A Survey on Test Case Generation using UML Diagrams and Feasibility Study to Generate Combinatorial Logic Oriented Test Cases

Subhash B. Tatale
and
Dr. V. Chandra Prakash
Koneru Lakshmaiah Education Foundation, Vaddeswaram, India

Generating test cases automatically from the design specification of a system is the most challenging phase in Software Development Life Cycle. UML diagrams are the industrial standard design modelling artifacts and the same can also be used for automatic generation of test cases which can be subsequently used by the testers to verify the functionality of the System under Test. In the present survey, authors have focused on automatic generation of test cases using UML Sequence and Activity diagrams. Also, the authors have conducted a feasibility study to know whether these diagrams can be made use of to generate combinatorial logic oriented test cases.

Keywords: Software Testing, Combinatorial Test Case Generation,UML Diagrams, Survey, Feasibility Study, Activity Diagram, Sequence Diagram.

1. INTRODUCTION

Software testing is an important phase of Software Development Life Cycle (SDLC) which is necessary to produce an effective, efficient and reliable system. For a complex system, a large number of test cases are to be generated for effective and efficient testing. Generating and validating a large number of test cases is a very laborious, time-consuming and costly task. During manual test case generation, there is a scope to get erroneous or example, redundant test cases can be generated; some requirements might get missed out; contradictory test cases may be generated, etc. To avoid this error condition, an automatic test case generation is extremely essential wherever possible. Moreover, automatic generation of test cases can reduce testing cost by eliminating costly manual test case generation.

1.1 Automatic test case generation using UML Sequence and Activity diagrams

Unified Modeling Language (UML) has now become the de facto standard Hartmann et al. [2005] for object-oriented modelling and design. UML diagrams are an essential source of information for generating test cases. They can be used as an input to generate test cases, but, of course, the generation of test cases from UML diagrams is one of the most challenging tasks. The process of test case generation from UML diagrams will help a lot to identify problems early in the software development process which significantly reduces the time and cost of testing. UML-based automatic test case generation is a pragmatic approach to generate test cases Briand and Labiche [2002] and it is receiving increasing attention from researchers.
A Sequence diagram captures the exchange of messages between objects during execution of a use case Nebut et al. [2006]. It focuses on the order in which the messages are sent. It also represents the flow of control among objects during interaction between objects. It also represents the discrete behaviour of an object through sequence graph. Test cases generated from Sequence diagram satisfy coverage criteria like message, message-sequence, path, etc.
An activity diagram is used to represent the workflow of activities and actions, in step by step manner, which takes place in a system. It can represent both the sequential and concurrent control

flow of activities in the system. A test case model can be derived from an Activity diagram and the test cases can be generated automatically from test case model. Generated test cases are useful to test all possible flows of execution of an Activity diagram. Test cases generated using an Activity diagram satisfies the coverage criteria like activity, transition, simple-path, concurrent-path, etc.

1.2 Combinatorial Testing

There are many systems viz. Concession management subsystem of Railway reservation system, College admission system, Tuition fee concession subsystem, etc. in which combinatorial logic is extensively used. Combinatorial Testing (CT) is gaining high importance to test such type of systems. Nowadays, Combinatorial Test Design Model (CTDM) is popularly used to generate combinatorial logic oriented test cases in an automatic way. There is a great need to put more focus on how to generate combinatorial logic oriented test cases while optimizing the size of the test suites.

Chandraprakash and Kadiyala [2006], Kondhalkar and ChandraPrakash [2018], Prakash et al. [2018], Bewoor et al. [2019], LakshmPrasad et al. [2019] used the different Heuristic algorithms viz. Genetic Algorithm, Particle Swarm Optimization, etc. to generate optimized test cases through combinatorial testing techniques. Ramgouda and Chandraprakash [2018] used a Neural Network based approach to improve combinatorial coverage in the combinatorial testing approach. Gouda and Chandraprakash [2019], Ramgouda and Chandraprakash [2019] used a multi-objective crow search and fruit-fly optimization techniques to optimize combinatorial test cases in constraints handling environment. Mudarakola et al. [2018], Prasad and J.K.R.Sastry [2018b], Prasad and J.K.R.Sastry [2018a], SasiBhanu et al. [2018], SasiBhanu et al. [2019] published research articles for testing embedded and distributed embedded systems using combinatorial methods.

1.3 Need for generating combinatorial logic oriented test cases using UML Sequence and Activity diagrams

The So far, different methods and techniques are used to generate test cases automatically using UML Sequence and Activity diagrams by many researchers. There is a need to generate combinatorial logic oriented test cases for those systems where combinatorial logic is essential. These systems are modeled by using different UML diagrams. The same UML diagrams can be used to generate combinatorial logic oriented test cases. The authors of this paper conducted a feasibility study to know whether combinatorial logic oriented test cases can be generated using UML Sequence and Activity diagrams.

The remaining paper is organized as follows. A survey on generation of test cases using Sequence and Activity diagrams is discussed in section 2. Section 3gives feasibility study of combinatorial logic oriented test cases from Sequence and Activity diagram. Section 4 concludes this paper and provides the guidelines for future work.

2. SURVEY ON TEST CASE GENERATION TECHNIQUES USING SEQUENCE AND ACTIVITY DIAGRAMS

This section deals with a survey on test case generation techniques using Sequence and Activity diagrams. Shirole and Kumar [2013] have done a survey on test case generation from UML behavioral diagrams. Subsequently, many more research articles have been published in this area till date. Hence, the authors of present paper are interested to conduct an extensive survey on test case generation from Sequence and Activity diagrams.

The UML diagrams are transformed into intermediate model to generate test cases. The authors have classified test case generation techniques basing on different approaches viz. Formal specification-based approach, Graphical representation approach, Heuristic approach, Direct UML specification processing approach, Hybrid behavior model approach and Concurrent model approach. These approaches are used to generate the test cases for functional testing of the software systems. The present survey is conducted based on above mentioned approaches.

The purpose of coverage criteria is to measure how far generated test suites covers the requirements. A Sequence diagram has coverage criteria like message, message-sequence, path etc. An Activity diagram has coverage criteria like activity, transition, simple-path, concurrent-path, etc. The following Tables I to VI depict survey conducted depending on various approaches. In this survey, we considered UML diagrams namely Sequence diagram, Activity Diagram, Class diagram, etc. as an input diagram. The methods/techniques used for test case generation are given in Method/Technique column. Furthermore, the intermediate model column in the table indicates types of intermediate models namely Sequence Flow Graph, Activity Graph, Message Flow Graph, etc.

