http://ansinet.com/itj

ITJ

ISSN 1812-5638

INFORMATION TECHNOLOGY JOURNAL

ANSInet
Asian Network for Scientific Information
308 Lasani Town, Sargodha Road, Faisalabad - Pakistan

Information Technology Journal 12 (8): 1600-1606, 2013
ISSN 1812-5638 / DOI: 10.3923/itj.2013.1600.1606

An Approach to Generate Test Goals from Use Case Scenarios

Azhar Mahmood and Shaheen Khatoon
School of Computer and Applied Technology, Huazhong University
of Science and Technology, Wuhan, China

Abstract: Scenarios are used to describe the functionality and behavior of a system. Scenarios are an important mechanism for requirements specification and can be used to generate test goals at requirements analysis level. By using scenarios in system testing, we are able to start testing at very early stage of software development. By eliminating model defects before the coding begins and the test case creation results in significant cost savings and higher quality code because later the defects captured are more costly in term of effort and time. Current approaches for system testing are using a use cases scenario which involves functional details that seem to be difficult at very initial level of software development. In this study, an approach is proposed for system testing directly derived from the specification without involving functional details. Pre and post conditions to use cases are utilized a guard that enables the generation of formalized test cases; also contracts are added to each level that makes it possible to generate test cases for each flow of the system. The contribution of this study is to provide an approach for testing software system that not only helps in starting testing at early stages of software development, but also provides a mechanism to elaborate and refine the specifications.

Key words: Requirement validation, scenarios, use cases, system testing, contracts

INTRODUCTION

A UML use case is a system specific usage model enabling identification of complete requirements and analysis consisting of collection of scenarios whereas use case analysis provide a key for complete understanding of the system (Briand and Labiche, 2002). By using UML technique user requirements are stated in terms of use cases that can better combine the user needs with the system behavior in the form of user-system interaction when the system will be in operation.

A use case based testing deal with capturing of user requirements and the generation of test cases for the system at early stage in the engineering process and validating the tests with the specification of the system. Hence, use cases may provides a foundation for the system level testing (Raza *et al*., 2007). The basic principle behind the system testing is to verify the functional and performance aspects of the intended system; alternatively the system is tested and compared to its specification which verifies the results and functionality of the system. UML use case testing validates the user requirements with an intention of clarifying what the user actually requires from the system. The objective of using use case based testing is the generation of test cases at early stage of software development which can help to identify the unclear requirements (Blackburn *et al*., 2004).

In this study, an approach is presented for system level testing based on UML analysis artifacts such as system use case diagram, scenario diagram and sequence diagram in order to generate system test requirements. UML use cases are used to model entire system usage flow whereas use case scenario expresses the execution flow of a use case that can be used to extract sequence diagram. We are enhancing the concept described (Whittle and Jayaraman, 2006) by using contractual use cases. In the proposed approach contracts are added to the use cases that helps to capture the sequential events alternatively representing complete flow of the system by which the system has to be pass while execution. Test requirements are generated as logical expressions with the help of contracts, as requirement level logical expression allows requirement validation and test case generation early in the design phase.

Corresponding Author: Azhar Mahmood, School of Computer and Applied Technology,
Huazhong University of Science and Technology, Wuhan, China

RELATED WORK

Ryser and Glinz (1999) presented a technique for the description of use cases with scenarios using state chart and deriving test cases from state charts in a systematic manner. Scenarios are created from requirement natural language by creating a step-by-step description of events and actions perform by the system. Narrative scenarios are then transformed into state charts; test cases for system test are generated by path traversal in the state charts. Furthermore, a comprehensive notation for modeling scenario dependencies has been proposed, however the approach is manual and state chart generation process is declared to be a creative activity and is achieved solely by the experience of the tester.

Briand and Labiche (2002) proposed an approach that involves use case diagram, activity diagram and sequence diagram for the generation of system level tests cases. Use case dependencies are modeled by using activity diagram and the functionality of the system is modeled by using class diagrams. The sequential constraints are used to generate test requirements which is described in meta-model and contains formal description of class, operators and contracts. Concrete test cases are obtained in the form of regular expressions written in OCL. However, there are several limitations exists in the work, first the test criterion is based on the coverage of the regular expressions obtained by the projection from the activity diagram. This criterion leads to a very large number of test cases. We believe that test criteria have to be found, leading to a more realistic number of test cases. Second, the activity diagram is either incomplete to generate significant test sequences (so all functions are not covered) or complex to define (with the risk of specifying infeasible use-case sequences). Moreover, not all the interactions between actors are taken into account.

