International Journal of Applied Engineering Research
ISSN 0973-4562 Volume 10, Number 9 (2015) pp. 23831-23838

http://www.ripublication.com

Estimating A Machine Learning Model For Fault Detection Using Test Case Prioritization Technique In Open Dependency

T. Dinesh Parthiban

Assistant Professor, Department of IT, PVP College of arts and science ,Dindigul
email id:tdineshparthiban@gmail.com

ABSTRACT

Tests can be run in any order so that functional dependencies that may exist between some test cases which lead to major complexity between the test cases. Where by using the Ranking algorithm the functional dependencies can be noted which is been useful to identify the relations between the classes of various dependencies file in the modules and flow pattern can be identified. In this paper, we present a Machine learning based learning model (Expectation Maximization algorithm) for test case prioritization techniques that use the probabilistic that hypothesis of this work is dependencies between tests are representative of interactions in the system which under test will executing complex interactions much earlier is likely to increase the fault detection rate, compared to arbitrary ordering the test. Empirical evaluations on software systems built toward industry use demonstrate that these techniques increase the rate of fault detection compared to the rates achieved by the improper prioritized orders, random orders and test suites ordered using existing "fine-grained" techniques based on functional evolution . Complexity can be reduced by using ranking algorithm. Flow of the test case can be identified.

Index terms: Machine learning Probabilistic methods, Expectation Maximization, Test Case Prioritization

1. INTRODUCTION

A software product development organization invests resources in product development and expects maximal added value from their investments. This means that providing value to different customer and end-user segments with products is a necessity for the business of product development companies. Providing value with the product requires, however, a successful selection of the requirements to be

implemented in the products. Requirements prioritization is defined as an activity during which the most important requirements for the system (or release) should be discovered. In practice, only a limited set of requirements can be implemented in one release, but the product must Satisfies the needs of the customers and should attain the high range to reach the markets in time. This means that trade-offs have to be made during the development work.

1.1 THE BACKGROUND OF THE RESEARCH

The ultimate sponsors of the project expect that the project's end result will be to add more value for them than they are paying the project team to create it. On a high level, this means that companies expect their product development organization to add more value to them than they invest in product development. Prioritizing requirements is recognized as an important activity to ensure value provision in product development. By definition, requirements prioritization is an activity during which the most important requirements for the system (or release) should be discovered. Origins for the importance of prioritization are in limited product development resources, since time and money are in practice. Where the customers thinking and their expectations are high and Short time period and the product must deliver the most valuable and essential functionality as early as possible. However, the scope of each release must be limited. The challenge is therefore to select the 'right' requirements out of a given superset of candidate requirements so that all the different key areas and their way of interest in several areas where it concern about their technical constraints, and the certain limitation of the of the critical stakeholders are fulfilled and the overall business value of the product is maximized. Requirements prioritization is one of the major challenging activity. It is widely 3 accepted that requirements prioritization involves complex decision-making. In order to make the prioritize requirements more successfully we need to concentrate on the domain knowledge and estimation skills are required. . In addition, requirements depend on each other and priorities are always relative. An important requirement in one release or to a certain customer may not be as important in the next release or to another customer.

The aim is to investigate the current state of practice in the area of requirements prioritization in software companies operating in the product business and the relationship between industrial practice and requirements prioritization methods from the literature.

The focus is on how the prioritization and selection of requirements is organized in software product development organizations and what the practical challenges can be obtained. Here to addition, the exact way of prioritization methods for solving these challenges is investigated.

Requirements prioritization is an activity during which the most important requirements for the system (or release) should be discovered. This concept originates from the context of the development of customer-specific systems, where all the requirements are to be concentrated, verified, documented, analyzed and validated within one project. Many of the findings concerning requirements prioritization in the literature, can, however, be generalized to concern the prioritization of features as well. To avoid unwanted complexity, the need of requirements prioritization can be

used as a general term for both feature and requirements prioritization. Additionally, the term prioritization practice is used as a general term for any activity performed to find the optimum implementation order of features or requirements.

2. RELATED WORKS

2.1 Test Case Prioritization

Prioritization is a process of scheduling test case to be executed in a particular order so that the test case with higher priority is executed first in the sequence. It is necessary to execute test suite in order of priority to utilize limited resource and time effectively. The main aim of is to increase the fault detection for a test suite. The priority is defined relative to some test criteria.