2.1 Survey on test case generation based on Formal specification approach

The formal specification approach plays an important role in software testing. The formal specification of a system is used as a channel for designing functional tests for the system. The tester considers formal specification in order to understand the functionality of the system. The specifications are written followed by any standard formal notation like Z-notation, transition systems, object constraints language, etc. Test cases are generated from the formal specification which is useful to test SUT. When these test cases are used to test SUT, the test results indicate whether the implementation maps to the specifications. The generation of test cases from formal specification approach is a simple, structured, and more accurate.
Panthi and D.P.Mohapatra [2013] proposed Model Based Testing (MBT) approach. A function minimization technique is used to generate the test cases. Depth First Search (DFS) technique is used to select the associated predicates and to guess an initial dataset. Zhang et al. [2016] proposed an approach which enables to improve correctness of test cases and reduce the complexity of test cases generation process. Rhmann and Saxena [2016b] proposed a real case study of cash withdrawal functionality of an Aadhaar card based ATM. Sequence Flow Graph (SFG) is generated from Sequence diagram which is later used to generate the test cases for the proposed case study. Dehimi and Mokhati [2019] proposed an agent based test case generation method using a Sequence diagram. This method covers interactions between agents as well as possible scenarios. These interactions can be performed in an inclusive or exclusive or parallel way.Mu and Gu [2006] proposed a system test method in which formal specifications of the Activity diagram and the definition of test coverage rules are applied. Chen et al. [2009] proposed an approach to validate the consistency between the program execution traces and the behaviour of Activity diagrams. Chen et al. [2010] proposed another approach to reduce the validation efforts by reducing generation time of test cases as well as and required number of test cases. Due to this, it is very easy to meet the functional coverage criteria. Teixeira et al. [2016] proposed UML specification approach. The authors proposed “Easy Test “tool which is based on gray box technique. Summary of survey on test case generation based on Formal specification approach is shown in Table I.

2.2 Survey on test case generation based on Graphical representation approach

In this approach, UML diagrams are transformed into tree or graph representation. These representations help for generating test cases from UML diagrams in many different ways. Different graph traversal methods viz. Category Partition Method (CPM), Breadth-First Search (BFS), Depth First Search (DFS) and modified versions of BFS or DFS, etc. are used to generate test cases. The Graphical representation approach uses different types of intermediate models. For example, an Activity diagram is transformed into Binary Extended Tree (BET), Activity Diagram Composition Tree (ADCT), Activity Graph (AG), Activity Flow Graph (AFG), Order Relation Tree (ORT), Flow Dependency Graph (FDG), Model Flow Graph (MFG), Activity Dependency Table (ADT), Activity Dependency Graph (ADG), Extended Activity Dependency Graph (EADG), or Condition Classification Tree (CCT). A Sequence diagram is transformed into a Variable Assignment Graph (VAG), Extended Variable Assignment Graph, scenario tree, Sequence Diagram Graph (SDG), Message Dependency Graph (MDG).

Input Diagram	Authors	Method	Intermediate Model	Coverage Criteria
Sequence Diagram	Panthi and D.P.Mohapatra [2013]	Model checking	Predicates	Object, Message path, Full predicate
Sequence Diagram	Zhang et al. [2016]	Formal specification	Event deterministic finite automata	Basic and internal path
Sequence Diagram	Rhmann and Saxena [2016b]	Object Constraint Language (OCL)	Sequence Flow Graph	Message
Sequence Diagram	Dehimi and Mokhati [2019]	Agent based approach, OCL	Sequence Dependency Graph	Path
Activity Diagram	Mu and Gu [2006]	Formal specification	Test coverage rules	Specification
Activity Diagram	Chen et al. [2009]	Program execution traces	-	Program
Activity Diagram	Chen et al. [2010]	Model checking	Formal method model, Mapping rules	Function
Activity Diagram	Teixeira et al. [2016]	Grey Box	XML Metadata Interchange (XMI)	Path

Table I: Summary of survey on test case generation based on Formal specification approach

Samuel and Mall [2009] proposed a dynamic slicing technique in which the FDG having specific sets of nodes and edges is created. The slicing of these nodes is done and verified in FDG. Test cases are generated with respect to each slice.Swain et al. [2014] proposed a novel test case generation technique using Sequence diagrams. MDG is generated from a Sequence diagram and conditional predicates are selected to traverse MDG. The slices that correspondent to each conditional predicate are computed. The test cases are generated with respect to each slicing criterion. This technique can be used for system and cluster level testing considering the object message and condition information. The generated test cases are useful for detecting interactions between the objects and operational faults. Dhineshkumar [2014] presented a novel approach for test case generation from a Sequence diagram. In this approach, SDG is created using a Sequence diagram to generate test cases. The traceability between the models is provided by using Relational Definition Language.

Linzhang et al. [2004] proposed UMLTGF prototype tool which is developed by using a gray box technique. In this approach, test scenarios are directly derived from an Activity diagram. All the information, i.e. input/output sequence and parameters, the conditions and expected object method sequence is extracted from each test scenario for test case generation. Swain and Mohapatra [2010] generated MFG as intermediate model to generate test sequence using an Activity diagram. The test cases were generated from test sequences satisfying test adequacy criteria. Ray et al. [2009] proposed conditioned slicing method for test case generation. This method builds FDG from an Activity diagram and applies conditioned slicing on each predicate node of the graph to generate test cases.Samuel and Mall [2008] proposed dynamic slicing approach. In this approach, nodes of an activity graph are used to generate dynamic slices by using an edge marking method. With respect to each slice, test cases are generated. Mingsong et al. [2006] proposed AGTCG prototype tool. Thistool is implemented in JAVA programming language using improved DFS algorithm. This approach checks the consistency between specifications and corresponding programs. Boghdady et al. [2011] proposed a technique where each activity diagram is transformed into an intermediate models likes Activity Dependency Table (ADT) and Activity Dependency Graph (ADG). All possible test paths are generated by traversing ADG using DFS technique. The test paths are updated in the ADT to generate the final test cases automatically. Cyclomatic complexity is computed and used to validate the generated test cases. Tripathy and Mitra [2013] proposed TCG-SYS method for test case generation using system graphs. Chouhan et al. [2012] proposed a technique to generate test cases from an Activity diagram. A case study of navigation of mobile application is presented. In this proposed technique, Activity Dependency Table (ADT) is created from an Activity diagram. Finally, Activity Dependency Graph is generated from ADT to generate test cases. Swain et al. [2013] proposed a method for test