Nebut *et al.* (2003) proposed an approach by the inspiration of Briand and Labiche (2002) on UML based approach for system testing. Contract language for requirements is defined as pre and post conditions associated as logical expression. To generate test objectives Use Case Transition System (UCTS) is build. UCTS is a valid sequence of use cases representing the possible ordering of instantiated use cases. Instantiated use cases are obtained by replacing their set of formal parameters by all possible combination of their effective values. The use case transition system represents all the valid sequence of use cases referred to as path. From UCTS test objectives are generated that are finite sequence of instantiated use cases and most test objectives are not directly implemented. Test objectives define a finite sequence of use cases if each test objective traverse path in UCTS then these test objectives are said to be consistent.

Raza *et al.* (2007) proposed a test path generation approach for scenarios using Interaction Overview Diagram (IOD). From the operational contracts in IOD a Contract Transition System (CTS) is built that specifies the pre and post conditions. A CTS metric is developed base on the operations in IOD that shows states and contracts in the CTS. Based on CTS metrics scenarios are generated for each use case and finally test path are created by applying coverage criteria i.e. all transition coverage or all state coverage.

Hsia *et al.* (1994) presented a tree based approach for generating scenarios from use cases. Scenario tree consist of nodes and arc corresponding to states and events. Scenarios are formally stated by using the regular expression that results into deterministic finite state machine with a single state that defines it's both initial and final state. Kosters *et al.* (1997) presented an approach for mapping use cases into static classes and methods. Directed graph are used to describe use case where a node inherits the scenarios. The technique transform he scenario steps into action by using tree methods. Alspaugh *et al.* (2005) proposed requirement based V and V model in order to develop requirement scenario description language named " ScenarioML", used to generate test goals from functional requirements. An event based tree is generated from functional requirements. Test goals are generated by using test coverage metric that covers all the sub-goals in the event tree. A test suits consist of set of event traces that integrally provide goal coverage. Kim *et al.* (1999) proposed an approach for class testing by using a set of coverage criteria based on data and control flow in UML state diagram. The state diagram shows the basics and composite states and described as OR-State/AND-State. States can have actions that contains list of operations for transition being occur. Test cases are generated by either using control flow or data flow method.

Most of the approaches present in the literature involved more functional details e.g. Briand and Labiche (2002) uses class diagram which demand more functional detail which cannot be captured at early stages of requirement analysis. Whereas, Whittle and Jayaraman (2006) focused on hierarchical state machine generation from scenarios. First, use case scenarios are created for each use case, then node sequence diagram are created. Finally, a hierarchical state chart is generated by combining node sequences. Since, contracts are not applied in sequence diagrams, hence testing and test

criteria are not the objective of this approach. Nebut *et al.* (2003) presented approach is a UML based system level testing. It defines the contract language for requirement as pre and post condition associated as logical expression.

We have presented an approach that has inspiration from Briand and Labiche (2002) and Nebut *et al.* (2003) work. Our proposed approach differs with the fact that we are taking into account only the specification of the system without involving the functional details so a level above on the specification by capturing the sequential ordering of the use cases with the guard annotation defined as contracts. Addition of contracts in the proposed approach is closer to the way Nebut *et al.* (2003) applied the contracts to use cases whereas Briand and Labiche (2002) and Whittle and Jayaraman (2006) does not imposed contracts. The proposed approach applied contracts on the use cases to capture the sequential dependencies in scenarios. The annotation of contracts on the scenario is used to generate the test objectives. Whereas, Nebut *et al.* (2003) does not imposed contracts on scenarios. Furthermore test objectives are generated based on the coverage criteria. The advantage of generating test objectives from contracts makes them executable by expressing as logical expression. The proposed approach also captures use case flow model and contracts from the specification. Additionally, it makes conditional testing easy and can be defined as logical expression. Our contribution to literature is the extraction of sequential dependencies of use cases involving use cases contracts and extraction of test objectives from the scenario's contracts both expressed as logical expression.

