testing are generated based on impacted area of new changes, i.e., new build, well-defined paths, i.e., old build, knowledge-based, risk-based, and impact-based. Based on above factors, test cases can be generated and implemented.

The various approaches have been adapted to generate regression test cases. Test case generation is still required an automation so that the overall efforts can be reduced. The authors adopted various models to generate test cases. Considering UML-based approaches, Activities, Sequence, and State diagrams are the most used methods for test generation. Other than the UML diagrams, different approaches have been considered. The techniques for test case generation that have been surveyed in this article are represented in the Figure 3.

The functionalities of the system have been focused in Black box and Gray Box approaches. Black box approaches consider external specification to check the functionalities, while the Gray box approach targets both external behavior and the code structure of the system for generation of test cases.

Kim et al. tried to focus on Extended Finite State machine for the test cases generation, and later in his work, he tried an activity diagram and worked on IOAD model. Some authors introduced the techniques that build the intermediate model by which changes can be incorporated such as in Slicing-based algorithms, which showed to be an efficient method to figure out faults and impact of changes.

The techniques that are based on UML-based or specification-based approaches are more efficient as compare to code-based approach (Fahad 2008). Specification-based approaches are much efficient to detect faults in implementation phases. An approach focused on artifacts of analysis stages, and hence the faults detection is easier in early phase. The fault detection by Code-based approaches may not be at early stage and hence will increase the cost of the testing. A model-based approach tends to focus on covering the given criteria and find outs the maximum faults. Prototype tool such as ModelJ Unit and BZ tool has been used for the validation of these techniques.

Slicing based approaches focused on cluster testing where interactions between objects were tested. The algorithm "a novel test" has been designed for generation of the test cases. It is further implemented by the tool for testing the program. The message dependency graph was statically made. Some approaches are based on architectural Slicing depends on Architectural Description Language. This type of approaches requires a tool to support the Software Architecture Specification Modifier so that any update in the components can be evaluated at the design level. The algorithm has $O(n^2)$ space complexity, where n represents the number of components that show more efficient algorithms compared to other slicing-based approaches. Tables 4 and 5 represent the comparison of the various approaches in terms of the techniques, advantages and limitations during the scope of this study. Table 4 represents the comparison of various techniques used in model-based approaches, while Table 5 represents various other approaches that have been considered in the study.

Prakash et al. (2016) in their approach compared the various tools for code-based approaches. The code-based techniques have been classified in Random-based, Symbolic execution-based and Search-based techniques. They have compared 15 tools based on Java and C++. All the tools that have been analyzed were for desktop applications.

We have seen lot of approaches proposed by the authors for test case generation. Various prototype tools, such as BZ, Model Junit, and AGEDIS, have been used and proposed to generate the test cases. As far as our study is concerned, the tools are not as capable of generating fully automized Regression Test Cases, as per the given scenarios and the requirements of the client. The detailed comparison of the tools in terms of its features and limitations or shortcomings has been tabulated in Table 6.

In a broader way, we have classified various approaches in Model-Based, Specification-Based, Slicing-Based approaches and Black box and Gray box approaches. The comparison of these techniques on different parameters has been shown in Table 7.

Recently, Agile software development has changed the way the software is being developed. The continuous development and deployment is a new trend of software development. The valuable features of the system are delivered and based on the data provided by the customers; the functionality of the build is modified (Dingsoyr 2016). The requirements are continuously changing during the course of the software development; therefore, there is a need of an automated tool that can generate story documents directly from the customer requirements. It should also produce test cases for software applications based on the requirements received from the customers and update it as per the modifications in real time. This will reduce the cost, effort and time of software testing.

5 REGRESSION TEST CASE GENERATION FOR WEB APPLICATION

A lot of approaches have been used for test case generation. Very few papers have been found that are related to the regression test case generation of web application. As per the above survey, an automated tool should be available that can generate Regression test cases for software applications and web applications based on requirements and story document. A special consideration has to be given to test web applications. The security and compatibility of browsers have to be considered for its efficiency (Zarrad 2015).

The study on Web tools for automation in testing has been studied by Sharma et al. (2014). They have reviewed several web automation-testing tools understanding the automation testing and their use for testing. Various factors have to be considered for choosing the best one for automation such as integration of various components, its cost and performance.

Table 4. Techniques, Advantages and Limitations of Model Based Approach for Test Case Generation

