Prioritization of Re-executable Test Cases of Activity Diagram in Regression Testing Using Model Based Environment

Prof. Sharmila M. Shinde (*Associate Professor*)
dept. of computer engineering
JSPM's Jaywantrao Sawant College of Engineering, Pune
Pune, India
sharmi_anant@yahoo.co.uk

Miss. Komal S. Jadhav *(PG Student)*
dept. of computer engineering
JSPM's Jaywantrao Sawant College of Engineering, Pune
Pune, India
jadhavkomal.19@gmail.,com

Miss. Hetal Thanki *(Assistant Professor)*
dept. of computer engineering
JSPM's Jaywantrao Sawant College of Engineering, Pune
Pune, India
hetal_thanki@yahoo.co.in

***Abstract*—** As we all know, software testing is of vital importance in software development life cycle (SDLC) to validate the new versions of the software and detection of faults. Regression Testing, however concentrates on generating test cases on changed part of the software to detect faults more earlier than any other testing practices. In case of model based testing approach, testing is performed using top-down method (black box method) and design models of the software, for example, UML diagrams. UML diagrams gives us requirement level representation of the software in graphical format which is now a days a standard used in software engineering.

In our proposed approach, we have derived a new technique which has never been used before to prioritize the test cases in model based environment. In this technique, we have used activity diagram as an input to the system. Activity diagram is used basically because it gives us the complete flow of each and every activity involved in the system and represents its complete working. Activity diagram is further changed as the requirement changes, each time, when the changes happen, they are recorded and test cases are generated for the changed diagram, test cases are also generated for the original diagram. Test cases for both the diagrams are compared and classified as re-usable and re-executable test cases. Re-usable test cases are those that remain unchanged during requirement changes and re-executable test cases belong to the changed part of the diagram. Then re-executable test cases are prioritized using one heuristic algorithm based on ACT(Activity Connector) table. Now, the question is why to prioritize only the re-executable test cases. Because, any how we have to execute re-usable test cases, as they remain same for both the versions of the diagram and are already tested when original diagram was made. But, re-executable test cases are never been tested and may detect faults in the modified design quickly and by prioritizing them we can also reduce the execution time of the test cases which will give us effective testing performance and will evolve a better new version of the software. All the existing prioritization techniques are either code based or are using various tool supports. Code based techniques are too complex and tedious because for a small change in code, we need to test whole application repeatedly. And in case of tool support, we have multiple assumptions and constraints to be followed. This proposed technique will surely give better results and as the type of technique has never been used before will also prove very effective.

Keywords- *activity diagram; regression testing; test prioritization; reusable test cases; re-executable test cases; ACT table*

*****

I. INTRODUCTION

With the advent of technology software becomes very crucial part in all the institute and industries. To develop particular software and to test it as per customer requirement is very important because the software which is not able to satisfy customer requirement after development will lead to increase and waste of cost, time and effort of all parts of organization. Model-based test case generation is gaining acceptance to the software practitioners. Advantages of this are the early detection of faults, reducing software development time etc. UML (Unified Modeling Language) is used to develop software designs, which includes various diagrams for complete representation of the software. Few work on the test case generation using activity diagrams is reported in literatures. To increase the productivity better test case suit generation is very important for any small or large software application. To generate better test cases of any project which reduce number of test cases to be executed and also all part coverage is necessary now days. Regression testing is used for generating the test cases, and it allows to reduce the test cases in case of modifications in the system. So, the idea to generate and prioritize optimum number of test case suite for better utilization of resources motivates to develop this system.

Model-driven software development is a new software development paradigm. Its advantages are the increased productivity with support for visualizing domains like business domain, problem domain, solution domain and generation of implementation artifacts. Activity diagram is an important diagram used for business modeling, control and object modeling, complex operation modeling etc. Main advantage of this model is its simplicity and ease of understanding the flow of logic of the system.

Regression testing allows to optimize and reduce test cases, so that there are minimum number of test cases to be generated after some changes in the design of software during software development cycle.

