International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 6.887
Volume 6 Issue XI, Nov 2018- Available at www.ijraset.com

Online Feedback Information and Prediction Established Dynamic Partition for Cost Effective Software Testing

M. Bhuvaneswari[1], P. Geetha[2]
[1]*Research Scholar, Department of Computer Science, Kaamadhenu Arts & Science College, Sathy-648503, India*
[2]*Assistant Professor, Department of Information Technology, Kaamadhenu Arts & Science College, Sathy-648503, India*

Abstract: ***With the emerging complexity of today's software applications injunction with the increasing competitive pressure has pushed the quality assurance of developed software towards new heights. Software testing is an unavoidable part of the software development lifecycle and keeping in line with its criticality in the pre and post development process makes it something that should be catered with enhanced and efficient techniques and methodologies. A test case is used to find out the undiscovered error in the software. A large number of test cases are available in the test pool and it is required to select the most important test case in the cost effective manner. A family of dynamic partitioning algorithm was proposed to selectively execute test cases from a large number of test suites for evolving software. Based on online feedback the dynamic partitioning was performed. The online feedback was collected during test case executions. It doesn't refer any kind of change analysis, coverage information or human judgment. Hence it improved the cost effectiveness of software testing. In this paper, the cost effective software testing is further improved by integrating the dynamic partitioning and feedback based defect prediction method which is named as Cost Effective- Dynamic Partitioning Strategy-Defect Prediction (CE-DPS-DP). The dynamic partitioning strategy partition the test cases based on the online feedback information. The partitioned test cases are used in the feedback based defect prediction method. This method employs the local predicator and global predicator for defect prediction. The local predicator and global predicator are combined by weight to output the prediction results. Thus the integrated dynamic partitioning and feedback based defect prediction improves the prediction accuracy and maintains the cost for software testing effectively. The experimental results show that the proposed approach provides better results than the conventional approach.***
Keywords: Software testing, dynamic partitioning, defect prediction, cost-effective software testing.

I. INTRODUCTION

Generally, Software testing [1] ensures the desired developed software meets its predefined objectives. Software testing is an important cost factor in the software development life-cycle [2]. Test activities are unavoidable and important to guarantee high quality software products. It is also an expensive and time consuming work, during which generally, software engineers need to spend 50%-60% development costs on software testing activities [3]. During Software Testing Process (STP), the situation of test resource constraints is a common phenomenon due to the limitation of development cost or deadline in software development. Defect prediction provides an effective way to reveal the likely defective parts before testing [4]. Its results are often used to guide the testing strategy.

A dynamic partitioning strategy [5] was proposed as a cost-effective software testing strategy which was an online partitioning strategy where test cases were selected based on the feedback information. In dynamic partitioning strategy, the partitioning was performed online rather than off-line and it was not based on program specifications or code. In this paper, the cost-effective software testing strategy is further improved by proposed Cost Effective- Dynamic Partitioning Strategy-Defect Prediction (CE-DPS-DP).

The CE-DSP-DP integrates the dynamic partitioning and feedback based defect prediction. The partitioned test cases by dynamic partitioning are used in the test action process of feedback based defect prediction method. The defect prediction method used the local and global predicator to predict the defects in software. The local predicator is trained by the data generated during the Software Testing Process (STP) and the global predicator is trained by the whole data. The local predicator and global predicator are combined by weight to output the prediction results.

II. LITERATURE SURVEY

A hybrid model called Artificial Neural Network (ANN) optimized by Artificial Bee Colony (ABC) [6] model was proposed for software defect prediction. In this model, a cost-sensitivity feature was added to ANN by using a parametric fitness function. A trade-off was made between the classification performance of majority and minority classes by the change of cost coefficients. However, the ANN is hardware dependence.

A multi-objective optimization based supervised method called MULTI [7]was proposed to build Just-In-Time Software Defect Prediction (JIT-SDP) models. In MULTI, JIT-SDP was formalized as a multi-objective optimization problem. One of the multi-objectives was designed to maximize the number of identified buggy changes and another multi-objective was designed to reduce the efforts in software quality assurance activities. But there was a conflict between these two objectives. MULTI utilized logistic regression and NSGA-II to build the models and to generate a set of non-dominated solutions respectively. However, this method is not more effective.