Approach/Techniques	Reference	Advantages	Disadvantages/Limitations
Design descriptions of software component interactions using Collaboration Diagrams.	Abdurazik (2000)	Focus on Design aspects instead of specifications and code and cover both static and dynamic testing	Lack of empirical evaluation and the tool support.
Architecture using UML and compiled in Intermediate format	Cavarra (2002)	Tool has the capability of combining all the test directives i.e. test constraints, coverage criteria & test purposes.	Lack of case study evidence to support the tool
Interactive software development process model based on use case models.	Riebish (2003)	Reduction in testing efforts and support requirement engineering. Test cases are developed independent of source code.	Difficult to find out usage information and need hierarchical model for efficient test case generation.
Data-oriented boundary coverage criteria.	Kosmatov (2004)	Technique can be used for existence data set or generating test data from formal models.	Study was based on a small example. Need more evaluation for the support of tool.
Message sequence path coverage criteria using sequence diagram	Abdurazik (2004)	Technique was used based on predicate coverage and state charts. State charts are better to find out defects at unit level as compare to sequence diagram,	Generalize the results, more test sets and applications are required. Concurrency is needed in the execution to automate the testing process.
Input/output Labelled Transition States using UML state charts.	Gnesi (2004)	Algorithm proposed for implementing the technique is non-deterministic and hence has a flexibility to produce outputs of different behavior.	The proper strategies for the test selection have to be required for implementing the algorithm.
Coherent approach using sequence and state diagrams	Bertolino et al. (2004)	Technique requires low effort and provide more accuracy in producing test cases.	Require case study to implement and fault detection to evaluate the effectiveness of the proposed approach.
Component behavior modelling and test case generation using global behavioral model.	Hartmann et al. (2004)	Increased the effectiveness by using the efficient coverage criteria.	Further improvement in the tool provided and empirical study is required.
Label transition system using UML sequence diagram	Cartaxo (2007)	Functionalities has been fully covered and tested under the coverage criteria.	The quality of a model used has to be considered for the effectiveness of the approach, since the faults in UML sequence diagrams may affect the test cases.
Input Output Explicit Activity diagram	Kim (2007)	The model used was capable of generating various test scenarios along with the test cases.	Need an automated tool and the approach that can cover maximum coverage criteria in a single framework and automate test case generation.
UML object diagrams using genetic algorithm	Prasanna et al. (2009)	Faults can be find out at the unit level and integration level both	Limited to only object diagrams and hence the effectiveness has to be calculated with other diagrams.
UML activity diagram considering various coverage criteria	Kundu et al. (2009)	Capable of detecting location of faults in the implementation phase.	Have a limitation to only considers activity diagram for one use case at a time.
Use case dependency analysis and test requirements	Mohapatra et al. (2010)	Technique can be best used with system and integration testing	A semi-automated tool has been proposed but requires a tool capable of taking UML diagrams in any format.

(Continued)

Table 4. Continued

Approach/Techniques	Reference	Advantages	Disadvantages/Limitations
Model based GUI testing approach based on annotated use cases.	Navarro et al. (2011)	The process generates the test cases for every possible paths. The approach does not rely on complete model	Implementation of the proposed work is required and hence a tool that support it.
UML models using class diagrams, use case diagrams and sequence diagrams	Sawant (2011)	Technique proposed considers UML models and considers UML diagrams by which test cases can be generated using any application.	Technique does not support any implementation and need empirical analysis for its effectiveness.
Based on activity dependence graph and an intermediate table using activity diagram	Boghdady et al. (2011)	Can be used in integration, regression and system testing. For validation, a cyclomatic complexity has been adopted.	There should be the tool support for a given technique and fault detection should be analyzed.
State activity coverage based on state and activity models.	Swaina et al. (2012)	It can detect the faults present in integrated components as compare to other approaches of same type.	The technique given considers the test data selection manually by the testers and hence needed an automation.
Using Extended finite state machine and UNL design artifacts	Swain et al. (2012)	The technique covered maximum coverage criteria for test case generation by only keeping the necessary test cases.	Optimization of test cases using the proposed technique is required and hence more UML diagrams can be added.
Association-end-Multiplicity criteria using design level class diagram	Wang (2012)	Able to provide most of the coverage criteria and cover all message paths.	Technique is implemented using an assumption that the presented model is correct.
Model driven and UML activity diagram	Thanki et al. (2014)	Technique was proposed to consider the duplicate test cases that can make more effort in test case generation.	Time complexity is dependent on the number of test cases generated and hence may require more time to execute.
Based on Meta model using test models and structural models	Saifan (2016)	Test model and structural model are dependent on each other and hence dynamically modify the content whenever required.	Lack of experimental evaluation to validate the approach in finding the defects.
UML with finite state machine	Patil (2017)	Used in functional test case generation and cover both paths and conditions by using both UML and FSM.	Need an approach that can find our defects in less time and effort. A proper automated tool support and case study-based evaluation is required.
Hybrid bee colony algorithm using combinational UML diagrams.	Sahoo (2017)	A new method of finding out test paths using honey bee algorithm and combinational system graph.	Test cases optimization may not be benefitted for the purpose of testing the components.

An object-oriented web-based model has been given by Kung (2000), where the focus was on three aspects, “the object aspect, the behavior aspects, and the structure aspects.” An object relation diagram is used to represent the relationship between different web application entities for the object aspects. For behavioral aspects, a page navigation diagram has been used to represent the flow of pages for web application. In structure aspects, the authors used two types of diagram to represent control and data flow: Block Branch Diagram and Function Cluster Diagram. A prototype tool is still required to validate the Web-based model.

An approach that only considers specifications for testing has been proposed by Lucca et al. (2006), where they had considered four functional aspects: testing scopes, test methods, testing

Table 5. Techniques, Advantages and Limitations of Various other Approaches for Test Case Generation