In our work, test cases are generated for different projects using XML format of the Activity Diagram and then the test cases for changed part are prioritized using prioritization algorithm with the help of ACT(Activity Connector Table) Table. The details of the ACT Table are given in the proposed system section. This proposed technique will prove better to minimize test cases after making changes in the model so as to validate new version with minimum efforts and evolve better version of the software.

II. LITERATURE SURVEY

In [3], test cases are prioritized using sequence diagram and state chart diagram. The design model from the requirement defines the state variables and rules written in the application and the states are identified by state machine which is written in cord scripts for exploring the models. The states are compared and prioritized based on severity. Test case generator generates and clusters test cases using agglomerative hierarchical process(AHP) which is cost effective method using dendragram. The model based approach here is a pre-implementation testing process starts at the design phase. Changes in the requirements have a quicker effect on the models rather than changes in coding phases. Agglomerative clustering approach is used to group similar test cases based on severity factor provide an efficient method to group similar test cases.

In [4], a new test case prioritization method is implemented basically for component based softwares. This method prioritizes test cases in descending order for Component Based Software Development(CBSD) using the concept of Prim’s algorithm. This algorithm makes use of CIG(Component Interaction Graph) as input for a medium/large size CBSD(Component Based Software Development Process) by taking any real-time system as an example and to generate prioritized test cases in descending order. The algorithm used finds the defects in component based software in less time. Here more importance is given to component interactions because maximum defect occur when components are going to interact with each other. This approach is mainly applicable to test the component composition in case of component based software maintenance.

In [5], the weighting based test case prioritization algorithm is used. The weighting factors used basically are code based. The values for factors used are approximate. The factors are customer-allotted priority(CP), developer-observed code implementation complexity(IC), changes in requirements(RC), fault impact of requirements(FI), completeness(CT) and traceability(TR). The algorithm is based on analysis of the percentage of test cases performed to find the faults and on Average Percentage of Fault Detected (APFD) metric’s results. Abiding by the percentage of executing test cases in earlier fault detection is important as sometimes regression testing ends without executing all test instances.

This project[6] uses the Extended Finite State Machine(EFSM) model and the analysis of dynamic dependencies namely data dependence and control dependence along with their interaction patterns. The proposed technique named dynamic interaction-based prioritization modifies the existing approach in order to improve the early fault detection capability. Other criterion for optimization is to reduce the resource cost. The proposed dynamic interaction-based prioritization technique performs better than the existing randomized prioritization of test cases from system models. It can be observed that the DIP(Dynamic Interaction-based Prioritization) algorithm is able to detect the maximum number of faults by almost 83% of the test case execution for the system models considered.

In [7], requirement based system level test case prioritization scheme is developed and validated to reveal more severe faults at an earlier stage and to improve customer-perceived software quality using Genetic Algorithm(GA). For this, a set of prioritization factors is proposed. The factors may be concrete, such as test case length, code coverage, data flow and fault proneness, or, abstract, such as perceived code complexity and severity of faults, which prioritizes the system test cases based on six factors: customer priority, changes in requirement, implementation complexity, completeness, traceability and fault impact. The goodness of these orderings was measured using an evaluation metric called APFD and PTR that will also be calculated.

In [1], Roberto S. Silva Filho, Christof J. Budnik,William M. Hasling, Monica McKenna and Rajesh Subramanyam have proposed a model based regression testing and prioritization scheme which efficiently selects test cases for regression testing based on different user defined concerns. It depends on traceability links between models, test cases and code and user defined properties associated to model elements. Here, an automatic tool called TDE/UML is used which generates test cases using UML diagrams and categorizes them as reusable and re-executable test cases and prioritizes them . The proposed approach in this paper is top-down approach as compared to traditional code based bottom-up approaches, because it works along with the software development life cycle i.e. from starting stage of the software. This approach works efficiently, as it detects the errors in the early development stages of the software. The technique reduces the efforts needed to validate the new versions of the software and improves the overall productivity of the software.