PROPOSED APPROCH

This section describes the proposed scenarios based approach for system level testing. It consists of following steps:

- Create an overall system use case design diagram
- Generation of sequential use case diagram
- Extracting sequential constructs for use cases
- Deriving the second level use case scenario diagrams where each node express the level-1 use case node with contracts
- Generating execution contracts to level-2 scenario use cases as a logical expression
- Extraction of test goals from contracts

In the proposed approach as shown in Fig. 1, the first level of the diagram represents the entire system usage

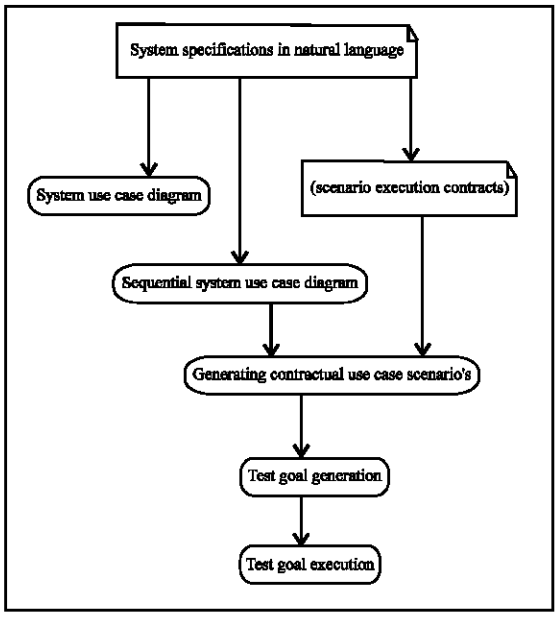

Fig. 1: Abstract model of scenario based approach

where nodes are use cases. The entire use case diagram is then modeled with sequential events which represents the use case sequences and the complete flow of the system by which it has to be passed during execution. Each use case node is then further modeled into use case scenario at level-2, where each use case scenario actually reflects the use case flow of each level-1 use case node and then at level-3 each level-2 use case scenario is transformed into sequence diagram. Sequence diagrams represent the interactions as messages at edges between the lifelines with the participants where life line increases vertically downwards.

Overall system use case diagram: The overall use case design diagram shows the entire system view showing the major actors involving in the system with the interacting system elements in the form of use cases. A UML use case allows the modeling and understanding of complete requirements. Use case models are purely based on the requirement specification with the major emphasis is to understand the actual problem domain without involving any implementation and functional details with an aim of completely capturing the user requirements and formally describe them. For the creation of use cases the system under test should be completely understood and modeled (El-Far and Whittaker, 2001) by capturing the complete response and sequence of events or inputs that needed to be modeled. The number of use cases may be very large in the system as each use case originally describe a single activity performed by a combination of user/system

interaction and may include several actors. Each of the use case contains its own set of events to occur, therefore the entire system use case diagram can comprises of several use case nodes by involving the interacting actors.

Generation of sequential use case diagram: A use case model can be transformed into functional requirement specification with structuring and formalization of use cases and can be used in acceptance testing as it involves requirement specification validation with the users. A use case based requirement validation requires that the sequential ordering of the use cases should be captured in behavioral model. Use case sequences can be expressed by using the pre and post conditions that may become the contracts. The use case sequential flow describes how the use cases follow each other and gives a clear idea of system usage (Some, 2007). Some use cases of the system can be run independently of others while some use cases may have sequential dependencies between them indicating the execution order of the use cases. Sequential execution of use cases can be the first component of system test requirements.

Extracting sequential constraints for use cases: The sequential constraints between the use cases can be specified by using the logical expression with the AND/OR operators, where the OR operator show the alternative paths in the execution sequential order. We are adding contracts to the use cases so the sequential contracts will be made with the combination of guards/contracts.

Generation of use case scenario diagram: A scenario specifies sequence of events for a use case. We are generating scenario chart from the specification of the system and adding the contracts to the scenario nodes where the contracts actually depicts the passing criteria to go to the next stage. Addition of contracts allows the requirement validation and test case generation.

Generating execution contracts: The execution contracts are generated from the use case scenario by adding the contracts applied to the sequential constructs, where as the alternative path are covered by ordering the decision conditions.

