ISSN 2229-5518

A Study on Dependency Optimization using Machine-Learning Approach for Test Case Prioritization

Sathya C, Karthika C

Abstract—The main goal of this paper is Test Case Prioritization where the process is to order test cases. This ordering of test case will give and increased rate in fault detection. Test Case Prioritization will improve the fault fixing process anmd thus leads a way to early delivery of the software. Due to the functional dependencies between the requirements the cse of executing the test case in any order goes false. In this paper, we present different techniques that provides us information about the various ways of prioritization the test case using the dependencies between them. The dependencies of the test case is main based on the interaction between the requirements or even between the various modules and funcitons of the whole system. This test case ordering based on the functional dependencies is likely to increase the fault detection earlier than other fault detection systems. This is known through the empirical evaluations on six systems that were built towards the industry. We also proposed a new system which is a machine learning techn ique. This is known through the empirical evaluations on six systems that were built towards the industry. We also proposed a new system which is a machine learning technique. Here Case-Based Paradigm is indulged with Analytical Hierarchy Processing which proves itself better than other techniques proposed to date.

Index Terms— Analytical Hierarchy Processing, Case-Based ranking, Dependency, Fault Detection, Prioritization, Test Automation, Test Case .

1 Introduction

Requirements prioritization plays a crucial role in software development, and in particular it allows for planning software releases, combining strategies a complex multicriteria decision making process. The indentification of requirement attributes in the second step is performed in a way to define uni variate ranking functions on the requirements set. For example, with reference to the goal of reducing development costs and the choice of "development cost" as a target ranking criterion, requirement attriputes such as the estimate number of "lines of code" or of "components" are suitable. The third step, namely the acquisition of attribute values over the set of requirements, usually represents the most expensive task in the prioritization process since it rests on the availability of expert knowledge or on the elicitation of evaluations from stakeholder. Since a target critetion might be encoded by manifold attributes and each attribute induces a ranking of the requirement set, the fourth step is concerned with the composition of the different attribute based rankings into a global ordering corresponding to the target criterion. This composition is usually defined in terms of a weighted aggregation schema. The assumption underlying the analysed approaches is that the ranking criteria, the requirement attributes, and the wat to compose them in case of multi criteria ranking can be defined independently of the nature of the current set of requirements prioritization probem which prevents exploiting available knowledge on the project's application domain. In contrast, an ex-post perespective will enable the exploitation of this knowledge through a prioritization process that is built on the actual set of requirements under evaluation and will lead to a different realization of steps 2 to 4. Namely, project stakeholders are asked to perform a pairwise comparision of the current requirements, allowing them to decide which requirementsm is to be given a higher rank between two alternatives without the need to identify a specific requirement attribute to encode the evaluation criterion adopted by the stakeholder.

So, for incstance. The users of an e-voting system may be asked to decide of the requirements"Graphical layout of the voting form" and "Getting audio feedback during the voting procedure" is more important. The difference between ex-ante and ex-post approaches can be summarized as follows. While in the ex-ante perspective the target ariterion is chosen in advance, in expost approaches project stakeholders are required to evaluate pairs of requirements along an underlying target criterion. Consequently, requirements ranking is not computed from the values of requirements attributes, but it is derived from the priority relations that are elicited directly from stakeholders, who may take into account implicit information that might not have been preliminarily encoded as requirement attributes. The composition of rankings in case of multi-stake holders prioritization is provided as instances of pairwise realtions and not as the result of the application of an analytical composition schema. An interesting advantage of eliciting input reagarding values rather than absolute values for attributes is that the noise on the input is recognized to be lower.

2 Objective Of The Study

The objective of the study is to propose a system which provides an order for the execution of test case and test suites based on their dependency structures. These dependency structure form directed acyclic graphs. The techniques for the prioritization is based on the Machine-Learning techniques which is based on the Analytical Hierarchy Process. This Analytical Hierarchy Process is further based on Case-Based Para-

digm. This proposed system should also help in increasing the detection of fault.

2.1 Principles of Software Testing

Presence of defects: Testing is the process of finding errors. But we have no proof to tell that the software being tested is fully of errors. What ever and how much testing is done on the software there may be still errors in the software which may or even may not be known to the people using it and also the developers.

Exhaustive testing is impossible: Testing the whole process of the software is very difficult. The number of test case is based on the requirments. If suppose there are 20 test suites and each have seven test cases then executing seven to the power twenty is very very tedious process, hence exhaustive testing cannot be done. We test only the important portions of the software.