Model-based regression testing ensures the reliability of the evolving softwares by optimally selecting the test cases to test the affected portion of the software. This technique promises the reduction in labor, time and cost to test the new version of the software.

In[2], Mr. Rohit N. Devikar had presented the automatic tool, Model-based regression testing tool(MBRT) ,which is java based tool and used to reduce, generate and also categorize test cases as obsolete, reusable and re-testable test cases. In this paper class diagram and state machine diagram are used for regression testing and flow graph is used to generate the test cases.

In [8], to optimize the priority of the test cases at different points in the design cycle, tool called Echelon is developed by A. Srivastava and J. Thiagarajan, which is a test prioritization system, that prioritizes the set of test cases of any application, based on changes made to the program.

Echelon uses a binary matching system that computes the differences at a basic block granularity between two versions of the program in binary form. Echelon works on heuristic approach. Echelon runs under the Windows environment.

In [9], R. France and B. Rumpe have given an overview of current research in Model Driven Engineering (MDE) . The research work in this paper is focused on providing technologies that address the recurring problem of bridging the problem-implementation gap. We also encourage research on the use of runtime models. In this paper, a vision of MDE environment is presented, that if realized, can result in improvement in software development productivity and quality. Progressively closer approximations of the vision will have increasingly significant effects on the effort required to develop complex software. The vision can act as a point of reference against which MDE research progress can be informally assessed.

In [10], a methodology and tool is presented by L. C. Briand, Y. Labiche, and S. He that support test selection from regression test suites based on change analysis in object-oriented designs. Regression test cases are categorized as Reusable, Re-testable, and Obsolete.

This paper focuses on automating the regression test selection based on design model represented by Unified Modeling Language (UML) and the traceability linking the design to test cases. UML is used in this case as it is becoming industry de-facto standard.

In[11], to remove disadvantages of code based regression test selection, a new specification based test selection technique is developed by Y. Chen, R. L. Probert, and D. P. Sims, which is based on customer requirements. The basic model used is Activity diagram which is part of UML(Unified Modeling Language). In this paper two types of regression tests are selected as targeted tests and safety tests. Targeted tests ensure that important current customer features are still supported in the new release. Safety Tests are risk-directed, and ensure that potential problem areas are properly handled. Proposed test selection technique is based on a practical risk analysis model.

III. PROPOSED SYSTEM

Various Techniques have been proposed to prioritize the test cases in model driven environment. In the proposed approach, Activity Diagram will be used to generate the test cases for the system and prioritize them. Heuristic based approach will be used to prioritize the test cases.

Proposed system is divided in three steps as:

A. UML to XML Conversion

B. Test case generation and classification as re-usable and re-executable test cases

C. Prioritization of re-executable test cases

A. UML to XML Conversion

In this step, an original activity diagram for the system is taken as an input then this diagram is backed up in another file and original file is used for modification to the model (diagram). Both these files are in .edg format which is a file format used to store uml diagrams in UML Diagrammer tool. Both the files then are converted to the xml format which represent them in the form of tags and saved to .xml files.

B. Test case generation and classification as re-usable and re-executable test cases

After uml to xml conversion, both the .xml files are used for the generation of test cases. Then test cases generated for both the diagrams are compared and common test cases are classified as re-usable test cases and uncommon test cases are classified as re-executable test cases.

C. Prioritization of re-executable test cases

Re-executable test cases are prioritized in order to execute test cases with minimum execution time and to detect faults earlier than normal process. This prioritization will evolve new better version of software with minimum efforts. In this algorithm, prioritization will be done on the basis of Activity Connector Table(ACT). A heuristic technique for prioritization of test cases derived from the activity diagram using Activity Connector Table (ACT). The heuristic prioritization algorithm will identify the most prioritize path for a test case.

We first extract the necessary information from the diagram. Based on the extracted information, an Activity Connector Table(ACT) is generated. With the help of ACT test cases are generated, and by applying the heuristic prioritization algorithm, prioritization sequence of test case is identified.