A Software Defect Prediction Model Learning Problem (SDPMLP) [8] was proposed for software defect prediction. In SDPMLP model, classification model was employed to select appropriate relevant inputs from a large volume of input and learned the classification function. SDPMLP model was a combinatorial optimization problem with factorial complexity and proposed two hybrid exhaustive search and Probabilistic Neural Network (PNN) and Simulated Annealing (SA) and PNN procedures to solve it. For small size, SDPMLP model was performed well and it provided an optimal solution. But for small size, the use of exhaustive search PNN approach was not pragmatic and only the SA–PNN was allowed to solve the SDPMLP in a practical time limit.

A general framework [9] was proposed for software defect prediction. This framework was consisted of scheme evaluation and defect prediction components. The scheme evaluation analyzed the prediction performance of competing learning schemes for given historical data sets. The defect predictor built models according to the evaluated learning scheme and predicted software defects with new data according to the constructed model. But this framework is little conservative.

A learning-to-rank approach [10] was proposed to predict the exact number of defects in software. This approach constructed software defect prediction model by directly optimizing the ranking performance. This approach was processed in two aspects are one is a novel application of the learning-to-rank approach to real world data sets for software defect prediction and the other is a comprehensive evaluation and compared the learning-to-rank method with other algorithms which was used to predict the order of software modules according to the predicted number of defects. However, this approach is more effective.

III. PROPOSED METHODOLOGY

In this paper, Cost Effective- Dynamic Partitioning Strategy-Defect Prediction (CE-DPS-DP) where the dynamic partitioning and feedback based defect prediction is integrated to improve the cost effectiveness of software testing. The dynamic partitioning, partition the test cases based on feedback information. Then the feedback based defect prediction model includes two closed feedback loops. One is the feedback loop during testing and the other is the feedback loop after testing. The first loop consists of four components are the predictor, the software under testing and the training data. The predictor produces prediction results for the Software Under Testing (SUT). The predicator generates prediction results for the SUT. Then the software parts are arranged in descending order of the defect rate according to this result. After that, a fixed number of top software parts are opted as the test set and the partitioned test cases by the dynamic partitioning are combined with the test set and it is executed by a test action. At the end of test action, the test data generated by test action will be treated as new data and be added into the training data. Finally, a modified predictor is constructed through model training algorithm based on the new training data.

Through the collaboration between the predictor and the test action, the entire STP works in an optimal manner until the test target is met. After achieving the test target, the second loop comes next. The test materials produced in whole STP will be collected and be recorded in the test library. Their data serve as a part of global data when a new SUT is tested. It should be noted that, in order to ensure the data quality used for constructing predictor, the global data should be selected through the global data filter.

A. DP using feedback from STP

DP is used code metrics and defect results as the training data for constructing a predictor that is a black box system which is reflected the relationship among software metrics and software defects.

Once the new software parts with metrics are provided for the predictor, the defect outcome of the novel software part is achieved. DP researches incorporate two goals: ranking software parts according to defect-proneness and classifying whether or not new software parts have defect. It is taken the ranking task as the target of defect prediction since ranking result is more flexible in

determining the priority ordering than classification result. For t-DP, there is no interaction between software testing and defect prediction during the STP. Once the defect prediction model is established, the prediction results cannot be modified.

The quality of training data cannot be guaranteed, especially when there is a major deviation between the characteristics of the training data and those of the SUT, in which case, adopting defect prediction may lead to worse test performance. To overcome this problem, defect prediction with the feedback control approach is combined. Feedback control theory is commonly employed in software testing. It can adjust the testing strategy on-line by employing the test results generated during the STP as the feedback to guide the subsequent testing work.

In the feedback-based software testing model, the STP is treated as a control problem, and the testing strategy serves as a controller to provide decision for the SUT in order to optimize the test action. The test results generated by the test action are collected as a part of historical data, which are treated as feedback information to support the controller. The controller, the SUT, the test action and the database constitute a closed feedback loop, increasing the convergence speed of the output variables to the expected values. Using the framework, an enhanced feedback-based defect prediction model is proposed with some additional criteria (severity level) in order to systematically describe the interaction between defect prediction, guide the software parts selection and the STP in a formal manner.