Approach/Techniques	Reference	Advantages	Disadvantages/Limitations
Specification based			
State Specification based technique using UML state charts	Offut et al. (1999)	Technique covered full predicate test, transition pair tests, transition coverage and the sequence coverage.	Limited to SCR specifications having one mode class that create the process repetitions.
Control flow and data flow technique using UML state diagrams	Kim et al. (1999)	By using the concurrent and hierarchical model, reduction in duplicate test cases has been achieved.	Technique can be used only for unit testing and needed an automated tool to support the class testing.
Requirement based technique based on customer behavior and activity diagram	Chen (2002)	The cost has been evaluated on different parameters considering both customer and vendor.	Implemented on small application. Require more empirical studies to proof the effectiveness.
Integrated classification tree methodology	Alhroob et al. (2010)	Technique reduces the duplicate test case generation resulting less effort and cost.	Provide less information about the test cases when software specifications are considered.
Actvity Diagram and use case specification	Sanz (2012)	Technique consider an cost effective approach designed for the system specifications.	Lack of empirical evaluation and need to remove duplicate test cases.
Functional Based			
A Gray box method using UML activity diagrams	Linzhang (2004)	It can describe the external system behavior as well as the code structure to generate the test cases.	Study evidence is required to show the effectiveness of the technique.
Steiner tree-based test case generation using UML diagram.	Sapna (2015)	The method is benefitted as the changed nodes can be added at the terminal end.	Tested for smaller functionality and hence needed to test on bigger application.
Slicing based			
Dynamic slicing	Korel (1988)	Technique is not dependent on CFG instead it works implemented on the program path.	Limited to arrays and data structures.
Conditioned program slicing	Canfora et al. (1998)	It works as an interface between static and dynamic approaches and hence has a capability of producing effective tests cases.	Require an empirical evaluation and a tool that support and evaluate the technique.
Dynamic dependence graph	Agrawal (1990)	It produces a node for every statement occurred previously including already executed statements.	Addition of new nodes in the graph independent of dynamic slice and hence increases the effort required.
System dependence graph	Horwitz et al. (1990)	Produces forward slices for interprocedural program	Cost of using the technique is more and depend on the size of the slice used.
Component Slicing using Architecture component dependence graph	Lalchandani (2008)	Take less response time to execute slicing commands therefore produces effective regression test cases.	Need to have evaluation for dynamic deletion and addition of test cases in design level.
Dynamic slicing based on dependence graph using UML sequence diagram	Samuel et al. (2008)	Require less number of test cases for slice test coverage.	Dependent on dependence graph and hence the effectiveness of graph is must.

(Continued)

Table 5. Continued

Approach/Techniques	Reference	Advantages	Disadvantages/Limitations
Some Others approaches			
Tabu search algorithm using UML	Shanthi et al. (2012)	Capable of finding faults in loops along with their location in implementation.	Require tool support and more case study evaluation for validation.
Based on metamorphic relations and fuzzy logic	Bandaru et al. (2016)	Covers the oracle problem by covering all the abstract states.	Need to implement with a tool so that proper fault detection can be achieved,
Based on Business process execution language	Li (2016)	The technique can find out the affected data flow, which may be due to process, binding, and interface change.	Test case generation still has to be done, which is limited to single version in the given approach.

Table 6. Some Tools Proposed by the Literature

References	Tool Name	Features	Limitations/shortcomings
Offut et al. (1999)	UMLTest tool	Integrated with rational rose and generate tests from specifications.	The tool accepts the input as a variable only in Boolean.
Cavarra (2002)	AGEDIS tool	Can combine all the test directives, i.e., test constraints, coverage criteria and test purposes	Lack of generating fully automated test cases
Kosmatov (2004)	BZ testing tool	Test case generation using UML specifications as per the given criteria.	Study was based on a small example. Need more evaluation for the support of tool.
Linzhang (2004)	UMLTGF tool	Tool is capable of analyzing the UML specifications.	Semi-automated tool that needs UML specifications
Riebish (2003)	XML based tool	Capable of transforming usage and system behavior in usage model.	Difficult to find out usage information using the tool
Swaina et al. (2012)	ModelJUnit	Produces UML design artifacts.	The tool given considers the test data selection manually by the testers and hence needed an automation.
Mohapatra et al. (2010)	ComTest	Generate test cases as per the UML diagram taken in the required format	Semi-automated tool
Sawant (2011)	ATCUM tool	Can generate the XML file from the UML diagrams	Semi-automated tool and does not provide regression test case generation.
Lalchandani (2008)	SRTWA prototype tool	For a given criteria, it computes a slice for a given architecture.	A tool only handles only the subset of wright Architectural description languages.
Thanki et al. (2014)	Test optimal tool	Test case analytic can be done	Can only minimize the test cases but require producing test cases.

Table 7. Comparison of Different Approaches for Test Case Generation

Comparison of Different Approaches							
Approaches	**References**	**Parameters**					
		UML Notations	Risk Based	Fault Detection	Cost Analysis	Tool Support	Case Study Evidence
Model-Based	Abdurazik (2000)	COD	No	No	No	No	Yes
	Cavarra (2002)	CD, OD, SD	No	No	No	Yes	No
	Riebish (2003)	SD	No	No	No	Yes	Yes
	Kosmatov (2004)	OCL	No	No	Yes	Yes	Yes
	Abdurazik (2004)	SCD, SD	No	Yes	No	Yes	Yes
	Gnesi (2004)	SCD	No	No	No	No	No
	Bertolino et al. (2004)	SCD, SD	No	No	No	No	No
	Hartmann et al. (2004)	SCD, SD	No	No	No	Yes	No
	Cartaxo (2007)	SD	No	No	No	No	Yes
	Kim (2007)	AD	No	No	No	No	Yes
	Prasanna et al. (2009)	OD	No	Yes	No	No	Yes
	Swaina et al. (2012)	SCD, AD	No	No	No	Yes	Yes
	Swain et al. (2012)	SCD	No	No	No	Yes	Yes
	Mohapatra et al. (2010)	SD	No	No	No	Yes	Yes
	Kundu et al. (2009)	AD	No	No	No	No	Yes
	Navarro et al. (2011)	UCD	No	No	Yes	No	Yes
	Sawant (2011)	UCD, CD, SD	No	No	No	Yes	Yes
	Boghdady et al. (2011)	AD	No	No	Yes	No	Yes
	Wang (2012)	ID	No	No	No	No	Yes
	Thanki et al. (2014)	SCD, AD	No	No	No	Yes	No
	Saifan (2016)	Test model	No	Yes	No	No	No
	Patil (2017)	AD	No	Yes	No	No	Yes
	Sahoo (2017)	SCD, SD	No	Yes	No	No	Yes
Specification -Based	Offut et al. (1999)	SCD	No	Yes	No	Yes	Yes
	Kim et al. (1999)	SCD	No	No	No	No	No
	Chen (2002)	AD	Yes	Yes	No	Yes	Yes
	Alhroob et al. (2010)	CD, OCL	No	No	No	No	Yes
	Sanj (2012)	AD	No	Yes	No	No	Yes
Functional-Based	Linzhang (2004)	AD	No	No	No	Yes	No
	Sapna (2015)	AD	No	No	No	No	Yes
Slicing techniques	Lalchandani (2008)	ADL	No	Yes	Yes	Yes	Yes
	Samuel et al. (2008)	SD	No	No	No	Yes	Yes
	Korel (1988)	-	No	No	No	No	Yes
	Agrawal (1990)	-	No	No	No	No	No
	Horwitz et al. (1990)	-	No	Yes	No	No	Yes
	Canfora et al. (1998)	-	No	No	Yes	No	Yes
Others	Shanthi et al. (2012)	AD	No	No	No	No	Yes
	Bandaru et al. (2016)	-	No	No	No	No	No
	Li (2016)	-	No	Yes	No	Yes	No