Test goal extraction: Test goal specifies the objective from the test i.e. what the user or tester require from the system should be identified separately. Identification of goals gives confidence to testing, with the introduction of goals the use case scenario either completed with success or fails, the goal plan also includes the alternatives as well (Alspaugh *et al*., 2005). We have applied contracts at the use case scenario that will be used to define the test goals which can be executed by routing through the path at the state diagram. The primary advantage of using contracts is the definition of test goals but these should be consistent while moving from one stage to other in order to make the consistent and the proper execution of test goals. Test goals are extracted from the execution contracts by involving the alternatives. For each of the alternative a test goal has been identified.

Case study: The proposed approach is validated by using a case study of inventory system. It contains purchase requisition for an item required; the purchase order for the requisitioned item is then created after that the product receipt for the purchased item is created. Which means the product is received. For the issuance of an item store requisition is required and the issued item will be stocked out from the system.

System specifications:

- Only authorized user can access the system
- The first step is to create a purchase requisition for the item indicating the item required
- Purchase order for an item can be made only for the completed purchase requisition
- Purchase order can be put to registered vendors against the requisition
- The item having purchase order must be stocked in the system
- A store requisition for the issuance of item can only be made if the item is in stock
- A stock out can only be made for an item against the store requisition

Overall system use case diagram: The overall system use case diagram is shown in Fig. 2. It represents the entire system use case where the actors that are interacted to the system are defined; similarly all the possible use cases must be identified and expressed at his level with participating actors.

Once the entire use case of the system is identified it is then converted into sequential use case, which represents the sequential flow of the entire system. For each use case node of the entire system we will generate use case scenarios by adding parameters and contracts. The contracts of the use case scenarios fulfill the passing criteria to reach the next point in the use case scenario.

Generation of sequential use case diagram: The overall sequence diagram corresponding to system use case diagram is shown in Fig. 3. It shows the execution flow of

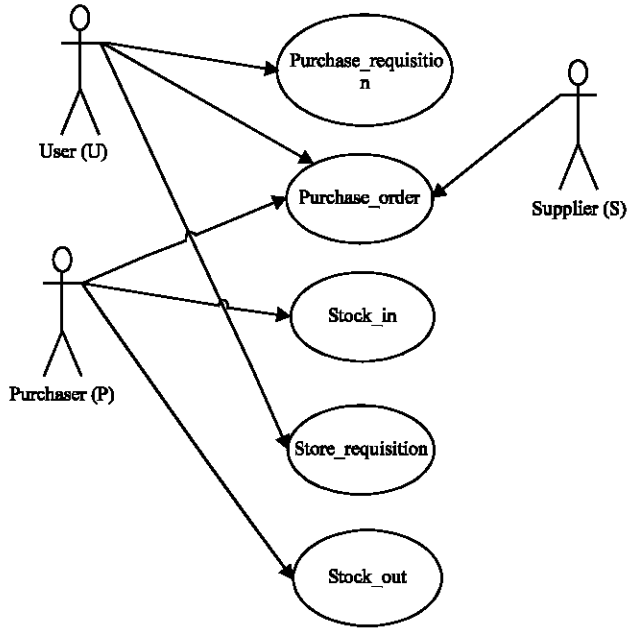

Fig. 2: Overall system use case diagram

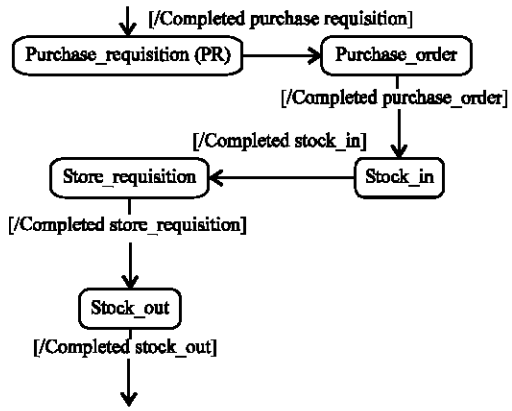

Fig. 3: System sequential use case diagram

the whole life cycle of the system with Pre and Post Condition of each use case representing a use case node, where the post condition of the previous node will become the pre condition for the next node in the sequence. The Pre and Post conditions for the use cases are derived from the specification of the system and appeared as guard to the sequential use case diagram.

The sequential use case diagram can be presented as activity diagram that shows the activity path of the system. The addition of contracts at the sequential use case diagrams enables to extract sequential constraints that can be recorded as logical expression that can be tested for the validation of sequential execution of the system.