case generation from an Activity diagram. AFG is generated from an Activity diagram. All the required information like branches, conditions, executions and loop statements is extracted from AFG. The DFS technique is used to traversing an AFG. Monim and Nor [2018] presented MBT approach for the entire process. The authors proposed a model to extract, utilize, and prepare the data from Activity diagram into test case generation. A case study of ATM cash withdrawal system is taken for demonstration. The automated generated test cases are compared manually test case generating techniques. The authors found that there is no difference between manually written test cases and automatically generated test cases. Heinecke et al. [2010] considered Inputs and Outputs that are depicted through an Activity diagram and generated an Interaction Flow Diagram (IFD).This IFD represents the control flow of given an activity diagram. An IFD is mapped with the interaction flow graph which is having nodes and edges. This IFD is used to generate test cases. Thanakorncharuwit et al. [2016] proposed a model for test case generation based on business flow constraints. Test cases are generated from the Activity graph and validated with test coverage criteria. The different sets of rules are applied for loops structures as well as forks and joins. Mahali et al. [2016] presented an Input/Output Activity Diagram (IOAD) which is constructed from Activity diagram. The intermediate Activity graph is traversed using BFS to generate the test cases.

Boghdady et al. [2011] proposed an enhanced approach of converting Activity diagram to XMI. Further, this XMI is converted to ADT consisting of different nodes, forks, joins etc. ADG is generated using ADT. Test cases are generated from this ADG for every condition by using DFS Algorithm. The Cyclomatic complexity is used to validate the generated test cases. Hashim and Salman [2011] proposed an improved algorithm to generate test cases from Activity diagram using an Activity graph. These generated test cases are compared with manually generated test cases to evaluate usability and reliability of algorithm. Pechtanun and Kansomkeat [2012] used Activity Convert grammar to generate test cases from an Activity diagram. Tiwari and Gupta [2013] proposed an approach to generate safety validation test cases. Software Fault Tree (SFT) and Software Success Tree (SST) are generated from an Activity diagram to generate test cases. SST is used to test the normal behaviour of the system. SFT is used to test an exceptional behaviour of the system. SST and SFT are analyzed to generate the minimum cut sets to generate the test cases. Hettab et al. [2013] used graph transformation approach of inputs and outputs of diagrams. The authors proposed grammars for transformation of graph. This grammar is used to transform Activity diagrams into ORT Models and generate paths from ORT. Hettab et al. [2015] proposed two graph grammars. In first graph grammar, an activity diagram is transformed into an intermediate model called Extended Activity Dependency Graph (EADG). The EADG model captures information from an Activity diagram which is relevant to the generation of test cases. In second graph grammar, test cases are generated from EADG by traversing path. Li et al. [2013] proposed a simple approach of generating test cases from the Euler circuit. Test cases are minimized that satisfies activity state coverage criteria.Dalal and Hooda [2017] proposed a technique for test case generation of aspect-oriented programs based upon an Activity diagram. In this technique, the test cases are generated corresponding to decision-to-decision-graph and faults are verified to obtain coverage criteria. Summary of survey on test case generation based on Graphical representation approach is shown in Table II.

2.3 Survey on test case generation based on Heuristic approach

In this approach, In order to generate optimized test cases, several meta-heuristic techniques viz. Hill-climbing, Simulated annealing, Tabu search, Genetic algorithm, Particle swarm optimization, etc. can be used.

Jena et al. [2015] proposed an approach to generate test cases from a Sequence diagram. A sequence flow chart is generated from a Sequence diagram and a Message Control Flow Graph (MCFG) is generated from a sequence flow chart. The test paths are generated by traversing MCGF. The test cases for these paths are generated by using Genetic algorithm. In this approach, message, sequence and path coverage criteria is achieved. Biswal et al. [2010] proposed

Input Diagram	Authors	Method	Intermediate Model	Coverage Criteria
Sequence diagram	Samuel and Mall [2009]	Dynamic slicing	MDG	Slice
Sequence diagram	Swain et al. [2014]	Slicing technique	MDG	Slice, Path, Full predicate
Sequence diagram	Dhineshkumar [2014]	Iterative deepening DFS	Tree, Graph	Path
Activity diagram	Linzhang et al. [2004]	CPM	Constraint conditions	Condition
Activity diagram	Swain and Mohapatra [2010]	CPM	MFG	Message, Path
Activity diagram	Ray et al. [2009]	Conditioned slicing	FDG	Path
Activity diagram	Samuel and Mall [2008]	Dynamic slicing	FDG	Path, Full predicate
Activity diagram	Mingsong et al. [2006]	DFS	Test adequacy criteria	Path
Activity diagram	Boghdady et al. [2011]	DFS	ADT, ADG	Branch, Predicate, Path
Activity diagram	Tripathy and Mitra [2013]	DFS	System Graph	Path
Activity diagram	Chouhan et al. [2012]	DFS	ADT, ADG	Branch, Full predicate, Path
Activity diagram	Swain et al. [2013]	DFS	AFG	Path
Activity diagram	Monim and Nor [2018]	DFS	AFG	Path
Activity diagram	Heinecke et al. [2010]	Modified DFS	Interaction Flow Diagram	Path
Activity diagram	Thanakorncharuwit et al. [2016]	Modified DFS	Activity Graph	Activity, Path, Transition
Activity diagram	Mahali et al. [2016]	BFS	I/O Activity Diagram	Path
Activity diagram	Boghdady et al. [2011]	XML Based approach	ADT, ADG	Branch, Predicate, Path
Activity diagram	Hashim and Salman [2011]	AG	-	Path
Activity diagram	Pechtanun and Kansomkeat [2012]	ACG	-	Path
Activity diagram	Tiwari and Gupta [2013]	FTA	Software Success and Fault Tree	Path
Activity diagram	Hettab et al. [2013]	Graph Transformation	ORT	Path
Activity diagram	Hettab et al. [2015]	Graph Transformation	Extended ADG	Path
Activity diagram	Li et al. [2013]	Euler Circuit Algorithm	-	Activity, State,Transition
Activity diagram	Dalal and Hooda [2017]	Aspect Oriented Programming	CFG	Transition