Notations used in comparison: COD: Collaboration Diagram, CD: Class Diagram, OD: Object Diagram, SD: Sequence Diagram, OCL: Object Constraint Language, SCD: State Chart Diagram, AD: Activity Diagram, UCD: Use Case Diagram, ID: Interaction Diagram, ADL: Architectural Description Language.

tools, and test strategies. In most of the web testing, only non-functional testing is performed, but Giuseppe tried to focus on functional testing of the web application.

A model-based technique has been proposed by Suhag et al. (2014), where test cases have been generated with the use of web diagrams and sequence diagrams. The technique focuses on both functional requirements and dynamic behavior. The technique is restricted to the single-page test case generation of an application.

Elbaum et al. (2005) proposed a user session-based technique where test cases have been generated on a user request. The test data is provided by the users, which provide an additional increase to the test data. Experiments require more validations to determine the actual execution time and cost-effectiveness when a larger number of sessions is provided and under different types of loads.

An approach based on web browser interaction is proposed by Zhu (2009). They have given an approach that focuses on action done by the browser. The approach generates the test cases through the navigation tree targeting browser history and their interfaces. The focus of study was on web browser interaction, like on back or forward buttons. The approach requires an experiment to show the effectiveness and an appropriate tool to generate the test cases.

An approach proposed by Zhongsheng et al. (2010) considered a series of user sessions, its prioritization, and genetic algorithm to optimize the same. They proposed a URL trace-based reduction algorithm where all the requested sequences of the URLs are covered. The approach is useful when dealing with high numbers of existing users. The test approach only considers the test cases as per its test coverage ratio. Other factors for the evaluation cost and time of running the test cases are also required to be considered.

A web Services-based approach has been proposed by Tarhini (2006). The approach identifies the errors related to the modification. The approach considers every sequence of test case that belongs to the various behaviors found in modified program. The task Precedence Graph shows the functional behavior of web application. The approach has been considered for connection with new web service, addition or deletion of any operation and modification of specification. Technique has not been implemented to web application so that its efficiency could be proven.

An approach that considered a session data repair has been proposed by Alshahwan (2008). The approach is different from others, which focused on session repair. It constructs the new session by repairing the old session, which has been obsolete due to changes done in it. Session repair approach is useful as compared to other techniques, as changes in web applications are very frequent and it detects faults that are not found in white box approaches.

Chen (2010) tried to find out the best test cases based on their weights for prioritization by only identifying the changed affected items. Test cases have been prioritized as per the elements having greater modifications and the highest weight. The technique has used the dependence analysis where the impact analysis has been done to find out the affected portion.

A technique for workflow-based web services is proposed by Mei et al. (2013). The technique focused on the workflow and based on adaptive testing. The preemptive regression testing has been given that considers the dynamic changes and manage sub-sessions to cover modifications in coverage. It executes the modified coverage and re-arranges the priority of new test cases dynamically.

The technique for java web services using the approach of code transformation has been given by Lin et al. (2006). Control-based java graph (JIG) has been formed for old code and the modified one and compared. The differences in the trees will state which edges are critical and need to be tested. The overall purpose was to reduce the test build, which requires re-run. The approach was limited to the web application, which is written in Java and deployed in Axis.

Ruth et al. (2007) has proposed an approach for web service for regression test selection. The technique was based on Control Flow graph (CFG) that is required from every application and services participating instead of taking the source code. Based on the CFG constructed and the critical edges, the test cases have been selected. The approach has been defined only for the static services.

In the work proposed by Kaur et al. (2016), the constraints have been reused for the web testing, which are treated as a priority-based testing. The genetic algorithm has been used to find the sequence of test cases execution. The approach proposed requires a case study for the validation.

5.1 Summary

Effective Regression Test Case generation is an important aspect of any Web application (Mounica 2015). Many authors have proposed models to generate the test cases for web application. An object-oriented web-based model has been used to show the flow of pages for web application and testing the same. Few of the techniques consider only the specification to be used in test case generation. Session-based techniques are also used to test case generation but require the experimental validation to verify the techniques against large sessions. Some techniques are based on web browser interaction, which can generate the test cases depending on the browser history and interfaces.