Pacestar UML Diagrammer is used to generate the activity diagram and its design structure saved in text format is used for conversion of this diagram to xml file. Prioritization of re-executable test cases will reduce the test cases and will reduce the testing time, efforts and cost of the software and will improve the overall productivity of the software by reducing the efforts needed to validate the new versions of the software.

IV. IMPLEMENTATION DETAILS

A. Mathematical Model

For given activity diagram X and set of test requirements r1, r2, ..., rn there exist Xi such that Xi satisfies all Ri.

Let S={s,e,x,y,f}be the programmer's perspective of prioritization of test cases.

S → System which represents programmer's perspective.
s → Distinct start of the system
e → Distinct end of the system
x → Activity diagram (old and modified)
Y → Set of prioritized test cases.
F → Central function for UML to XML Conversion, Test Case Generation and Classification, Prioritization of Re-executable Test Cases
Φ → Constraints

Let X be the input such that X = X1, X2, ..., Xn where X1, X2,....., Xn are different activity diagram of same domain projects.

M=m1, m2,...., mn where m1, m2, ..., mn are modified activity diagram for X.

F(X,M)= Activity Diagram| Φ=must enter old and new activity diagram

Y = prioritized test cases.

UML to XML Conversion:
INPUT: **X**
OUTPUT: Y1=M, Modified Activity Diagram
F(X):X| Φ =Original Activity Diagram in ".edg" Format

Test Case Generation and Classification:
INPUT: Y1(X, M)
OUTPUT: Y2=(RU, RE)
RU-Reusable Test Cases
RE-Re-executable Test Cases
F(X',M'):X',M'| Φ=original and modified activity diagram in .xml format

Prioritization of Re-executable Test Cases:
INPUT: Y2(RE)
OUTPUT: Y=Prioritized Test Cases
F(RE):RE| Φ=By using different heuristics

B. Block Diagram

Figure 1. shows the block diagram of the proposed system. The description of this diagram is already given in section III(A).

C. Algorithm for System

- *Take original activity diagram as an input, which is a file with .edg extension.*
- *Backup this file (to be used later).*
- *Make changes to the original file taken as an input.*

- *Now map the backup file and the modified one to the xml format.*
- *Generate the test cases for these two files.*
- *Categorize the test cases as re-usable and re-executable.*
- *Prioritize the re-executable test cases using the prioritization algorithm.*
- *Prioritized test cases obtained from above step will be the output of the proposed system.*

D. *Prioritization Algorithm*

- *Input Modified Activity diagram.*
- *Generate Activity Connector Table (ACT) using Activity diagram.*
- *Loop the steps for all the test case states possible using ACT.*
    - *Create a test case state*
    - *By using the ACT table get the next state.*
    - *If the next state is the decision state then,*
        - *Store the entire path into the next empty location*
        - *Continue the true side until it reaches the end state*
        - *For the false side just add the end state and continue*
    - *If the next state is the end state*
        - *Stop the process*
        - *Check any other path is incomplete, if yes then continue with there.*
        - *Else exit*
    - *If the next state is the ordinary state, then add the state at the end of the current path*
- *For assigning weigthage, use ACT Table*
    - *Start the process with the start state, assign the value as 1.*
    - *Using ACT Table, check the next state*
    - *Increment the value and assign it to next*
    - *If the current state is decision then,*
        - *Increment the value and assign it to both true and false side of the decision*
    - *Repeat the step until it reaches the end state*
- *Compute fitness value as f,*
    - *For each node calculate the number of incoming nodes as a and number of outgoing nodes as b.*
    - *Compute f=a*b.*
- *For prioritization, create three variables called p_in, p_out and p_best.*
- *Calculate the initial test path value, store it in the p_best and store the path in p_in*
- *Calculate the neighborhood test path value.*
- *Compare the current value with the p_best.*
    - *If p_best is greater than the current value*
        - *Store the current path in p_out.*
        - *Then go to next main step*
    - *If p_best is smaller than the current value*
        - *Store the current value in the p_best.*
        - *Add the current path to the p_in*
- *Repeat step before the previous step until all path have been covered.*
- *By using the p_in we can get the prioritized path.*