Table II: Summary of survey on test case generation based on Graphical representation approach

an approach in which an Activity diagram and constrained based Genetic algorithm technique to generate optimal test cases. Shanthi and MohanKumar [2012] applied Genetic algorithm to generate, optimize, validate and prioritize the test cases. The test cases generated using this approaches are useful to detect more faults like synchronization faults, loop faults. Jena et al. [2014] proposed an approach to generate an Activity Flow Table (AFT) from an Activity diagram and to generate AFG from AFT. Test paths are generated by traversing AFG to achieve activity coverage criterion. Test cases are generated by using Genetic algorithm. Sumalatha and Raju [2012] proposed a Greedy heuristic method to generate test cases from an Activity diagram. These test cases are optimized using Genetic algorithm. Singla [2015] proposed an approach in which ADG is generated from an Activity diagram. Test suites are generated using the DFS algorithm by traversing the ADG. Optimized test cases are generated from the test suites by

using Genetic algorithm.
Nanda et al. [2008] used a heuristic approach to find the best test case from an existing path coverage set. An Activity diagram is parsed and then heuristic rule is applied to path coverage set. The test cases are analyzed and evaluated based on their coverage criterion. Test cases having the same paths are identified using heuristic approach. Shanthi and Kumar [2012] applied Tabu search algorithm to generate, optimize, validate and prioritize the test cases from an Activity diagram. Rhmann and Saxena [2016a] proposed an approach to generate optimized test path from an Activity diagram using Firefly algorithm. An Information Flow Metric is used to calculate adjacency metric of Activity graph. Cyclomatic complexity and Information Flow Metric are used for prioritization of generated test paths. The proposed approach is suitable for earlier identification of the faults in the system. Arora et al. [2017] generated concurrent test scenarios from an Activity diagram using a bio-inspired approach called as an Amoeboid Organism Algorithm. The authors found that the proposed algorithm is better than the existing Ant colony optimization and Genetic algorithm in terms of number of feasible test scenarios generated. Summary of survey on test case generation based on Heuristic based approach is shown in Table III.

Input Diagram	**Authors**	**Method**	**Intermediate Model**	**Coverage Criteria**
Sequence Diagram	Jena et al. [2015]	Genetic algorithm	Message Control Flow Graph	Message
Activity diagram	Biswal et al. [2010]	Genetic Algorithm	Event generator	Transition
Activity diagram	Shanthi and MohanKumar [2012]	Genetic Algorithm	Activity Dependency Table	Path
Activity diagram	Jena et al. [2014]	Genetic Algorithm	Activity Flow Table	Activity
Activity diagram	Sumalatha and Raju [2012]	Genetic Algorithm, Greedy Heuristic	-	Path
Activity diagram	Singla [2015]	Genetic algorithm	Activity Dependency Graph	Path
Activity diagram	Nanda et al. [2008]	Heuristic method	Parsing	Path
Activity diagram	Shanthi and Kumar [2012]	Tabu Search	Activity Dependency Table	Path
Activity diagram	Rhmann and Saxena [2016a]	Firefly algorithm	Control Flow	Path
Activity diagram	Arora et al. [2017]	Amoeboid Organism Algorithm	XML Metadata Interchange	Scenario

Table III: Summary of survey on test case generation based on Heuristic based approach

2.4 Survey on test case generation based on Direct UML specification processing approach

In Direct UML specification approach, test cases are generated without generating any intermediate model. Object Management Group (OMG) defined a standard process called XML Metadata Interchange (XMI) to exchange metadata information. Direct processing of UML models has become simple by using XMI representation of UML models, parserslike Document Object Model (DOM) and Simple API of XML (SAX).
Zhen [2003] proposed a Markov Chain Usage Model to generate test cases from Sequence diagram. This model is a combination of statistical usage testing and specification-based testing. Costa et al. [2014] proposed MBT technique to derive structural test cases from a Sequence diagram. In this technique, four steps namely Parser, Test Case Generator, Script Generator and Executor are considered. The automation tools namely PletsCoverageJabuti and PletsCoverageEmma are derived from this technique.
Oluwagbemi and Asmuni [2015] proposed a method that is useful to generate test cases from an Activity diagram. Activity Flow Tree (AFT) is constructed from an Activity diagram. A

parser is used to extract information from AFT.Test cases are generated based on the activity sequences, associated descriptions and conditions of the constructed tree. The authors presented a case study of ATM withdrawal operation. Xu and Wu [2019] proposed an automatic test case generation method using System Modelling Language Activity diagram but not UML Activity diagram. This method generates test cases from an Activity diagram with a complex structure to meet the test adequacy criteria. Summary of survey on test case generation based on Direct UML specification processing approach is shown in Table IV.

Input Diagram	**Authors**	**Method/Model**	**Coverage Criteria**
Sequence diagram	Zhen [2003]	Markov Chain Usage Model	Message
Sequence diagram	Costa et al. [2014]	Parsing	Message
Activity Diagram	Oluwagbemi and Asmuni [2015]	UML specification	Path, Branch, Condition
Activity Diagram	Xu and Wu [2019]	Depth First Search	Path

Table IV: Summary of survey on test case generation based on Direct UML specification processing approach

2.5 Survey on test case generation based on Hybrid behavior model approach

Sometimes, generation of complete test suite may not be possible when we consider only one type of UML diagram. As each UML diagrams represent a system from different perspective, test case generation techniques from different UML diagrams should be considered. In Hybrid behavior model approach, test cases are generated from different UML diagrams. In this approach, at first, an UML diagram may be considered to get important information and next another UML diagram may be also considered for some more information so that generated test cases will be complete. For example, an Activity diagram may enhance test specific details from combined fragments of a Sequence diagram. A test case generation technique can take first input from an UML diagram and then validates test cases using another UML diagram.