As there is a lack of Regression test case generation for web application, there is a need of an automated tool that can generate story documents directly from the customer requirements. It should also produce test cases for software applications depending on the requirements received from the customers and update it as per the modifications in real time. This will reduce the cost, effort and time of software testing. An automated tool should be available that can generate Regression test cases for software applications and web applications based on requirements and story document. The techniques proposed in the study were not intended to generate the regression test cases by an automated tool.

6 DISCUSSION

We have seen a lot of approaches proposed for test case generation. We have also surveyed techniques for web application for object-oriented applications and session-based and browser-based interaction, but only a few authors proposed the model and automated techniques for regression test case generation. The web application needs special attention, as almost every application is moving toward the web and requires an effective testing. The web application has completely changed the processing system by reducing the efforts and cost of resources (Monier 2015).

Various approaches have been given for test case generation but very few for regression test cases generation for Web applications. It still requires automation so that the overall efforts can be reduced. The cost of regression testing depends on the characteristics that its test build possesses. Rothermel et al. (2001) showed in their work the importance of granularity for the cost of regression testing. Authors have tried to focus on grouping of test cases in the test build. Elbaum et al. (2003) showed that there is an impact of changes in the modified versions of software and has effect on the cost of regression testing. The study showed that the size of the change, tester's choice between fault detection and cost effectiveness, and the distribution of changes on all the functions may produce the cost-effective test cases.

Automated regression testing is important for hosting and building web application, mobile application, and cloud application. Initially, when an organization starts building their application with a scope defined, but over time, with changing user needs, the scope also keeps changing. So, the complexity of these applications increases, and there is a need of testing the additional functionality added without any effect on existing functionalities. The traditional technique used for these complex applications lack a number of features, such as fixing errors, identifying hypermedia, testing database connections, and so on. Our study on regression testing and the tool for automated regression test case generation may therefore also be considered for various other applications, like cloud and mobile application.

7 CONCLUSION & FUTURE SCOPE

Good quality products and software are important in every aspect to the end users. The customer requires a good product that fulfills all of its requirements and specifications. The industry is benefited if the product implemented does not have any defects and meets the quality standards at the end of product implementation.

If we can automate test case generation with the help of a tool, we can reduce a lot of time, effort, and cost. Studies based on literature review and industry trend suggest that there is still a requirement of an automated tool for test cases generation in regression testing for a web application and specification-based tool. A lot of cost in terms of effort, time, and resources has been invested on regression testing to generate efficient test cases. There should be a proper automated regression testing tool that can generate test cases according to different scenarios for web application. As per the survey conducted in this article, there is a need of an automation tool that can generate automated regression test cases for the web application. The tool should also have the capability to generate test cases directly from the customer requirements and story document. Researchers need to emphasize research on a tool that can generate test cases from the story line or story document, which is still not available for test cases generation. A tool should also produce regression test cases for software applications and web applications based on story document generated in earlier phases. Further, test cases should be prioritized so that the reduction in the overall effort and cost can be achieved.

Test case generation of regression testing for web applications with the help of automation tools will definitely help testers to test the test suites efficiently and in less time. An automation tool that can convert requirements directly into the test cases will help in reducing time of the testers to test the test cases in low cost and resources.

REFERENCES

Aynur Abdurazik and Jeff Outt. 2000. Using UML collaboration diagrams for static checking and test generation. *Proceedings of the 3rd International Conference on the Unified Modelling Language.* 383–395.

Aynur Abdurazik, Jeff Offutt, and Andrea Baldini. 2004. A controlled experimental evaluation of test cases generated from UML diagrams. Technical Report: ISE-TR-04-03, Dept of Info and Software Engineering, George Mason University. DOI: 10.1.1.124.9847

Aynur Abdurazik, Jeff Offutt and Andrea Baldini. 2005. A comparative evaluation of tests generated from different UML diagrams: Diagrams and data. *GMU Technical Report ISE-TR-05-04.*

Hiralal Agrawal and Joseph Robert Horgan. 1990. Dynamic program slicing. In *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI'90).* 246–256.

Aysh Alhroob, Keshav Dehal, and Alamgir Hossain. 2010. Automatic test cases generation from software specifications. *E-info. Softw. Eng. J.* 4, 1, 109–121.

Nadia Alshahwan and Mark Harman. 2008. Automated session data repair for web application regression testing. In *Proceedings of the 1st International Conference on Software Testing, Verification, and Validation.* DOI: 10.1109/ICST.2008.56

Saswat Ananda, Esmund Burkeb, Tsong Y. Chen, John Clark, and Myra B. Cohen. 2013. An orchestrated survey of methodologies for automated software test case generation. *J. Syst. Softw.* 86, 8, 1978–2001. DOI: 10.1016/j.jss.2013.02.061

Ramya Bandaru and J. A. Mayan. 2016. Novel approach for whole test suite generation using metamorphic relations. *Indian J. Sci. Technol.* 9, 10. DOI: 10.17485/ijst/2016/v9i10/88983

Ghinwa Baradhi and Nashat Mansour. 1997. A comparative study of five regression testing algorithms. In *Proceedings of the Australian Software Engineering Conference.* 174. DOI: 10.1109/ASWEC.1997.623769.

Antonia Bertolino, Eda Marchetti, and H. Muccini. 2004. Introducing a reasonably complete and coherent approach for model-based testing. *Electron. Notes Theoret. Comput. Sci.* 116, 85–97. DOI: 10.1016/j.entcs.2004.02.084

David Binkley. 1995. Reducing the cost of regression testing by semantic guided test case selection. In *Proceedings of IEEE International Conference on Software Maintenance.* IEEE Computer Society Washington, DC, 251–260.