B. Global data filter

Global data mainly come from the training data selected from the test library, and also include the local data generated during the STP. It is well known that predictor can be constructed based on the rich historical data from other projects to compensate for the poor performance caused by the lack of local data in the early test stage. However, the quality of real cross-project data is usually unstable. Using the data without filtering to construct the predictor may not be able to achieve the desired effect. The global data filter focuses on the selection of training data from the test library and it can effectively improve the quality of global data.

The filter is a formula for calculating software similarity, and the data with high similarity is retained by adjusting the data filter's threshold of the similarity. The similarity of these sub-items is evaluated; then a comprehensive calculation formula is derived to obtain the similarity between other projects and the SUT; finally, the similarity value is used to filter the global data. SUT represents the software under test, CP means cross project software, n represents sub-item, and Sim indicates the similarity between the CP and the SUT. The $Sim\_name[n]$ is the similarity of the sub-item n.

For category estimation,

$$Sim_{category}[n] = \begin{cases} 1 & SUT_{category}[n] == CP_{category}[n] \\ 0 & otherwise \end{cases} \quad (1)$$

$$Sim_{category} = \frac{1}{3} * \sum_{n=1}^{3} Sim_{category}[n] \quad (2)$$

To level valuation,

$$Sim_{level}[n] = 1 - \frac{1}{4} * |SUT_{level}[n] - CP_{level}[n]| \quad (3)$$

$$Sim_{level} = \frac{1}{2} * \sum_{n=1}^{2} Sim_{level}(n) \quad (4)$$

To feature estimation,

$$Sim_{feature}[n] = \begin{cases} 1 & SUT_{feature}[n] == CP_{feature}[n] \\ 0 & otherwise \end{cases} \quad (5)$$

$$Sim_{feature} = \frac{1}{5} * \sum_{n=1}^{5} Sim_{feature}[n] \quad (6)$$

The similarity among the candidate software and the SUT are achieved using,

$$Sim = \alpha * Sim_{category} + \beta * Sim_{level} + \chi * Sim_{feature} \quad (7)$$

If the similarity of the candidate software is greater than the threshold, after that the data of software will be chosen as global data.

C. Determine the severity

A risk is the chance of damage, injury or loss and is established during the probability of its impact and its occurrence. The standard risk system is employed in the proposed approach is using the two factors probability (P), establishing the likelihood that a failure assigned to a risk happens and impact (I), deciding the severity of a failure if it occurs in operation.

A risk exposure R of an arbitrary risk item a is established using the probability P and the impact I,

$$R(a) = P(a)_{\circ} I(a) \quad (8)$$

So, the risk exposure R denotes a comparable value which determines the risk, that is, an uncertain condition or event which, if it happens, has a negative effect on the system. The binary operator $_{\circ}$ which connects P *and* I is the multiplication of two numbers or the cross product of two numbers or arbitrary characters.

In the risk identification step, risk items are identified and a list of risk items covering the whole system under test is compiled. Risk items are elements to which tests are assigned and for which the risk exposure is calculated. In the test planning and control step, the test plan is defined and controlling that is an ongoing activity in parallel to the other activities in the software test procedure, is initiated. In this process, the test plan considers the risk model.

In the risk-assessment step, the risk-exposure value is calculated and classified for each risk item based on probability and impact factors. In the test analysis and design step, a concrete test schedule is defined based on the test plan and a concrete risk classification. In the test implementation and execution step, the test schedule is executed. The execution of the test cases is determined by their priority and the resource limitations. As an outcome of the test execution, a test log is created. The test implementation and execution are typically performed by testers.

In the test evaluation and reporting step, the test log data is evaluated and a test report is created to support decisions of the test or project management. The report emphasizes an estimation of the mitigated risks and the residual risks. The test evaluation and reporting are typically performed by a test manager. In the test closure activities step, experiences from the actual test project are evaluated. The test closure is a very important activity to steadily adapt risk-based testing according to the experiences of the test organization. The test closure activities are typically led by a test manager.

D. Feedback based integrated prediction (FIP) approach with Severity level

The proposed approach is assumed the difference among the local data source and the global data source and adopted a distinguished treatment strategy. Local predictor is trained on the local data and global predictor is constructed on the global data. Two predictors are used to predict the defect-proneness of software parts. In addition, severity level is also considered for software parts selection. After that, the final prediction results are integrated by weight. The proposed approach is to utilize the local data generated during the STP, so as to improve the prediction accuracy.