Khurana et al. [2016] presented a novel approach for generation of test cases from Sequence, Activity and Use case diagrams. In this approach, Sequence Diagram Graph (SDG), Activity Diagram Graph (ADG) and Use case Diagram Graph (UDG) are generated Sequence, Activity and Use case diagrams respectively. A System Testing Graph (SYTG) is generated by integrating SDG, ADG and UDG. The test cases are generated by traversing System Testing Graph. These test cases are optimized using Genetic algorithm. The generated test cases are suitable to detect maximum number of faults like use case dependency, interaction, scenario, pre-post condition faults and error handling. Shah et al. [2016] presented a tool that generates automated test cases using Sequence and Class diagram without creating any intermediate model. Sequence diagram is generated from Class diagram that includes class name, methods, attributes and relationship between classes.

Khurana and Chillar [2015] proposed a technique for generating test cases using Sequence diagram and State chart diagram. In this approach, Sequence Graph and State chart Graph are generated from Sequence and State chart diagrams respectively. Finally, System Testing Graph (SYTG) is generated by integrating these two Graphs. After integrating these two graphs, appropriate test cases are generated for system testing. These test cases are useful to detect errors, correctness of the system, and different faults like pre and post conditions, interactions, message sequences and scenarios. State chart is useful to identify unit level faults whereas sequence diagram is useful to identify integration level faults. Finally, test cases are optimized using Genetic algorithm. Efendi and Asmuni [Efendi and Asmuni]] proposed technique to generate and validate the test cases using the automated tool named UML Sequence and State chart Test Case Generation (USSTCG). Vu et al. [2017] proposed an automated test case generation method using Sequence and Class diagrams with string constraints. Septian et al. [2017] presented a hybrid behavioral model and heuristic approach to generate and optimize test cases. Genetic algorithm is used for optimization of test cases. Arora and Bhatia [2018] proposed Agent based test case generation

approach using Class, Use cases and Activity diagrams. It is found that the use of UML diagrams results in better identification of changes in code and hence it leads to efficient test case generation. Biswal et al. [2008] presented TC-ASEC (Test Case-Activity, Sequence, Class) model to generate test cases from Activity, Sequence and Class diagrams. This model gives full coverage criteria with reuse of the UML diagrams by applying gray box testing technique. The proposed approach handles complex nested fork-joins by giving as high priority in activity state. Summary of survey on test case generation based on Hybrid behaviour model approach is shown in Table V.

Input Diagram	**Authors**	**Method**	**Intermediate Model**	**Coverage Criteria**
Sequence, Activity and Use case diagram	Khurana et al. [2016]	Genetic algorithm	System Testing Graph	Path
Sequence and Class diagram	Shah et al. [2016]	Parsing	XML Metadata Interchange	Object, Parameter
Sequence and State chart diagram	Khurana and Chillar [2015]	Genetic algorithm	System Testing Graph	Path
Sequence and State chart diagram	Efendi and Asmuni [Efendi and Asmuni]	UML specification approach	Sequence and State chart Control Flow Graph	Path, State
Sequence and Class diagram	Vu et al. [2017]	Pre-processing rules	Control Flow Graph	Path
Sequence and Activity diagram	Septian et al. [2017]	Modified DFS	System testing Graph	Path
Activity, Class and Use case diagram	Arora and Bhatia [2018]	Agent based approach	XML Metadata Interchange	Specification
Activity, Sequence and Class diagram	Biswal et al. [2008]	Gray box technique	Fork-Join	Full coverage

Table V: Summary of survey on test case generation based on Hybrid behaviour model approach

2.6 Survey on test case generation based on Concurrent model approach

The concurrent models are generally used in the Mission-critical systems. In this approach, the main focus is to generate test suites that meet concurrency coverage criteria. UML provides concurrent execution behavior in Sequence and Activity diagrams by using concurrent asynchronous messages and fork-join constructs respectively.

Khandai et al. [2011] presented an approach for generating test cases for concurrent systems. In this approach, a Concurrent Composite Graph (CCG) is generated from a Sequence diagram. The CCG is traversed using BFS and DFS techniques to generate the test cases for a concurrent system. This approach achieves message, sequence, and path coverage criteria. The authors found that proposed approach is very effective in handling the test case explosion problem. The generated test cases are useful to detect scenario, interaction and operational faults of concurrent systems.

Mani and Prasanna [2017] proposed an approach to generate efficient test cases from Sequence diagram. This approach uses stack array and boundary value techniques for embedded system. Yimman et al. [2017] proposed a dynamic programming approach. An Activity graph is generated from An Activity diagram by focusing on the concurrency problem. Then, dynamic programming technique is applied to generate all the paths from concurrent test cases. Kamonsantiroj et al. [2019] proposed a dynamic programming technique to generate test cases from the concurrent activities of Activity diagram. The authors proposed a Memoized- ConstPath algorithm to generate all activity paths. This approach fulfills concurrency coverage criteria. The result of the proposed technique is claimed to be more efficient than DFS and BFS techniques. Summary of survey on test case generation based on Concurrent model approach is shown in Table VI.

Input Diagram	Authors	Method	Intermediate Model	Coverage Criteria
Sequence Diagram	Khandai et al. [2011]	BFS, DFS	Concurrent composite graph	Message, Sequence,Path
Sequence Diagram	Mani and Prasanna [2017]	Stack array and boundary value	Stimulus linking table	Message, Path
Activity Diagram	Yimman et al. [2017]	Dynamic Programming	Activity graph	Complete path Activity, Transition
Activity Diagram	Kamonsantiroj et al. [2019]	Dynamic Programming	Activity paths	Activity and Causal ordering

Table VI: Summary of survey on test case generation based on Concurrent model approach

3. FEASIBILITY STUDY TO GENERATE COMBINATORIAL LOGIC ORIENTED TEST CASES USING SEQUENCE AND ACTIVITY DIAGRAMS

This section provides a feasibility study to generate combinatorial logic oriented test cases using Sequence and Activity diagrams. The basic elements like parameters, their values, interactions among parameters and constraints are very essential to generate combinatorial logic oriented test cases. Modeling is one of the most important steps of combinatorial testing because the subsequent steps are highly dependent on it. Combinatorial Test Design Model (CTDM) consists of these basic elements. Hence, it is necessary to derive CTDM using Sequence and Activity diagrams to generate combinatorial test cases. Nowadays, the test designers derive CTDM by looking into the requirements based on their knowledge and experience. Satish and Rangarajan [2016] proposed a method to generate CTDM which can be used for generating combinatorial logic oriented test cases.