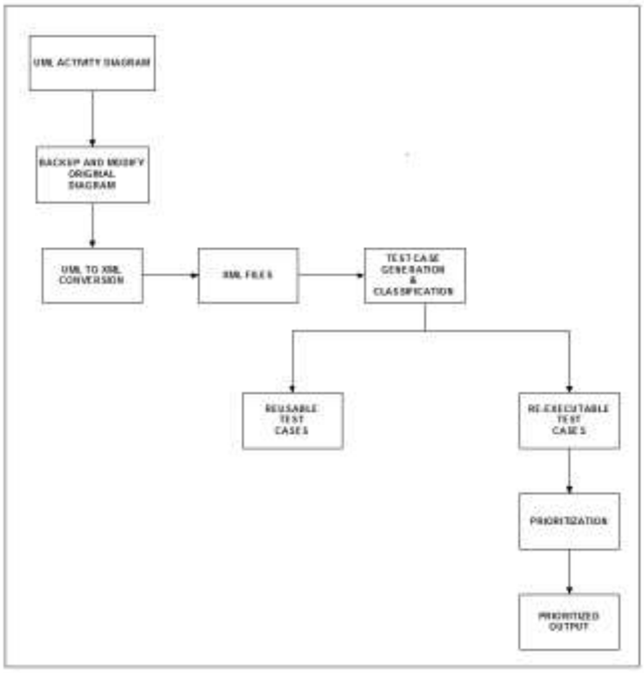


Figure 1. Block Diagram

E. *Activity Connector Table (ACT)*

Activity Connector Table (ACT) is the paradigm used to prioritize the executable test cases. It contains the decision and action states and simply its next state in the activity diagram. It can be generated automatically or manually. But if we try to generate it automatically, it gives a high error rate. As we are generating the activity diagram in Pacestar UML Diagrammer, it is saved in some textual format from which we are getting information to generate test cases and we can also get the information about the connectors. Many times it may happen that the diagram is not drawn 100% accurate by means of connecting the connectors and other components or we may delete and add connectors many times. Because of this, the textual format we obtain is not 100% accurate and we cannot get the information about the connectors 100% accurate. So ACT Table is generated manually which will give us 100% correct results. The example of the ACT Table is given in the next results and discussion section.

V. RESULTS AND DISCUSSION

A. *Case Study*

With the use of Pacestar UML Diagrammer, we are drawing activity diagram, then using the same tool we are modifying it. Fig.2 and Fig.3 shows the original and modified diagram. These diagrams belong to the Online Order System. In the modified part, as we can see, the payment module is added extra. The ACT table for these diagrams can be prepared as shown in the Fig.4.

In this case, while modifying the diagram, original part is maintained as it is, so there will not be any obsolete test cases, i.e. test cases that become of no use during testing as components belong to those test cases is been removed from the original diagram permanently. But in our work, we are considering only re-usable and re-executable test cases.

Once, the original diagram is saved in backup and modified, they are converted to xml format to generate the test cases. After generating the test cases, they are classified as re-usable and re-executable test cases. Using ACT table, prioritization algorithm is applied on re-executable test cases and they are prioritized. Fig.5 gives the output for the diagram in case study.

From the result window, it can be clearly seen that the test cases for modified part are classified as re-executable test cases and the prioritization output for the same is shown in the same window.

This system gives us 100% accurate results as we are making the ACT table manually. In case of automatic generation of ACT Table, we could have some error rate.

In prioritization algorithm, the heuristic factor used is ACT table and the weighting is done on the basis of number of in and out edges to the action or decision component.

VI. CONCLUSION AND FUTURE SCOPE

In software development life cycle, software testing is very important part, as the success of the software project depends on it.