Extracting sequential constraints for use cases: The sequential contracts for the entire system use case is derived by following the path in the transition as logical expression by using the "AND/OR" logical operators. Where OR indicates optional path of the system flow:

- [/Completed Purchase_Requisition and/Completed Purchase_Order and/Completed Stock_In and/Completed Store_Requisition and/Completed Stock_Out]

For extraction of sequential contracts each of the use case nodes i.e. used in Fig. 2 has to be involved in path execution of the whole system.

Generation of use case scenario diagram: A use case scenario is a system usage view of a specific actor which can be a user, external system or communicating device (the basic course plus any appropriate alternate paths). Use cases scenario normally focus on the behavior of the system and typically describe several paths for a use case and simulate the sequence of actions to real happenings as expected to occur when the system is in operation (Liang *et al*., 2006).

For each of the use case there will be a scenario indicating the ordering of events. As there are multiple use cases in the system so for each use case there will be a separate scenario diagram. We are only dealing with the use case scenario Purchase Requisition (PR) as shown in Fig. 4.

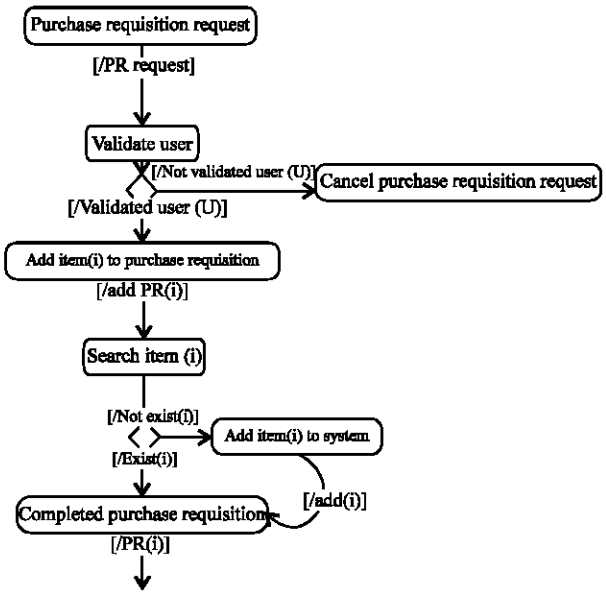

Fig. 4: Use case scenario for purchase requisition

Generating execution contracts: Contracts are generated by traversing the use case scenario which will be used to define the test goals. Contracts that are applied to the use case scenarios will be appeared as a user_action/system_response when converted to the sequence diagram. The primary advantage of using contracts is the definition of test goals but these should be consistent while moving from one stage to other in order to make the proper execution of test goals.

- **Pre condition:** User(u)
- **Execution contracts:** [/PR_Request and {(/Validated User (U) and /add_PR(i) and (exist(i) or (Not /Exist(i) and /add(i))) and /PR(i)) or /Not Validated User(U)}]
- **Post condition:** PR(i)

Test goal extraction: Test goals are extracted from the execution contracts defining the path flow for the scenario. Each test goal defines the alternative path of the scenario:

- **Test goal TG_PR1:** TG_PR1= [/PR_Request and /Validated User (U) and /add_PR(i) and exist(i) and /PR(i)]
- **Test Goal TG_PR2:** TG_PR2= [/PR_Request and /Validated User (U) and /add_PR(i) and Not /Exist(i) and /add(i)) and /PR(i)]
- **Test goal TG_PR3:** TG_PR3= [/PR_Request and //Not Validated User (U)]

RESULTS AND DISCUSSION

We are generating results based on the related techniques that presents use case based system testing. (Briand and Labiche, 2002) work provides a base for system testing based on use cases. (Nebut *et al*., 2003) has extends Briand's work by adding contracts. However, both the approaches have lack of some formalization technique for test case generation and to maintain consistency between use cases to scenario. The main advantages of the proposed approach are following:

- Addition of contracts to the use cases as pre and post condition enables to formally express sequential flow as logical expression. AND/OR logical operators can be used to identify execution paths. The advantage of proposed approach is that it allows the addition of contracts to use cases which added more strength to testing by aiding to generate complete test conditions and enabling to derive conditional test case generation and also sequential flow can be tested by contracts easily
- A use case scenario presents the execution trace of a system and provides a base for the development of state machine. Use case scenarios can be expressed by using the sequence diagram that shows the flow of events but it is difficult to define contracts at the sequence diagram. However, through pre and post conditions applied to use case scenario enables the generation of test paths. The proposed

approach applies the contractual sequence diagram derived from the use case scenario that can be used to bridge the gap between the test objectives and test cases alternatively depicting the use case scenario as it may contain additional information than scenario