An automatic extraction of parameters and their values from UML diagrams is a very important task in generation of combinatorial logic oriented test cases. Some researchers have used semi-automatic method for extraction of parameters and values from UML diagrams. Satish et al. [2013]Satish et al. [2014]Satish et al. [2017] have presented a rule based approach to extract parameters and values from UML Sequence, Activity and Use case diagrams. Since this process is semi-automatic, manual refinement of the model is necessary. Esfandyari and Rafe [2020] used model checking techniques to extract parameters and values. In model checking technique, all reachable states of a given system are transformed into a directed graph. The model checker traverses the state space completely to generate test paths.

As mentioned earlier, CTDM is very much useful to generate the test cases. After the survey, the authors opine that there is a scope to extend CTDM so that it is useful to generate Combinatorial Logic Oriented Test cases. Various components in Software Development Life Cycle viz. software requirements specifications, UML design artifacts, source code, test scenarios, etc. are to be considered while designing CTDM. In case, if the customer provides Acceptance test cases, then it may be possible to generate Combinatorial logic oriented acceptance test cases by further extending CTDM. While generating combinatorial logic oriented test cases, the following challenges are to be tackled.

(1) To extract parameters, values of parameters, interaction among parameters and constraints from Sequence and Activity diagrams. In most of the existing approaches, this information is given as an input to the system manually which may be an error prone task.
(2) To handle the situation in which the size of extracted information from Sequence and Activity diagrams is too large.

There is scope to extract these parameters, values and constraints in an automatic manner using new techniques. We conducted a feasibility study in order to know whether UML diagrams can be used to generate combinatorial logic oriented test cases and our finding is that" it is feasible".

The following steps may be adapted to generate combinatorial logic oriented test cases.

(1) The necessary UML diagrams are to be given as input to the test case generator.
(2) The parameters, values and constraints can be extracted automatically from UML diagrams by using new techniques.
(3) Efficient algorithms including evolutionary algorithms have to be designed to generate combinatorial logic oriented test cases by using extracted input parameters, values and constraints.

4. CONCLUSION AND FUTURE SCOPE

This paper provides a systematic survey on automatic test case generation from Sequence and Activity diagrams. The survey covers 80 research papers. Various researchers generated test cases from Sequence and Activity diagrams by using different techniques and methods.
The outcome of feasibility study indicates that it is possible to generate combinatorial logic oriented test cases using UML Sequence and Activity diagrams. As the feasibility study is positive, in future, the research work can be carried to realize the aforesaid generation of combinatorial logic oriented test cases using UML Sequence and Activity diagrams.

References

Arora, P. and Bhatia, R. 2018. Agent-based regression test case generation using class diagram, use cases and activity diagram. *Procedia Computer Science Vol.125,* No, pp. 747–753.

Arora, V., Bhatia, R., and Singh, M. 2017. Synthesizing test scenarios in uml activity diagram using a bio-inspired approach. *Computer Languages, Systems Structures Vol.50,* No, pp. 1–19.

Bewoor, L. A., Chandraprakash, V., and Sapkal, S. 2019. Evolutionary hybrid particle swarm optimization algorithm to minimize makespan to schedule a flow shop with no wait. *Journal of Engineering Science and Technology Vol.14,* No 2, pp.609–628.

Biswal, B., Nanda, P., and Mohapatra, D. 2008. A novel approach for scenario-based test case generation. *IEEE Vol.,* No, pp. 244–247.

Biswal, B., Nanda, P., and Mohapatra, D. 2010. A novel approach for optimized test case generation using activity and collaboration diagram. *International Journal of Computer Applications Vol.1,* No 14, pp. 67–71.

Boghdady, P., Badr, N., Hashemand, M., and Tolba, M. 2011. A proposed test case generation technique based on activity diagram. *International Journal of Engineering Technology Vol.11,* No 3, pp.1–21.

Boghdady, P., Badr, N., Hashim, M., and Tolba, M. 2011. An enhanced test case generation technique based on activity diagrams. *IEEE Vol.,* No, pp.289–294.

Briand, L. and Labiche. 2002. A uml-based approach to system testing. *Software and Systems Modeling Vol.1,* No.1, pp.10–42.

Chandraprakash, V. and Kadiyala, P. 2006. Automatic test generation: A use case driven approach. *IEEE Transactions on Software Engineering Vol.32,* No.3, pp.140–155.

Chen, M., Mishra, P., and Kalita, D. 2010. Efficient test case generation for validation of uml activity diagrams. *Design Automation for embedded systems Vol.14,* No.2, pp.105–130.

Chen, M., Qiu, X., Xu, W., Wang, L., Zhao, J., and Li, X. 2009. Uml activity diagram-based automatic test case generation for java programs. *The Computer Journal Vol.52,* No.5, pp.545–556.

Chouhan, C., Shrivastava, V., and P.S.Sodhi. 2012. Test case generation based on activity diagram for mobile application. *International Journal of Computer Applications Vol.57,* No 23, pp.

Costa, L., Zorzo, A., Rodrigues, E., Silveira, M., and Oliveira, F. 2014. Structural test case generation based on system models. *International Conference on Software Engineering Advances Vol.,* No, pp. 276–281.

Dalal, S. and Hooda, S. 2017. Automated test sequence generation of aspect-oriented programs based upon uml activity diagram. *International Journal of Engineering and Technology Vol.9,* No 2, pp.

Dehimi, N. and Mokhati, F. 2019. Novel test case generation approach based on auml sequence diagram. *IEEE Vol.,* No.

Dhineshkumar, M. 2014. An approach to generate test cases from sequence diagram. *IEEE Vol.8,* No 6, pp.345–349.

Efendi, N. and Asmuni, H. Exhaustive search for test case generation from uml sequence diagram and statechart diagram. *Vol.,* No, pp.

Esfandyari, S. and Rafe, V. 2020. Extracting combinatorial test parameters and their values using model checking and evolutionary algorithms. *Applied Soft Computing Vol.,* No, pp.

Gouda, R. and Chandraprakash, V. 2019. Optimization driven constraints handling in combinatorial interaction testing. *International Journal of Open Source Software and Processes Vol.10,* No 3, pp.19–37.

Hartmann, J., Vieira, M., H.Foster, and Ruder. 2005. A uml-based approach to system testing. *Innovations in Systems and Software Engineering Vol.1,* No.1.