Pakinam N. Boghdady, Nagwa L. Badr, Mohamed Hashem, and Mohamed F. Tolba. 2011. A proposed test case generation technique based on activity diagrams. *Int. J. Eng. Technol.* 11, 03.

Gerardo Canfora, Aniello Cimitile, and Andrea De Lucia. 1998. Conditioned program slicing. *Info. Softw. Technol.* 40, 11, 595–607. DOI: 10.1016/S0950-5849(98)00086-X

Emaruela G. Cartaxo, Francisco G. O. Neto, and Patricia D. L. Machado. 2007. Test case generation by means of UML sequence diagrams and labeled transition systems. In *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics.* 1292–1297. DOI: 10.1109/ICSMC.2007.4414060

Alessandra Cavarra, Thierry Jeron, and Alan Hartman. 2002. Using UML for automatic test generation. *Proceedings of the International Symposium on Software Testing and Analysis (ISSTA'02).*

Lin Chen, Ziyuan Wang, Lei Cu, Hongmin Lu, and Baowen Xu. 2010. Test case prioritization for web service regression testing. In *Proceedings of the 5th IEEE International Symposium on Service Oriented System Engineering.* IEEE Computer Society. DOI: 10.1109/SOSE.2010.27

Yanping Chen, Robert L. Probert, and D. P. Sims. 2002. Specification based regression test selection with risk analysis. In *Proceedings of Conference of IBM Center for Advanced Studies.* IBM Press, 1–14.

Torgeir Dingsoyr and Casper Lassenius. 2016. Emerging themes in agile software development: Introduction to the special section on continuous value delivery. *Info. Softw. Technol. ScienceDirect* 77, 55–60. DOI: 10.1016/j.infsof.2016.04.018

Sebastian Elbaum, Gregg Rothermel, Srikanth Karre, and Marc Fisher. 2005. Leveraging user session data to support web application testing. *IEEE Trans. Softw. Eng.* 31, 3, 187–202. DOI: 10.1109/TSE.2005.36

Sebastian Elbaum, Praveen Kallakuri, Alexey G. Malishevsky, Gregg Rothermel, and Satya Kanduri. 2003. Understanding the effects of changes on the cost-effectiveness of regression testing techniques. *J. Softw. Test. Verificat. Reliabil.* 13, 2, 65–83. DOI: 10.1002/stvr.263

W. Eric Wong, J. R. Horgan, Saul London, and Hira Agrawal. 1997. A study of effective regression testing in practice. In *Proceedings of the 8th IEEE International Symposium on Software Reliability Engineering.* 264–274. DOI: 10.1109/ISSRE.1997.630875

Muhammad Fahad and Aamer Nadeem. 2008. A survey of UML based regression testing. In *Proceedings of the International Conference on Intelligent Information Processing.* Springer, Vol. 288. 200–210. DOI: 10.1007/978-0-387-87685-6_25

Stefania Gnesi, Diego Latella, and Mieke Massink. 2004. Formal test-case generation for UML statecharts. In *Proceedings of the 9th IEEE International Conference on Engineering Complex Computer Systems Navigating Complexity in the e-Engineering.* Vol. 42, No. 7, 75–84.

Todd L. Graves, Mary J. Harrold, Jung M. Kim, Adam Porte, and Gregg Rothermel. 2001. An empirical study of regression test selection techniques. *ACM Trans. Softw. Eng. Methodol.* 10, 2, 184–208. DOI: 10.1145/367008.367020

Sumit Gupta, Sunil Indori, and Atishey Bansal. 2015. Automation beyond application test scripting. *Proceedings of the 15th Annual International Software Testing Conference.*

J. Hartmann, M. Vieira, H. Foster, and Axel Ruder. 2004. UML based Test generation and execution. *Technical report, Siemens Corporate Research, Inc.* DOI: 10.1.1.84.3327

Douglas Hoffman. 1999. Cost benefits analysis of test automation. In *Proceedings of the Software Testing Conference on Software Quality Methods (STARWEST'99).*

Susan Horwitz, Thomas W. Reps, and David Binkley. 1990. Interprocedural slicing using dependence graphs. *ACM Trans. Program. Lang. Syst.* 12, 1, 26–60. DOI: 10.1145/77606.77608

Seifedine Kadry. 2011. A new proposed technique to improve software regression testing cost. *Int. J. Secur. Appl.* 5, 3, 45–48.

P. Kandil, S. Moussa, and N. Badr. 2017. Cluster based test cases prioritization and selection technique for agile regression testing. *J. Softw.: Evol. Process.* 29, 6.

Sarbjot Kaur and Makul Mahajan. 2016. Re-usability of constraints for test case generation in different applications using genetic algorithm. *Int. J. Comput. Sci. Netw. Secur.* 16, 5, 113–117.

Monalisha Khandai, Arup A. Acharya, and Durga P. Mohapatra. 2011. A survey on test case generation from UML model. *Int. J. Comput. Sci. Info. Technol.* 2, 3, 1164–1171.

M. Khatibsyarbini, M. A. Isa, D. N. Jawawi, and R. Tumeng. 2017. Test case prioritization approaches in regression testing: A systematic literature review. *Info. Softw. Technol.* 93, 1, 74–93. DOI: 10.1016/j.infsof.2017.08.014

Hyungchoul Kim, Sungwon Kang, Jongmoon Baik, and Inyoung Ko. 2007. Test cases generation from UML activity diagrams. In *Proceeding of 8th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing.* 556–561. DOI: http://doi.ieeecomputersociety.org/10.1109/SNPD.2007.189

Y. G. Kim, H. S. Hong, S. M. Cho, D. H. Bae, and S. D. Cha. 1999. Test case generation from UML state diagrams. *IEE Proc. Softw.* 146, 4, 187–192.