It is therefore very essential to handle this task of software development very carefully. In our paper, we present a regression testing and prioritization technique which will work from early stages of the software development life cycle and will reduce time, efforts and cost for testing. The type of technique is never been used before for the prioritization. This is very simple and effective technique with 100% accuracy, giving better results. Heuristic approach is used to prioritize the re-executable test cases which makes use of ACT table. This prioritization technique is giving better improved version of the software with minimum efforts and increases software productivity. This technique can also be applied to various domains as education, medical, finance, etc.

This is very effective technique, as it is very simple and easy to understand. This technique gives better results using top-down i.e. black box testing approach. At the design level only, we can have testing applied, which detects faults earlier before coding. Complexities in code based technique can be totally removed using this technique, which is the main advantage of this.

In future, we can try to make the ACT table automatically, by using another better tools for drawing the activity diagram which can specify each and every component and connector 100% correctly.

ACKNOWLEDGMENT

The satisfaction that accompanies the successful completion of any task would be incomplete without mentioning the people who make it possible. I wish to thank honourable guide Prof. S. M. Shinde, H.O.D. Prof. H. A. Hingoliwala, PG coordinator Prof. M.D. Ingle and Principal Dr. M.G. Jadhav of JSPMs JAYAWANTRAO SAWANT COLLEGE OF ENGG. Pune, for providing vital comments, information, and review of this paper. We also acknowledge thanking different sites and references which we receive from different sources.