Hashim, N. and Salman, Y. 2011. An improved algorithm in test case generation from uml activity diagram using activity path. *Vol.,* No, pp.

Heinecke, A., Brückmann, T., Griebe, T., and Gruhn, V. 2010. Generating test plans for acceptance tests from uml activity diagrams. *IEEE Vol.,* No, pp.57–66.

Hettab, A., Chaoui, A., and Aldahoud, A. 2013. Automatic test cases generation from uml activity diagrams using graph transformation. *ICIT Vol.,* No, pp.

Hettab, A., Kerkouche, E., and Chaoui, A. 2015. A graph transformation approach for automatic test cases generation from uml activity diagrams. *International Conference on Computer Science Software Engineering Vol.,* No, pp.88–97.

Jena, A., Swain, S., and Mohapatra, D. 2014. A novel approach for test case generation from uml activity diagram. *IEEE Vol.,* No, pp.621–629.

Jena, A., Swain, S., and Mohapatra, D. 2015. Test case creation from uml sequence diagram: a soft computing approach. *Springer Vol.,* No, pp. 117–126.

Kamonsantiroj, S., Pipanmaekaporn, L., and Lorpunmanee, S. 2019. A memorization approach for test case generation in concurrent uml activity diagram. *International Conference on Geoinformatics and Data Analysis Vol.,* No, pp. 20–25.

Khandai, M., Acharya, A., and Mohapatra, D. 2011. A novel approach of test case generation for concurrent systems using uml sequence diagram. *IEEE Vol. 1,* No, pp.157–161.

Khurana, N., Chhillar, R., and Chhillar, U. 2016. A novel technique for generation and optimization of test cases using use case, sequence, activity diagram and genetic algorithm. *Journal of Software Engineering Vol.11,* No 3, pp. 242–250.

Khurana, N. and Chillar, R. 2015. Test case generation and optimization using uml models and genetic algorithm. *Procedia Computer Science Vol.57,* No, pp.996–1004.

Kondhalkar, V. and ChandraPrakash, V. 2018. Automated generation of test cases for conducting pairwise plus testing. *Journal of Advanced Research in Dynamical and Control Systems Vol.,* No, pp.1484–1492.

LakshmPrasad, M., Reddy, A. R. S., and Sastry., J. 2019. Gapso: Optimal test set generator for pairwise testing. *International Journal of Engineering and Advanced Technology Vol.8,* No 6, pp.

Li, L., Li, X., He, T., and Xiong, J. 2013. Extenics-based test case generation for uml activity diagram. *Procedia Computer Science Vol.17,* No, pp.1186–1193.

Linzhang, W., Jiesong, Y., Xiaofeng, Y., Jun, H., Xuandong, L., and Guoliang, Z. 2004. Generating test cases from uml activity diagram based on gray-box method. *IEEE Vol.,* No, pp.284–291.

Mahali, P., Arabinda, S., Acharya, A., and Mohapatra, D. 2016. Test case generation for concurrent systems using uml activity diagram. *IEEE Vol.,* No, pp. 428–435.

Mani, P. and Prasanna, M. 2017. Test case generation for embedded system software using uml interaction diagram. *Journal of Engineering Science and Technology Vol.12,* No 4, pp.860–874.

Mingsong, C., Xiaokang, Q., and Xuandong, L. 2006. Automatic test case generation for uml activity diagrams. *international workshop on Automation of software test Vol.,* No, pp.2–8.

Monim, M. and Nor, R. 2018. An automated test case generating tool using uml activity diagram. *International Journal of Engineering Technology Vol.7,* No 4, pp.58–63.

Mu, K. and Gu, M. 2006. Research on automatic generating test case method based on uml activity diagram. *Journal of Computer Applications Vol.4,* No.

Mudarakola, L. P., Sastry, J. K., and Prakash, V. C. 2018. Testing embedded systems using test cases generated through combinatorial techniques. *International Journal of Engineering Technology Vol.7,* No 2, pp.146–158.

Nanda, P., Biswal, B., and Mohapatra, D. 2008. A novel approach for test case generation using activity diagram. *ICIT Vol.1,* No 1, pp.60–63.

Nebut, C., F., F., Y., L. T., and J.M., J. 2006. Automatic test generation: A use case driven approach. *IEEE Transactions on Software Engineering Vol.32,* No.3, pp.140–155.

Oluwagbemi, O. and Asmuni, H. 2015. Automatic generation of test cases from activity diagrams for uml based testing (ubt). *Jurnal Teknologi Vol.77,* No 13, pp.

Panthi, V. and D.P.Mohapatra. 2013. Automatic test case generation using sequence diagram. *Springer Vol.,* No, pp.277–284.

Pechtanun, K. and Kansomkeat, S. 2012. Generation test case from uml activity diagram based on ac grammar. *IEEE Vol.,* No, pp. 895–899.

Prakash, V., S.Tatale, V.Kondhalkar, and L.Bewoor. 2018. A critical review on automated test case generation for conducting combinatorial testing using particle swarm optimization. *Int. J. Eng. Technol. Vol.7,* No 3, pp.

Prasad, M. and J.K.R.Sastry. 2018a. Building test cases by particle swarm optimization (pso) for multi output domain embedded systems using combinatorial techniques. *Journal of Advanced Research in Dynamical and Control System Vol.6,* No, pp.1221–1229.

Prasad, M. and J.K.R.Sastry. 2018b. A graph based strategy (gbs) for generating test cases meant for testing embedded systems using combinatorial approaches. *Journal of Advanced Research in Dynamical and Control System Vol.10,* No 1, pp.314–324.

Ramgouda, P. and Chandraprakash, V. 2018. Neural network based approach for improving combinatorial coverage in combinatorial testing approach. *Journal of Theoretical and Applied Information Technology Vol.20,* No 96, pp.6677–6687.

Ramgouda, P. and Chandraprakash, V. 2019. Constraints handling in combinatorial interaction testing using multi-objective crow search and fruitfly optimization. *Soft Computing Vol.23,* No 8, pp.2713–2726.

Ray, M., Barpanda, S., and Mohapatra, D. 2009. Test case design using conditioned slicing of activity diagram. *International Journal of Recent Trends in Engineering Vol.1,* No 2, pp.117.

Rhmann, W. and Saxena, V. 2016a. Optimized and prioritized test paths generation from uml activity diagram using firefly algorithm. *International Journal of Computer Applications Vol.145,* No 6, pp.16–22.