- The advantage of applying contracts at the scenario enables to generate the test cases also referred to as test goals. These test goals capture the flow of events for the use case scenario. As the test goals are based on contracts so that can be formalized as logical expression

CONCLUSION

In this study, we have presented a approach based on use cases, as use cases are good source for generating test requirements at the analysis level, with the addition of contracts at the use case sequential flow allows to track the path selection at the top node, with the introduction of contracts to each use case enables to strength the conditional execution flow of use cases where as the post condition of the last use case becomes the pre condition of the next use case in order. Each use case consist of a set of scenarios to execute under which the use case may run usually there is one nominal scenario and a number of exceptional scenarios. With the introduction of contracts at the scenario enables to make a conditional testing likewise generation of conditional test path selection becomes easy. The sequence diagram generation from use case scenario is more appropriate if contracts are available at the use case scenarios, where the contracts of the use case scenario becomes messages for the sequence diagram, hence enhances the power of testing at the analysis level.

REFERENCES

Alspaugh, T.A., D.J. Richardson, T.A. Standish and H. Ziv, 2005. Scenario-driven specification-based testing against goals and requirements. Proceedings of the 11th International Workshop on Requirements Engineering: Foundation for Software Quality, June 12-13, 2005, Porto, Portugal, pp: 187-202.

Blackburn, M., R. Busser and A. Nauman, 2004. Why model-based test automation is different and what you should know to get started. Proceedings of the International Conference on Practical Software Quality and Testing, October 25-29, 2004, Minneapolis, MN., USA., pp: 212-232.

Briand, L. and Y. Labiche, 2002. A UML-based approach to system testing. Software Syst. Modeling, 1: 10-42.

El-Far, I.K. and J.A. Whittaker, 2001. Model based Software Testing. In: Encyclopedia Software Engineering, Marciniak, J.J. (Ed.). Vol. 1. Wiley-Interscience, USA., ISBN-13: 978-0471210085, pp: 825-837.

Hsia, P., J. Samuel, J. Gao, D. Kung, Y. Toyoshima and C. Chen, 1994. Formal approach to scenario analysis. IEEE Software, 11: 33-41.

Kim, Y.G., H.S. Hong, D.H. Bae and S.D. Cha, 1999. Test cases generation from UML state diagrams. IEE Proce. Software, 146: 187-192.

Kosters, G., B.U. Pagel and M. Winter, 1997. Coupling use cases and class models. Proceedings of the BCS FACS/EROS Workshop on Making Object-Oriented Methods more Rigorous, June 24, 1997, Imperial College, London, UK.

Liang, H., J. Dingel and Z. Diskin, 2006. A comparative survey of scenario-based to state-based model synthesis approaches. Proceedings of the International Workshop on Scenarios and State Machines: Models, Algorithms and Tools, May 20-28, 2006, Shanghai, China, pp: 5-12.

Nebut, C., F. Fleurey, Y. Le Traon and J.M. Jezequel, 2003. Requirements by contracts allow automated system testing. Proceedings of the 14th International Symposium on Software Reliability Engineering, November 17-20, 2003, Denver, CO., USA., pp: 85-96.

Raza, N., A. Nadeem and M.Z.Z. Iqbal, 2007. An automated approach to system testing based on scenarios and operations contracts. Proceedings of the 7th International Conference on Quality Software, October 11-12, 2007, Portland, OR., USA., pp: 256-261.

Ryser, J. and M. Glinz, 1999. A scenario-based approach to validating and testing software systems using statecharts. Proceedings of the 12th International Conference on Software and Systems Engineering and their Applications, December 8-10, 1999, Paris, France.

Some, S.S., 2007. Specifying use case sequencing constraints using description elements. Proceedings of the 6th International Workshop on Scenarios and State Machines, May 20-26, 2007, Minneapolis, MN., USA., pp: 4.

Whittle, J. and P.K. Jayaraman, 2006. Generating hierarchical state machines from use case charts. Proceedings of the 14th IEEE International Conference on Requirements Engineering, September 11-15, 2006, St. Paul, MN., USA., pp: 19-28.