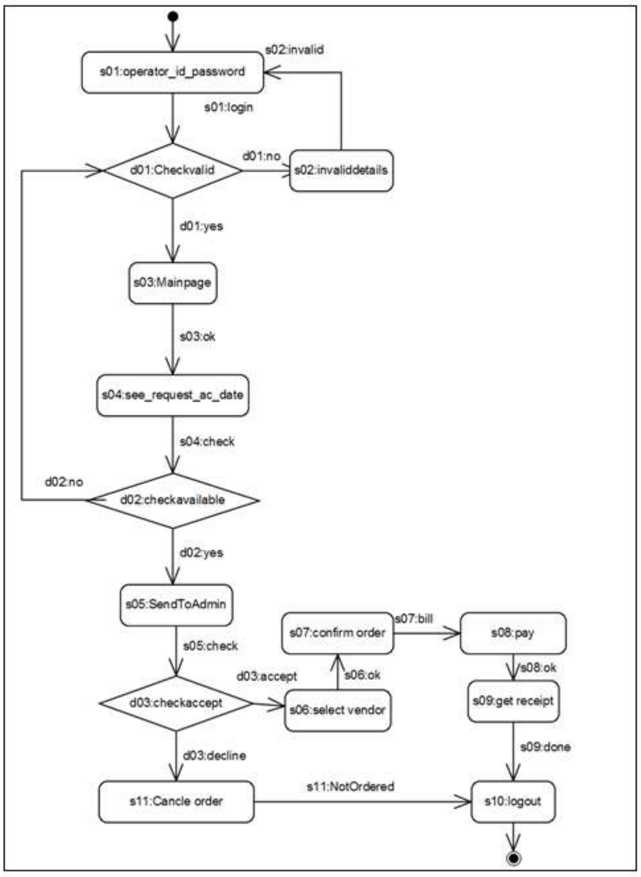


Figure 2. Original Activity Diagram

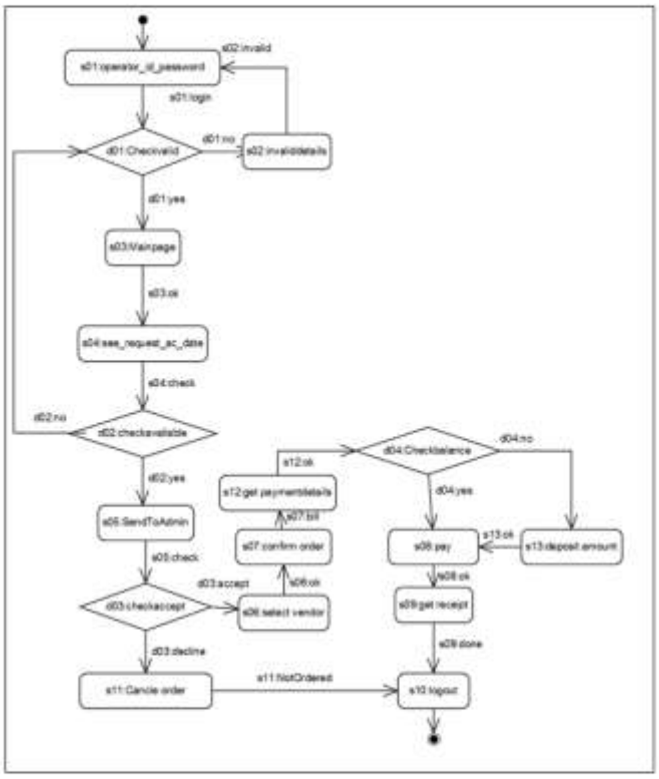
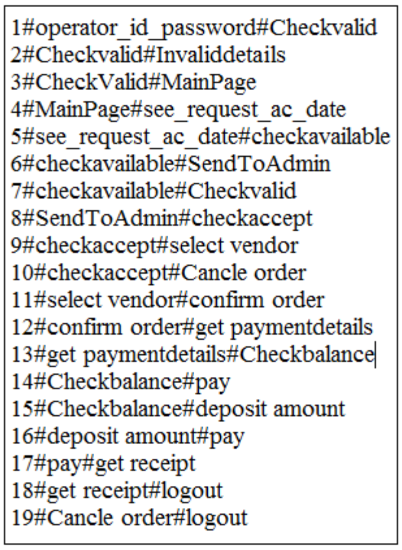
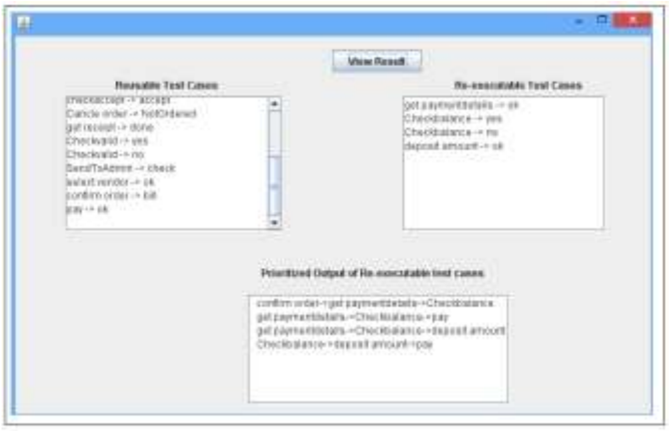


Figure 3. Modified Activity Diagram



```
1#operator_id_password#Checkvalid
2#Checkvalid#Invaliddetails
3#CheckValid#MainPage
4#MainPage#see_request_ac_date
5#see_request_ac_date#checkavailable
6#checkavailable#SendToAdmin
7#checkavailable#Checkvalid
8#SendToAdmin#checkaccept
9#checkaccept#select vendor
10#checkaccept#Cancle order
11#select vendor#confirm order
12#confirm order#get paymentdetails
13#get paymentdetails#Checkbalance
14#Checkbalance#pay
15#Checkbalance#deposit amount
16#deposit amount#pay
17#pay#get receipt
18#get receipt#logout
19#Cancle order#logout
```

Figure 4. ACT Table



```
checkaccept -> accept
Cancle order -> NotOrdered
get receipt -> done
Checkvalid -> yes
Checkvalid -> no
SendToAdmin -> check
select vendor -> ok
confirm order -> bill
pay -> ok
```

```
get paymentdetails -> ok
Checkbalance -> yes
Checkbalance -> no
deposit amount -> ok
```

```
confirm order->get paymentdetails->Checkbalance
get paymentdetails->Checkbalance->pay
get paymentdetails->Checkbalance->deposit amount
Checkbalance->deposit amount->pay
```

Figure 5. Result

REFERENCES

[1] Roberto S. Silva Filho, Christof J. Budnik,William M. Hasling, Monica McKenna, Rajesh Subramanyam, “Supporting Concern-Based Regression Testing and Prioritization in a Model-Driven Environment”, 34th Annual IEEE Computer Software and Applications Conference Workshops,2010.