Early Testing: Early testing deals with the process of testing the software from the beginning of the life cycle process, means the requirement ar also tested.

Defect clustering: Defects are all based on certain type of modules. There re many types of test cases during the repeating of the same modules. This will enable the testing process to find more errors than usually executing the same test cases without any modification.

Pesticide paradox: Pesticide Paradox testing is the process of creating new types of test cases during the reapeating of the same modules. This will enable the testing process to fine more errors than usually executing the same test cases without any modification.

Absence-of-errors fallacy: The Testing of a software is done to only a software which will be used by the user. Even after knowning that the software will not satisfy the customer, test the software by wasting time and errors should be avoided.

2.2 Test Case

Test cases involve the set of steps, conditions and inputs which can be used while performing the testing tasks. The main intent of this activity is to ensure whether the Software Passes or Fails in terms of its functionality and other aspects. There are many types of test cases like: functional, negative, error, logical test cases, physical test cases, UI test cases etc. Furthermore test cases are written to keep track of testing coverage of software. Generally, there is no formal template which is used during the test case writing. However, following are the main components which are always available and included in every test case: Test case ID, Product Module, Product Version, Revision history, Purpose, Assumptions, Preconditions, steps, Expected outcome and actual outcome.

- *Sathya C is currently pursuing masters degree program in Software Engineering in CIET, India, PH-7402107222. E-mail:c.sathyachandran92@gmail.com*
- *Karthika C name is currently working as Assistant Professor in Electronics and Communication Engineering at Dr. NGP Institute of Technology, India, PH-9524761426. E-mail:ck.ckarthika@mail.com*

2.3 Traceability Matrix

Traceability Matrix(also known as Requirement Traceability Matrix-RTM) is a table which is used to trace the requirements during the Software development life cycle. It can be used for forward tracing(i.e. Requirements to Design or coding) or backward(i.e. from Coding to Requirements). There are many user defined templates for RTM. Each requirement in the RTM document is linked with its associated test case, so that testing can be done as per the mentioned requirements. Further more, Bug ID is also included and linked with its associated requirements and test case. The main goals for this matrix are: To make sure Software is developed as per the mentioned requirements, To help in finding the root cause of any bug and to help in tracing the developed documents during different phases of SDLC.

2.4 Test Case Prioritizaiton

The priortizaiton of test case is the most important aspect in reducing the time needed for testing, effective use of resources and also early finding of faults or defects. The test case prioritization is the process of organizing the test cases in a order that test cases of higher priority are executed first. This priority is based on certain criteria based on the method of prioritizaiton.

2.5 Dependencies

Functional Dependency

Scenarios are defined as the sequence of interactions between two systems or more. The order in which these interactions are being processed is the order in which the dependencies are being found. Functional dependency is where some instructions should definitely be executed before the other instructions, just because the latter is dependent on the one which was executed before it.

Open and Closed Dependencies

A Closed Dependencies is one in which the dependable test case should be executed first and the dependent test case should be execute after it but not necessarily executed immediately after it. An Open Dependency is one in which the dependent test case should be immediately executed after the test case on which it is depended.

Dependent and Independent test cases

Dependent test cases are those who are dependent on each other which means they have interactions between each other. A independent test case in one which the test case in not dependent on any other test cases, hence therefore they do not have interactions with other modules in the system.

3 Techniques For Test Suite Prioritization

As referred in the paper[2] by D.Jeffrey and N.Gupta, the detection of errors in the software is done by occurring and reoccuring of software testing during the software development life cycle. The size of the test cases is dependeable on the size of the software. Due to various factors like time constraint and resource constraint we are prioritizing the test cases to know test case has the most importance to be surely tested. The number of test cases can be avoided by the number of requirements given by the customer. This paper produces an approach which is based on the output of the software. Here the output of the system is divided into various divisions, these divisons are called as slices. The number of requirements for the output slice determines the priority of the test cases. This approach has the ability high rate of fault detection.

The paper by D.Kundu, M.Sharma, D.Samantha and R.Mail, proposes a method which integrates both design, development and testing process in the software development life cycle. In the design phase, interaction diagrams are being developed from the use case matrix. These interactions diagram produces a list of sceanrios. From these scenarios the dependencies are being calculated. The module having the large number of sceanrios will be given higher priority and will be tested first. But this does not prove so good because the module with the large number of scenarios does not logically prove its importance. This approach is employed to improve the productivity of the testing process through scenarios prioritization.