Bogdan Korel and Janusz W. Laski. 1988. Dynamic program slicing. *Info. Process. Lett.* 29, 3, 155–163. DOI: 10.1016/0020-0190(88)90054-3

Nikolai Kosmatov, Bruno Legeard, Fabien Peureux, and Mark Utting. 2004. Boundary coverage criteria for test generation from formal models. *Proceedings of the 15th International Symposium on Software Reliability Engineering.*

Debasish Kundu and D. Samanta. 2009. A novel approach to generate test cases from UML activity diagrams. *J. Object Technol.* 8, 3, 65–83. DOI: 10.5381/jot.2009.8.3.a1

David C. Kung, Chien H. Liu, and Pei Hsia. 2000. An object-oriented web test model for testing web applications. In *Proceedings of the 1st Asia-Pacific Conference on Quality Software.* IEEE Computer Society, 111–120. DOI: 10.1109/APAQ.2000.883784

Adriaan Labuschagn, Laural Inozemtseva, and Reid Holmes. 2017. Measuring the cost of regression testing in practice. In *Proceedings of the 11th Joint meeting of the European Software Engineering Conference and the ACM SIGSOFT symposium on the foundations of Software Engineering.* Retrieved from https://doi.org/10.1145/3106237.3106288

Jaiprakash T. Lalchandani and Rajib Mall. 2008. Regression testing based-on slicing of component-based software architectures. *Proceedings of the 1st India Software Engineering Conference.* ACM New York, NY, 67–76. DOI: 10.1145/1342211.1342227

Bikin Li and Pengcheng Zhang. 2016. Test case selection for data flow based regression testing of BPEL composite services. *Proceedings of the 2016 IEEE International Conference on Services Computing.* DOI: 10.1109/SCC.2016.77

Feng Lin, Michael Ruth, and Shengru Tu. 2006. Applying safe regression test selection techniques to java web services. In *Proceedings of the International Conference on Next Generation Web Services Practices.* IEEE Computer Society Washington, DC, 133–142. DOI: 10.1109/NWESP.2006.8

Wang Linzhang, Yuan Jiesong, Yu Xiaofeng, Li Xuandong, and Zheng Guoliang. 2004. Generating test cases from UML activity diagram based on gray-box method. Technical Report No. NJU-SEG-2004-IC-002. In *Proceedings of the 11th Asia-Pacific Software Engineering Conference.* IEEE Computer Society Washington, DC, 284–291. DOI: 10.1109/APSEC.2004.55

Wang Linzhang, Yuan Jiesong, Yu Xiaofeng, Li Xuandong, and Zheng Guoliang. 2004. Generating test cases from UML activity diagram based on gray-box method. In *Proceedings of the 11th Asia Pacific Software Engineering Conference.* IEEE Computer Society, 284–291. DOI: 10.1109/APSEC.2004.55

Giuseppe A. Di Lucca, and Anna R. Fasolino. 2006. Testing web-based applications: The state of the art and future trends. In *Proc. Info. Softw. Technol.* 48, 12, 1172–1186. http://dx.doi.org/10.1016/j.infsof.2006.06.006

Pranali Prakash Mahadik and D. M. Thakore. 2016. Survey on automatic test data generation tools and techniques for object-oriented code. *Int. J. Innovat. Res. Comput. Commun. Eng.* 4, 357–364. DOI: 10.15680/IJIRCCE.2016.0401082.

Lijun Mei, W. K. Chan, T. H. Tse, Bo Jiang, and Ke Zhai. 2013. Preemptive regression testing of workflow-based web services. *IEEE Trans. Serv. Comput.* 8, 5, 740–754. DOI: 10.1109/TSC.2014.2322621

Durga P. Mohapatrab and Rajib Mall. 2010. Test case generation based on use case and sequence diagram. *Int. J. Softw. Eng.* 3, 2, 21–52.

Mohamed Monier and Mahmoud Mohamed El-mahdy. 2015. Evaluation of automated web testing tools. *Int. J. Comput. Appl. Technol. Res.* 4, 5, 405–408.

Agasarpa Mounica and Lokanadham Naidu Vadlamudi. 2016. Automated model-based testing for an web applications. *Int. J. Sci. Eng. Technol. Res.* 5, 5, 1551–1555.

Pedro L. M. Navarro, Diego S. Ruiz, and Gregorio M. Perez1. 2010. A proposal for automatic testing of GUIs based on annotated use cases. *Adv. Softw. Eng.—Special Issue on Software Test Automation.* 2010, 5. DOI: 10.1155/2010/671284

Jeff Offutt and Aynur Abdurazik. 1999. Generating test cases from UML specifications. Technical Report, ISE-TR-99-09, *Information and Software Engineering, George Mason University.*

Supriya S. Patil and Prof. Pramod A. Jadhav. 2017. Functional test case generation based on model driven testing using FSM and UML activity diagram. *Int. J. Adv. Res. Comput. Sci.* 8, 5, 1527–1530.

M. Prasanna and K. R. Chandran. 2009. Automatic test case generation for UML object diagrams using genetic algorithm. *IJA Soft Comput. Appl.* 1, 1.

D. Qiu, B. Li, S. Ji, and H. Leung. 2015. Regression testing of web service: A systematic mapping study. *ACM Comput. Surveys* 47, 2, 21.

Matthias Riebisch, Ilka Philippow, and Marlo Gotze. 2003. UML-based statistical test case generation. *Obj. Comp. Arch. Serv. Appl. Netw. World* 2591, 394–411.