The local data are more valuable than the external filtered data and should be paid more attention since it directly comes from the SUT. A common practice is to give the local data a larger weight and generate a predictor with the external filtered data. However, if the external data have too large proportion in the integrated data, the local data with high weight can still be overwhelmed due to the imbalance in the amount of data. As a result, the characteristics of the software defect cannot be reflected by the prediction model. Therefore, the local data is used to construct a predictor alone. By allocating weights directly to the predictors rather than the data, the local data can get a larger proportion.

1) Algorithm: CE-DPS-DP Approach

a) Set test time $t = 0$. Initialize local data $D_L = NULL$, software parts $S_M = SUT$, global data $D_G = Filter(Library)$

b) Preprocess D_L *and* D_G

c) Utilize the Linear Regression Model for constructing local predictor $Model\_L$ using local data and global predictor $Model\_G$ using global data

d) Employ $Model_L$ *and* $Model_G$ for predicting the defect-proneness of software parts in SUT at time t $S_M^{(t)}$ and achieve the outcomes local defect probability at time t $P_L^{(t)}$ and global defect probability at time t $P_G^{(t)}$

e) Use risk based testing approach for predicting risk exposure RE and obtains risk exposure probability at time t $P_{RE}^{(t)}$

f) Normalize $P_L^{(t)}, P_G^{(t)}$ *and* $P_{RE}^{(t)}$

$$\begin{cases} P_L^{(t)}(i) = P_L^{(t)}(i) / \sum_{i=1}^{m} P_L^{(t)}(i) \\ P_G^{(t)}(i) = P_G^{(t)}(i) / \sum_{i=1}^{m} P_G^{(t)}(i) \\ P_{RE}^{(t)}(i) = P_{RE}^{(t)}(i) / \sum_{i=1}^{m} P_{RE}^{(t)}(i) \end{cases} \quad (9)$$

In the above equation, m denotes the whole number of software parts in S_M and i indicates the $i-th$ software module

g) Update weight of global predictor at time t $\omega_G^{(t)}$ and weight of the local predictor $\omega_L^{(t)}$

$$\omega_L^{(t)} = \begin{cases} \frac{W}{R} * (state) & state < R \\ W & state \geq R \end{cases} \quad (10)$$

$$\omega_G^{(t)} = 1 - \omega_L^{(t)} \quad (11)$$

In the above equation, $state$ denotes the proportion of tested software parts, W indicates weight of threshold and R represents ratio threshold

h) Obtain the integrated defect proneness $P^{(t+1)} = \omega_L * P_L^{(t+1)} + \omega_G * P_G^{(t+1)} + P_{RE}^{(t+1)}$

i) Rank integrated defect probability at time t $P^{(t)}$ and choose the top K (test steps of test action) parts from S_M as the test set S_T (software parts under test), $S_M = S_M - S_T$

j) Include the test case generated by dynamic algorithm with S_T and represented as S_{DT}

k) Test S_{DT} and output D_T

l) Update the local and global data, $D_L = D_L + D_T$ *and* $D_G = D_G + D_T$, respectively

m) $t = t + 1$

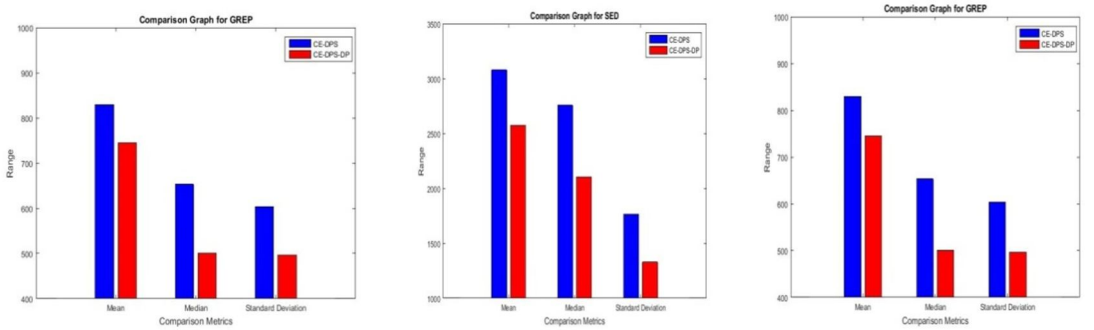
n) If the test target is obtained, after that result the local data D_L; else go to step 2.