2.2 Open and closed dependency structure

In an Open dependency structure here there will be arising dependency between the two test cases t1 and t2 specifies that t1 must execute before t2 but not immediately.

A Closed dependency structure is not same as open dependency, dependency between the two test cases t1 and t2 specifies that t1 must execute immediately before t2. Some dependency structure contains both the open and closed dependency such structure is named as closed dependency structure. The closed dependency structure is regrouped into a single test, resulting in an open dependency structure.

2.3 Independent and dependent test case

Independent test case is a test case whose execution of one test case is not dependent on any other test cases.

Dependent test case is a test case whose execution of one test case is dependent on any other test cases.

There are several ways of tests can be related to these types of operations where we can consider only five steps for illustration:

1. Select a binary file,
2. Select a record file (a Non binary file),
3. If we attempted to read a binary file in these operations the drawback is where the selected file is not a Original or not a real binary file,
4. Attempt to update a binary file where the selected file is not a binary file, and
5. Attempt to update a binary where the selected file is a binary file and is successfully read.

2.4 Prioritizing test cases based on dependency structure

The dependency structure between test cases is closely related to the interaction between the parts of systems. They found that concatenating test together increases the fault detection rate of the tests due to the interaction occur between the tests.

Dependency structure prioritization is a technique that assigns priority based on a graph coverage value. The complexity of the dependents in test case can measured using the graph coverage value of a certain test case.

The Value of graph coverage of in a test case based on a open/closed dependency structure can be measured in two important ways.

1. The total number of dependents of the test case, and
2. The longest path of direct and indirect dependents of the test case.

The prioritization for open dependency structure is based on the two measurements DSP Volume and DSP Height.

DSP Volume

It is a measure which gives a higher weight to those test cases that have more dependents. To calculate the DSP volume of a test case, one need to calculate all direct and indirect dependents of that test case.

DSP Height:

The DSP height measure gives a higher weight to those test cases that have a higher dependent. Height of a test case, one needs to calculate the height of all paths form that test case, and take the length of the longest paths as a weight is used to calculate the DSP. This can be done using a straight forward depth-first search algorithm on the graph. The Prioritization for closed dependency structure is based on DSP Sum, DSP Ratio, DSP sum / ratio.

DSP Sum:

The DSP Sum coverage measure gives a higher weight to the paths that have more non executed test cases. To calculate the DSP Sum of a path, here it been used to counts only the number of non-executed test case in the path.

DSP Ratio:

The DSP ratio coverage measure gives a higher weight to paths that have a higher ratio of non-executed tests to executed tests, where by giving weight to certain longer paths. The weighted Sum of the path can be calculated by DSP in which the weight of a test case is its index in the path if it has not been executed, or otherwise, and then divides this by the height of the path.

DSP Sum/Ratio

The DSP sum / ratio coverage is used to define the number of non-executed test case. Where they can’t be used in the prioritizing suits.

3. Methods and Discussion

Granularity is the extent to which a system is broken down into small parts, either the system itself or its description or observation. It is the extent to which a larger entity is subdivided. For example, a yard broken into inches has finer granularity than a yard broken into feet. Coarse-grained systems consist of fewer, larger components than fine-grained systems; a coarse-grained description of a system regards large subcomponents while a fine-grained description regards smaller components of which

the larger ones are composed. The terms granularity, coarse, and fine are relative, used when comparing systems or descriptions of systems.

3.1 Machine learning

Several decades of research in the field have resulted in a multitude of different algorithms for solving a broad range of problems. To tackle a new application, a researcher typically tries to map their problem onto one of these existing methods, often influenced by their familiarity with specific algorithms and by the availability of corresponding software implementations. In this study, we describe an alternative methodology for applying machine learning, in which a be spoke solution is formulated for each new application. The solution is expressed through a compact modelling language, and the corresponding custom machine learning code is then generated automatically. This model-based approach offers several major advantages, including the opportunity to create highly tailored models for specific scenarios, as well as rapid prototyping and comparison of a range of alternative models. Furthermore, newcomers to the field of machine learning do not have to learn about the huge range of traditional methods, but instead can focus their attention on understanding a single modelling environment. In this study, we show how probabilistic graphical models, coupled with efficient inference algorithms, provide a very flexible foundation for model-based machine learning, and we outline a large-scale commercial application of this framework involving tens of millions of users. We also describe the concept of probabilistic programming as a powerful software environment for model-based machine learning, and we discuss a specific probabilistic programming language called Infer.NET, which has been widely used in practical applications