Rhmann, W. and Saxena, V. 2016b. Test case generation from uml sequence diagram for aadhaar card number based atm system. *System Vol.11,* No 4, pp.1221–1229.

Samuel, P. and Mall, R. 2008. A novel test case design technique using dynamic slicing of uml sequence diagrams. *e-Informatica Vol.2,* No 1, pp.71–92.

Samuel, P. and Mall, R. 2009. Slicing-based test case generation from uml activity diagrams. *ACM SIGSOFT Software Engineering Notes Vol.34,* No 6, pp.1–14.

SasiBhanu, J., D.Baswaraj, Bigul, S. D., and Sastry, J. 2019. Generating test cases for testing embedded systems using combinatorial techniques and neural networks based learning model. *International Journal of Emerging Trends in Engineering Research Vol.7,* No 11, pp.417–429.

SasiBhanu, J., LakshmiPrasad, M., and Sastry, D. J. 2018. A combinatorial particle swarm optimization (pso) technique for testing an embedded system. *Journal of Advanced Research in Dynamical and Control System Vol.10,* No 7, pp.321–336.

Satish, P., Milind, B., Narayan, M., and Rangarajan, K. 2017. Building combinatorial test input model from use case artefacts. *IEEE Vol.,* No, pp. 220–228.

Satish, P., Paul, A., and Rangarajan, K. 2014. Extracting the combinatorial test parameters and values from uml sequence diagrams. *IEEE Vol.,* No, pp.88–97.

Satish, P. and Rangarajan, K. 2016. A preliminary survey of combinatorial test design modeling methods. *International Journal Of Scientific Engineering Research Vol.7,* No 7, pp.1455–1459.

Satish, P., Sheeba, K., and Rangarajan, K. 2013. Deriving combinatorial test design model from uml activity diagram. *IEEE Vol.,* No, pp.331–337.

Septian, I., Alianto, R., and Gaol, F. 2017. Automated test case generation from uml activity diagram and sequence diagram using depth first search algorithm. *Procedia computer science Vol.116,* No, pp. 629–637.

Shah, S., Shahzad, R., Bukhari, S., and Humayun, M. 2016. Automated test case generation using uml class sequence diagram. *Current Journal of Applied Science and Technology Vol.,* No, pp. 1–12.

Shanthi, A. and Kumar, G. 2012. A heuristic technique for automated test cases generation from uml activity diagram. *i-Manager's Journal on Software Engineering Vol.6,* No 3, pp.

Shanthi, A. and MohanKumar, G. 2012. A novel approach for automated test path generation using tabu search algorithm. *International Journal of Computer Applications Vol.48,* No 13, pp.28–34.

Shirole, M. and Kumar, R. 2013. Uml behavioral model based test case generation: a survey. *ACM SIGSOFT Software Engineering Notes Vol.38,* No 4, pp.1–13.

Singla, I. 2015. A semantic approach for the generation of test cases from activity diagram. *International Journal of Computer Applications Vol.116,* No 10, pp.

Sumalatha, V. and Raju, G. 2012. Uml based automated test case generation technique using activity-sequence diagram. *International Journal of Computer Science Applications Vol.1,* No 9, pp.

Swain, R., Panthi, V., and Behera, P. 2013. Generation of test cases using activity diagram. *International journal of computer science and informatics Vol.3,* No 2, pp.1–10.

Swain, R., Panthi, V., Behera, P., and Mohapatra, D. 2014. Slicing-based test case generation using uml 2.0 sequence diagram. *International Journal of Computational Intelligence Studies Vol.3,* No 2, pp.221–250.

Swain, S. and Mohapatra, D. 2010. Test case generation from behavioral uml models. *International Journal of computer applications Vol.6,* No 8, pp.5–11.

Teixeira, Diniz, F. A., and e Silva, G. B. 2016. Easytest: an approach for automatic test cases generation from uml activity diagrams. *Springer Vol.,* No., pp.

Thanakorncharuwit, W., Kamonsantiroj, S., and Pipanmaekaporn, L. 2016. Generating test cases from uml activity diagram based on business flow constraints. *International Conference on Network, Communication and Computing Vol.,* No, pp.155–160.

Tiwari, S. and Gupta, A. 2013. An approach to generate safety validation test cases from uml activity diagram. *IEEE Vol.,* No, pp. 189–198.

Tripathy, A. and Mitra, A. 2013. Test case generation using activity diagram and sequence diagram. *Springer Vol.10,* No 7, pp. 121–129.

Vu, T., Hung, P., and Nguyen, V. 2017. A method for automated test cases generation from uml models with string constraints. *Springer Vol.,* No, pp.525–536.

Xu, Y. and Wu, L. 2019. An automatic test case generation method based on sysml activity diagram. *Materials Science and Engineering Vol.563,* No 5, pp.

Yimman, S., Kamonsantiroj, S., and Pipanmaekaporn, L. 2017. Concurrent test case generation from uml activity diagram based on dynamic programming. *International Conference on Software and Computer Applications Vol.,* No, pp. 33–38.

Zhang, C., Duan, Z., Yu, B., Tian, C., and Ding, M. 2016. A test case generation approach based on sequence diagram and automata models. *Chinese Journal of Electronics Vol.25,* No 2, pp.234–240.

Zhen, F. 2003. Automated ttcn-3 test case generation by means of uml sequence diagrams and markov chains. *Test Symposium Vol.,* No, pp. 102–105.

Subhash B. Tatale has completed BE (Computer), (M.Tech. Information Technology). Currently, he is pursuing Ph.D. in Computer Science and Engineering at Koneru Lakshmaiah Education Foundation, Vaddeswaram, India. He is having 13 years of experience in teaching and software industry. His research area includes Software Engineering, Unified Modeling Language and Artificial Intelligence. He has published research papers in national and international conferences and journals. He has the membership of International Association of Engineers (IAENG).

Dr. V. Chandra Prakash has been working as professor in Computer Science and Engineering at Koneru Lakshmaiah Education Foundation, Vaddeswaram, India for the past 35 years. He also has 10 years of experience in software industry. His research interests include Artificial Intelligence, Machine Learning, Cognitive Science, Cognitive Computing and Software Engineering. He has published research papers in reputed journals having indexing like Scopus, Web of Science etc.