IV. RESULT AND DISCUSSION

In this section, the performance of the existing Cost Effective- Dynamic Partitioning Strategy (CE-DPS) and proposed CE-DPS-Defect Prediction (CE-DPS-DP) is analyzed in terms of mean, median and standard deviation. The experiment is carried out in both small and large subject programs are SPACE, SED and GREP. The following Table 1 defines the SPACE, SED and GREP programs.

TABLE I
PROGRAM DEFINITION

Programs	No. of Lines	Coding Language	No. of functions
SPACE	6,199	C code	136
SED	14,427	C code	255
GREP	10,068	C code	146



(a) (b) (c)

Fig. 1 Experimental results. (a) Results of Experiments with SPACE, (b) Results of Experiments with SED, (c) Results of Experiments with GREP

For each algorithm and each program, the experiment has been repeated 1000 times and the mean, median and standard deviation data have been calculated. Fig. 1a, 1b and 1c show the comparison between CE-DPS and CE-DPS-DP in terms of mean, median and standard deviation for SPACE, SED and GREP respectively. The comparison metrics are represented in X axis and range of mean, median and standard deviation is represented in Y axis. From Fig.1 it is understand that the proposed CE-DPS-DP has better mean, median and standard deviation than CE-DPS for SPACE, SED and GREP programs.

V. CONCLUSION

A Cost Effective- Dynamic Partitioning Strategy-Defect Prediction (CE-DPS-DP) is proposed for an efficient cost-effective software testing. It consists of two processes are dynamic partitioning and feedback based defect prediction. In dynamic partitioning process, the test cases are partitioned based on online feedback. In feedback based defect prediction process, the defects in the software are predicted by using local and global predictor where the partitioned test cases by dynamic partitioning are used. The experimental results show that the proposed CE-DPS-DP has better mean, median and standard deviation than the conventional method.

REFERENCES

[1] O. A. L. Lemos, F. F. Silveira, F. C. Ferrari, and A. Garcia, "The impact of Software Testing education on code reliability: An empirical assessment", Journal of Systems and Software, 137, 497-511, 2018.
[2] Y. Amannejad, V. Garousi, R. Irving, and Z. Sahaf, "A search-based approach for cost-effective software test automation decision support and an industrial case study", In 2014 IEEE Seventh International Conference on Software Testing, Verification and Validation Workshops (ICSTW), 302-311, 2014.
[3] T. T. Nguyen, T. Q. An, V. T. Hai, and T. M. Phuong, "Similarity-based and rank-based defect prediction", In 2014 IEEE International Conference on Advanced Technologies for Communications (ATC), 321-325, 2014.
[4] T. Menzies, J. Greenwald, and A. Frank, "Data mining static code attributes to learn defect predictors", IEEE transactions on software engineering, (1), 2-13, 2007.
[5] Z. Q. Zhou, A. Sinaga, W. Susilo, L. Zhao, and K. Y. Cai, "A cost-effective software testing strategy employing online feedback information", Information Sciences, 422, 318-335, 2018.
[6] Ö. F. Arar, and K. Ayan, "Software defect prediction using cost-sensitive neural network", Applied Soft Computing, 33, 263-277, 2015.
[7] X. Chen, Y. Zhao, Q.Wang, and Z. Yuan, "MULTI: Multi-objective effort-aware just-in-time software defect prediction", Information and Software Technology, 93, 1-13, 2018.
[8] P. C. Pendharkar, Exhaustive and heuristic search approaches for learning a software defect prediction model. Engineering Applications of Artificial Intelligence, 23(1), 34-40, 2010.
[9] Q. Song, Z. Jia, M. Shepperd, S. Ying, and J. Liu, "A general software defect-proneness prediction framework", IEEE Transactions on Software Engineering, 37(3), 356-370, 2011.
[10] X. Yang, K. Tang, and X. Yao, "A learning-to-rank approach to software defect prediction", IEEE Transactions on Reliability, 64(1), 234-246, 2015.