[2] Mr. Rohit N. Devikar, Prof. Manjushree D. Laddha, “Automation of Model-based Regression Testing”, Internatonal Journal of Scientific and Research Publications, Volume 2, Issue 12, December 2012.

[3] Vinothkumar.N, Galeebathullah.B,”Prioritizing Test Cases for Regression Testing A Model Based Approach ” International Journal For Trends in Engineering & Technology, Volume4, Issue1, April 2015.

[4] Shweta A. Joshi, Prof. D. S. Adiga, Prof. B. S. Tiple, “Effective Use of Prim’s Algorithm for Model Based Test Case Prioritization” International Journal of Computer Science and Information Technologies, Vol 5(3), 2014, 3444-3447.

[5] Thillaikarasi Muthusamy, Dr. Seetharaman.K,”Effectiveness of Test Case Prioritization Techniques Based on Regression Testing”, International Journal of Software Engineering & Applications, Vol 5, No. 6, November 2014.

[6] Chris Nitin Adonis Petrus, M.S. Razou, M. Rajeev, M. Karthigesan, “Model-Based Test Case Minimization and Prioritization for Improved Early Fault Detection Capability”, International Journal of Innovative Technology and Exploring Engineering, Volume-2, Issue-5, April 2013.

[7] S. Raju, G. V. Uma, “Factors Oriented Test Case Prioritization Technique in Regression Testing Using Genetic Algorithm”, European Journal of Scientific Research, Vol. 74, No. 3, 2012.

[8] A. Srivastava and J. Thiagarajan, "Effectively Prioritizing Tests in Development Environment," in Intl. Symposium on Software Testing and Analysis Roma, Italy: 2002.

[9] R. France and B. Rumpe, "Model-driven Development of Complex Software: A Research Roadmap," in Future of Software Engineering: IEEE Computer Society, 2007.

[10] L. C. Briand, Y. Labiche, and S. He, "Automating regression test selection based on UML designs," Inf. Softw. Technol., vol. 51, pp. 16-30, 2009

[11] Y. Chen, R. L. Probert, and D. P. Sims, "Specification-based Regression Test Selection with Risk Analysis," in 2002 Conference of the Centre for Advanced Studies on Collaborative Research Toronto, Ontario, Canada: IBM Press, 2002.

[12] O. Pilskalns, G. Uyan, and A. Andrews, "Regression Testing UML Designs," in 22nd IEEE International Conference on Software Maintenance: IEEE Computer Society, 2006.

[13] T. J. Ostrand and M. J. Balcer, "The Category-partition Method for Specifying and Generating Fuctional Tests," Commun. ACM, vol. 31, pp. 676-686, 1988.

[14] L. Naslavsky, H. Ziv, and D. J. Richardson, "A Model-based Regression Test Selection Technique," in IEEE International Conference on Software Maintenance, 2009, pp. 515-518.

[15] S. Elbaum, A. G. Malishevsky, and G. Rothermel, "Prioritizing test cases for regression testing," in ACM SIGSOFT International Symposium on Software testing and analysis Portland, Oregon, United States: ACM, 2000.

[16] G. Rothermel, S. Elbaum, A. G. Malishevsky, P. Kallakuri, and X. Qiu, "On Test Suite Composition and Cost-effective Regression Testing," ACM Trans. Software Engineering Methodology, vol. 13, pp. 277-331, 2004.

[17] P. K. Chittimalli and M. J. Harrold, "Regression test selection on system requirements," in 1st India Software Engineering Conference Hyderabad,India:ACM,2008.