Z. Li, M.Harman and R.Hierons proposed a method in their paper a search algorithm for regression test case prioritization. As discussed before due to insufficient resources for regression testing- regression testing is the process of executing the test cases repeatedly due to the change made in the module-prioritization of test cases is needed, which improves the effectiveness of regression testing. Older researches of these testing was done on greedy algorithms, but these algorithms produce sub-optimal results sicne the results gives only one minina. They used the algorithms like metaheuristic and evolutionary search algorithms to avoid the above problems. The results of this paper shows that genetic algorithm performs well for such purposes.

4 Importance of the Survey

Through evaluation systematic reviews and devlop understanding about systematic reviews we

- Investigate to what extent systematic review is beneficial as a priorization mechanism for the software engineering community.
- Investigate benefits form of sort of systematic reviews have been conducted in software engineering in general and in are of requirements prioritization in particular.
- Investigation benefits from the art requirements prirotization techniques and studies relevant to the different techniques.
- Conduct a systematic review of the requirements prioritization area to see what evidence regarding different prioritization techniques exist.
- Develop a research framework based on the systematic review to align research within requirements prioritization area and facilitate systematic reviews in future.

5 Requirements Prioritization

Requirements prirotization should also consider business issues and implementation issues. Business issues might involve financial benefits for the developing organization, market trends and focus, competitors, regulations whereas implementation issues mostly involve implementation cost, cost if not implemented, available resources etc. Another important aspect to be considered while prioritizing requirements is the customer perspective along with the perspectives of developers and financial personals. Customers provide vital inforamtion about the user/customer value; developers are better suited for the technical addition, all those perspectives can be involved and combined that adds value to the project and that have stake in the prject or product.

6 Prioritization Techniques using Case Based Ranking

Prioritization can be done with various different scales and types. Below, few of the prioritization techniques are presented. Some of the prioritization techniques assume that requirements have a priority associated with them while others group them in priority level.

6.1 Architecture Diagram

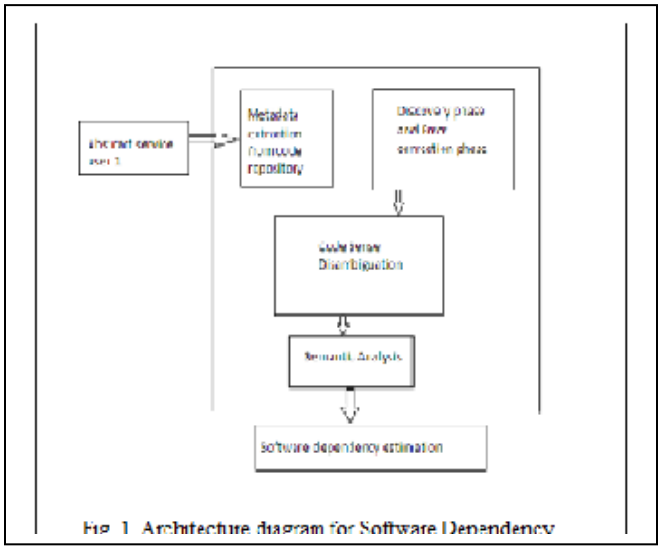

Fig. 1. Architecture diagram for Software Dependency

6.2 Analytical Hierarchy Processing

Analytical Hierarchy Process(AHP) is a systematic statistical technique based on relative assessment that has been used to prioritize software requirements in software community. The AHP

is a powerful and flexible decision making process to help people set priorities and make te best decision when both qualitative and quantative aspects of a decision need to be considered. By reducing with complex decisions to a series of one-on-one comparisions, AHP helps decisions makers arrive at the best decision. With AHP, one can synthesize the results, which provide a cleasr rational for choosing the candidate requirements. It is very complex in terms of sophistication and fine in terms of granularity. During the process, considering *n* requirements, *n*(n-1)/2* comparisons are to be made to each hierarchy level. This is often seen as a draw back in this process because with the increase number of requirements, the number of comparisons increases with a magnitude of O(n2). AHP can be used to prirotize requirements on the basis of different aspects and there have been number of studies which have reported the use of it in the industrial setting and real projects as an efficient and more difficult to use. In another study, AHP was reported more time consuming and difficult to use in certain situations considering aspects of cost and value. Therefore there is a need for more experimentation and industrial case studies to actually come toa final conclusion for its effectiveness under different situations.