Model based learning. Typically, model-based machine learning will be implemented using a model specification language in which the model can be defined using compact code, from which the software implementing that model can be generated automatically. The key goals of a model-based approach include the following

1. The ability to create a very broad range of models, along with suitable inference or learning algorithms, in which many traditional machine learning techniques appear as special cases.
2. Each specific model can be tuned to the individual requirements of the particular application: for example, if the application requires a combination of clustering and classification in the context of time-series data, it is not necessary to mash together traditional algorithms for each of these elements (Gaussian mixtures, neural networks and hidden Markov models (HMMs), for instance), but instead a single, integrated model capturing the desired behaviour can be constructed.
3. Segregation between the model and the inference algorithm: if changes are made to the model, the corresponding modified inference software is created automatically. Equally, advances in techniques for efficient inference are available to a broad range of models.

4. Transparency of functionality: the model is described by compact code within a generic modelling language, and so the structure of the model is readily apparent. Such modelling code can easily be shared and extended within a community of model builders.
5. Pedagogy: newcomers to the field of machine learning have only to learn a single modelling environment in order to be able to access a wide range of modelling solutions. Because many traditional methods will be subsumed as special cases of the model-based environment, there is no need for newcomers to study these individually or indeed to learn the specific terminology associated with them.

4. Result and methods

To implement this system to estimate the fault detection during the application deployment therefore if fault detection to be fast then need to consider test case prioritization. Process of ordering the execution of test case to achieve the increasing the rate of fault detection. Tests can be run in any order so that functional dependencies that may exist between some test cases Where by using the Ranking algorithm the functional dependencies can be noted which is been useful to identify the relations between the classes of various dependencies file in the modules and flow pattern can be identified .Complexity can be reduced by using ranking algorithm. Flow of the test case can be identified. The main output of the result is reducing in the complexity between the modules and the flow of test can be identified.

4.1 Ranking Algorithm

The ranking approach to retrieval seems to be more oriented towards the end-users. Where ranking algorithm it will allow the customers or users to input a simple or small query such as a sentence or a phrase (no Boolean connectors) and retrieve a list of documents ranked in order of likely relevance.

The main reason the natural language/ranking approach is more effective for end-users is that all the terms in the query are used to retrieve the results based on co-occurrence while in query terms in the rank based method can be validate ,analyzed and modified by constant term-weighting, Where it can be explained in many circumstance. Ranking method can be used often to eliminates the wrong Boolean syntax which is used by the end-users, and these end users can provides some results even some query term is not explained or incorrect, which is not used in the any test term in the data. However there is a chance of misspelled also been taken in the account.

The ranking methodology also works well for the complex queries that may be difficult for end-users to express in certain Boolean logic. For certain conditions, "human issues and/or system details in medical databases" is difficult for end-users to express in Boolean logic because it contains many high- or medium-frequency words without any clear necessary Boolean syntax.

5. Conclusion and future work

We have a designed and implemented a test suite model for test case prioritization between the test cases. We present a Machine learning based learning model (Expectation Maximization algorithm) for test case prioritization techniques that use the probabilistic methods which yield better fault detection rate, in comparison with randomly generated test case using structure and functional priority. The nature of the techniques preserves the probabilistic range in sorting and indexing the test cases based on the ranking in the test ordering. The Results indicates that hypothesis of this work is dependencies between tests are representative of interactions in the system which is under test, and while it comes to execution there will be complex interactions is obtained to increase the fault detection rate, compared to arbitrary test orderings. In addition, for open dependency graphs, our techniques achieved better APFDs for most experiments than the state-of-the-art coarse-grained function coverage techniques First, information from previous test runs is not needed to calculate the priorities, so our techniques can be used on first versions of systems. Furthermore, it can be used even if previous test runs have not completed, which is useful in development processes containing short iterations. Second, maintaining fine-grained test suites and prioritizing these based on dependencies preserves the flexibility of fine-grained test suites, while also enabling larger test scenarios to be uncovered, thus increasing the probability of each test finding a fault. The result is reducing in the complexity between the modules and the flow of test can be identified. Where it can be specified to particular case .It says the dependencies between the modules and run flow of the case. In future without using functional dependencies complexity should be reduced by using the Machine learning approach.

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.