Gregg Rothermel and Mary J. Harrold. 1996. Analyzing regression test selection techniques. *IEEE Trans. Softw. Eng.* 22, 8, 529–551.

Gregg Rothermel, Sebastian Elbaum, Alexey Malishevsky, Praveen Kallakuri, and Brian Davia. 2001. The impact of test suite granularity on the cost-effectiveness of regression testing. In *Proceedings of the 24th International Conference on Software Engineering.* ACM New York, NY, 130–140. DOI: 10.1145/581339.581358

Michael Ruth and Shengru Tu. 2007. Towards automating regression test selection for web services. In *Proceedings of the 16th International Conference on World Wide Web (WWW'07).* 729–736. DOI: 10.1145/1242572.1242799

Rajesh Ku. Sahoo, Santosh Kumar Nanda, Durga Prasad Mohapatra, and Manas Ranjan Patra. 2017. Model driven test case optimization of UML combinational diagrams using hybrid bee colony algorithm. *Int. J. Intell. Syst. Appl.* 9, 6, 43–54. DOI: 10.5815/ijisa.2017.06.05

Ahmad A. Saifan, Iyad Alazzam, Mohammed Akour, and Feras Hanandeh. 2016. Regression test-selection technique using component model-based modification: Code to test traceability. *(IJACSA) Int. J. Adv. Comput. Sci. Appl.* 7, 4, 88–92.

Philip Samuel and Rajib Mall. 2008. A novel test case design technique using dynamic slicing of UML sequence diagrams. *E-info. Softw. Eng. J.* 2, 1, 71–92.

Leus F. Sanz and Sanjay Misra. 2012. Practical application of UML activity diagrams for the generation of test cases. In *Proc. Roman. Acad. Ser. A: Math. Phys. Techn. Sci.* 13, 3, 251–260.

P. G. Sapna and Arun K. Balakrishnan. 2015. An approach for generating minimal test cases for regression testing. *Proced. Comput. Sci.* 47, 188–196. DOI: 10.1016/j.procs.2015.03.197

Vinaya Sawant and Ketan Shah. 2011. Automatic generation of test cases from UML models. In *Proceedings on International Conference on Technology Systems and Management.* Vol. 2, No. 7, 7–10.

A. Schwartz and H. Do. 2016. Cost-effective regression testing through adaptive test prioritization strategies. *J. Syst. Softw.* 115, 61–81. DOI: 10.1016/j.jss.2016.01.018

Syed Asad Ali Shah, Raja Khaim Shahzad, Syed Shafique Ali Bukhari, Nasir Mehmood Minhas, and Mamoona Humayun. 2015. A review of class-based test case generation techniques. *J. Softw.* 11, 5, 464–480. DOI: 10.17706/jsw.11.5.464-480.

A. V. K. Shanthi and G. Mohankumar. 2012. A novel approach for automated test path generation using tabu search algorithm. *Int. J. Comput. Appl.* 48, 13, 28–34. DOI: 10.5120/7410-0449

Monika Sharma and Rigzin Angmo. 2014. Web based automation testing and tools. *Int. J. Comput. Sci. Info. Technol.* 5, 1, 908–912.

Vikas Suhag and Rajesh Bhatia. 2014. Model based test cases generation for web applications. *Int. J. Comput. Appl.* 92, 3. DOI: 10.5120/15991-4948.

Ranjita Swain, Vikas Panti, Prafulla K. Behra, and Durga P. Mohpotra. 2012. Automatic test case generation from UML state chart diagram. *Int. J. Comput. Appl.* 42, 7. DOI: 10.1.1.259.1439

Santosh K. Swaina, Durga P. Mohapatrab, and R. Mall. 2010. Test case generation based on state and activity models. *J. Object Technol.* 9, 5, 1–27.

Abbas Tarhini, Hacene Fouchal, and Nashat Mansour. 2006. Regression testing web services-based applications. In *Proceedings of the IEEE International Conference on Computer Systems and Applications.* IEEE Computer Society Washington, DC, 163–170. DOI: 10.1109/AICCSA.2006.205085

Hetal J. Thanki and S. M. Shinde. 2014. Test case generation and minimization using UML activity diagram in model driven environment. *Int. J. Comput. Org. Trends* 9, 1, 41–44. DOI: 10.14445/22492593/IJCOT-V9P309

Filip I. Vokolos and Phyllis. G. Frankl. 1997. Pythia: A regression test selection tool based on textual differencing. In *Proceedings of the 3rd International Conference on Reliability, Quality and Safety of Software-Intensive Systems.* Chapman & Hall, Ltd. London, UK, 3–21.

Yiwen Wang and Mao Zheng. 2012. Test case generation from UML models. In *Proceedings of the 45th Annual Midwest Instructions and Computing Symposium.*

Shin Yoo and Mark Harman. 2012. Regression testing minimisation, selection and prioritisation: A survey. *Softw. Test. Verificat. Reliabil.* 22, 2, 67–120. DOI: 10.1002/stvr

Anis Zarrad. 2015. A systematic review on regression testing for web-based applications. *J. Softw.* 10, 8. DOI: 10.17706//jsw.10.8.971-990.

Qian Zhongsheng. 2010. Test case generation and optimization for user session-based web application testing. *J. Comput.* 5, 11.

Bin Zhu, Huaikou Miao, and Lizhi Cai. 2009. Testing a web application involving web browser interaction. In *Proceedings of the 10th ACIS International Conference on Software Engineering, Artificial Intelligences Networking and Parallel/Distributed Computing.* DOI: 10.1109/SNPD.2009.59

Received June 2017; revised January 2018; accepted June 2018