4 Conclusion

The Functional Prioritization method follows the case-based paradigm for problem solving, according to which a solution to a hew problem can be derived form(partial) examples of previous solutions to similar problem. In the context of requirements prioritization, these examples are elicited form project stakeholders as pairwise preferences on samples of the set of requirements to be prioritized, and used to compute an approximated ranking for the whole set. The machine learning technique exploited by the method has been presented, both with the help of an untuitive example and be describing the Rank Boost algorithm, which is implemented in the method. The prioritization processs based on Functional Priorizaiton has been presented. A discussion of the method performance, which is defined in terms of tradeoffs between pregerence elicitation effort and ranking accuracy and of its domain adaptively, has been given, with the support of a set of different experimental measurements and of a case study. The experitmntal measures were taken by applying Functional Prioritizaiton to different prioritization problems, varying the number of requirements, the number of elicited pairs, and the accuracy of the computed ranking. Indicators for the statistical significance of the measurements have been provided. Finally, the Functional Prioritization method has been positioned with respect to state-of-the art approaches, with particular reference to the AHP method, which can also be considered an instance of the case-based problem solving paradigm. Differently from AHP, the Functional Prioritizaiton method enables a prioritization process, even over 100 requirements, thanks to the exploitations of machine learning techniques that induce requirements ranking approximations from the acquired data.

Acknowledgment

The authors wish to thank the institution for their great support.

References

[1] J. Bach, "Useful Features of a Test Automation System (Part iii)," Testing Techniques Newsletter, Oct. 1996.
[2] F. Basanieri, A. Bertolino, and E. Marchetti, "The Cow_Suite Approach to Planning and Deriving Test Suites in UML Projects," Proc. Fifth Int'l Conf. Unified Modeling Language, pp. 275-303, 2002.
[3] S. Elbaum, A. Malishevsky, and G. Rothermel, "Incorporating Varying Test Costs and Fault Severities into Test Case Prioritization," Proc. 23rd Int'l Conf. Software Eng., pp. 329-338, 2001.
[4] S. Elbaum, A.G. Malishevsky, and G. Rothermel, "Test Case Prioritization: A Family of Empirical Studies," IEEE Trans. Software Eng., vol. 28, no. 2, pp. 159-182, Feb. 2002.
[5] R.W. Floyd, "Algorithm 97: Shortest Path," Comm. ACM, vol. 5, no. 6, p. 345, June 1962.
[6] D. Jeffrey and N. Gupta, "Experiments with Test Case Prioritization Using Relevant Slices," J. Systems and Software, vol. 81, no. 2, pp. 196-221, 2008.
[7] B. Jiang, Z. Zhang, W. Chan, and T. Tse, "Adaptive Random Test Case Prioritization," Proc. IEEE/ACM Int'l Conf. Automated Software Eng., pp. 233-244, 2009.
[8] J. Kim and D. Bae, "An Approach to Feature Based Modelling by Dependency Alignment for the Maintenance of the Trustworthy System," Proc. 28th Ann. Int'l Computer Software and Applications Conf., pp. 416-423, 2004.
[9] R. Krishnamoorthi and S.A. Sahaaya Arul Mary, "Factor Oriented Requirement Coverage Based System Test Case Prioritization of New and Regression Test Cases," Information and Software Technology, vol. 51, no. 4, pp. 799-808, 2009.
[10] D. Kundu, M. Sarma, D. Samanta, and R. Mall, "System Testing for Object-Oriented Systems with Test Case Prioritization," Software Testing, Verification, and Reliability, vol. 19, no. 4, pp. 97- 333, 2009.
[11] K.S. Lew, T.S. Dillon, and K.E. Forward, "Software Complexity and Its Impact on Software Reliability," IEEE Trans. Software Eng., vol. 14, no. 11, pp. 1645-1655, Nov. 1988.
[12] J. Li, "Prioritize Code for Testing to Improve Code Coverage of Complex Software," Proc. 16th IEEE Int'l Symp. Software Reliability Eng., pp. 75-84, 2005.
[13] J. Li, D. Weiss, and H. Yee, "Code-Coverage Guided Prioritized Test Generation," J. Information and Software Technology, vol. 48, no. 12, pp. 1187-1198, 2006.
[14] Z. Li, M. Harman, and R. Hierons, "Search Algorithms for Regression Test Case Prioritization," IEEE Trans. Software Eng., vol. 33, no. 4, pp. 225-237, Apr. 2007.