Software Cost Regressing Testing Based Hidden Morkov Model

[1]**Mrs. P.Thenmozhi**, [2]**Dr. P. Balasubramanie**,
[1]Assistant Professor, Kongu Arts and Science College,
Erode – 638 107, Tamil Nadu, India,
[2]Professor & Head, Department of Computer Science and Engineering,
Kongu Engineering College,Perundurai – 638052, Tamil Nadu,

***Abstract*— Maintenance of software system accounts for much of the total cost associated with developing software. The nature of the modifying the software is a highly error-prone task which is the main reason for the cost. Correcting fault by changing software or add new functionality can cause existing functionality to regress, introducing new faults. To avoid such defects, one can re-test software after modifications, a task commonly known as regression testing. Re-execution of test cases developed for previous versions is typically called Regression test. However, is often costly and sometimes even infeasible due to time and resource constraints. Re-running test cases that do not exercise changed or change-impacted parts of the program carries extra cost and gives no benefit. This paper presents a novel framework for optimizing regression testing activities, based on a probabilistic view of regression testing. The proposed frame- work is built around predicting the probability that each test case finds faults in the regression testing phase, and optimizing the test suites accordingly. To predict such probabilities, we model regression testing using a Hidden Morkov Model Network (HMMN), a powerful probabilistic tool for modeling uncertainty in systems. We build this model using information measured directly from the software system. The results show that the proposed framework can outperform other techniques on some cases and performs comparably on the others. This paper shows that the proposed framework can help testers improve the cost effectiveness of their regression testing tasks.**

Keywords: Software testing, Testing tools, Regression testing, Software maintenance

1. Introduction

The nature of Software systems is to evolve with time and specially as a result of maintenance tasks. Software maintenance is defined as “The modification of a software product after delivery to correct faults, to improve performance or other attributes, or to adapt the product to a modified environment”.

The presence of a costly and long maintenance phase in most software projects, specially those manipulating large systems, has persuaded engineers that software evolution is an inherent attribute of software development. Moreover, maintenance activities are reported to account for high proportions of total software costs, with estimates varying from 50% in the 80s to 90% in recent years. Reducing such costs has motivated many advancements in software engineering in recent decades. The objective of maintenance is “to modify the existing software product while preserving its integrity”. The later part of the stated objective, preserving integrity, refers to an important issue raised as a result of software evolution. One need to ensure that the modifications made to the product for maintenance have not damaged the integrity of the product.

From theory and practice that changing a system in order to fix bugs or make improvements can affect its functionality in ways not intended. These potential side effects can cause the software system to regress from its previously tested behavior, introducing defects called regression bugs. Although rigorous development practices can help isolate modifications, the inherent complexity of modern software systems prevents us from accurately predict the effects of a change. Practitioners recognized such a phenomenon and hence are reluctant to change their programs in fear of introducing new defects. Researchers have tried to find ways of analyzing the impact of a change on different parts of a system and predicting the effects. In absence of formal presentations of software systems, however, such attempts, although helpful, will not provide the required confidence levels.

Unless we are able to find regression bugs, once they occur, Software maintenance remains a risky task. Despite the introduction and adaptation of other verification methods (such as model checking and peer reviews), testing remains the main tool to find defects in software systems. Naturally, retesting the product after modifying it is the most common way of finding regression bugs. Such a task is very costly and requires great of organizational effort. This has motivated a great deal of research to understand and improve this crucial aspect of software development and quality assurance.

This paper is organized as follows. Literature surveys are given in section 2. In section 3 we will devote ourselves to discussing the probabilistic modeling and reasoning in detail. Conclusions will be drawn in section 5.

2. Literature survey

In this section, research areas related to the topic of this paper are elaborated. The subject to start with is that of the problem in question, “Software Regression Testing”. There exists an extensive body of research addressing this problem using many different approaches. This section takes a critical look at this line of research, trying to find strong points and ideas as well as the gaps. Through this examination, many terms and concepts related to software testing area will be introduced as well.

2.1 Software Regression Testing

Research in regression testing spans a wide range of topics. Earlier work in this area investigated different environments that can assist regression testing. Such environments particularly emphasize automation of test case execution in the regression testing phase. For example, techniques such as capture playback have been proposed to help achieve such an automation. Furthermore, test suite management and maintenance have been addressed by much research. Measurement of regression testing process has also been researched extensively and many models and metrics have been proposed for it. Most of the research work in this area, however, has focused on test suite optimization.

Test suite optimization for regression testing consists of altering the existing test suite from a previous version to meet the needs of regression testing. Such an optimization intends to satisfy an objective function, which is typically concerned with reducing the cost of retesting and increasing the chance of finding bugs (reliability). There exists a variety of techniques addressing this problem. Most of these techniques can be categorized into two families of test case selection and test case prioritization. Regression test selection techniques reduce testing costs by including a subset of test cases in the new test suite. These techniques are typically not concerned with the order in which test cases appear in the test suite. Prioritization techniques, on the other hand, include all test cases in the new test suite but change their order in order to optimize a score function, typically the rate of fault detection. These two approaches can be used together; one can start with selecting a subset of test cases and then prioritize those selected test cases for faster fault detection. The rest of this section first looks into test case selection approaches from the literature and then touches up to an existing techniques for test case prioritization.

2.1.1 **Test Case Selection**

Test case selection, as the main mechanism of selective regression testing, have been widely studied using a variety of approaches. In a survey of techniques proposed up to 1996, Rothermel and Harrold[12] propose an approach for comparison of selection techniques and discuss twelve different family of techniques form the literature accordingly.They evaluate each technique based on four criteria: inclusiveness (the extent to which it selects modification revealing tests), precision (the extent to which it omits tests that are not modification revealing), efficiency (its time and space required), and generality (its ability to function on different programs and situations). These four criteria, in principle, capture what we expect form a good test case selection approach. These four criteria inherently impose a trade-off situation where proposed techniques usually satisfy one of the criteria in expense of the others.

Main Approaches

An early trend in test selection research evolved around minimizing test cases selected for regression. This approach, often called test case minimization, is based on a system of linear equations to find test suites that cover modified segments of code. Linear equations are used to formulate the relationship between test cases and program segments (portions of code through which test execution can be tracked, e.g., basic-blocks or routines). This system of equations is formed based on matrices of test-segment coverage, segments-segment reachability and (optionally) definition-use information about the segments. A 0-1 integer programming algorithm is used to solve the equations (an NP-hard problem) and find a minimum set of test cases that satisfies the coverage conditions. This approach is called minimization in the sense that it selects a minimum set of test cases to achieve the desired coverage criteria. In doing so, test cases that do cover modified parts of code can be omitted because other selected test cases cover same segments of the code.

A different set of approaches have focused on developing safe selection techniques. Safe techniques aim to select a subset of test cases which could guarantee, given certain preconditions, that the left-out test cases are irrelevant to the changes and hence will pass. Informally speaking, the aforementioned conditions as described in are:

- the expected result for test cases have not changed from the last version to the current version;
- Test cases execute deterministically (i.e., different executions results in identical execution path).

Safe techniques first perform change analysis to find what parts of the code can be possibly affected by the modifications. Then, they select any test case that covers any of the modification-affected areas of the code. Safe techniques are inherently different from minimization techniques in that they select all test cases that have a chance of revealing faults. In comparison, safe techniques usually result in a larger number of selected test cases but also achieve a much better accuracy.

Many techniques are neither minimizing nor safe. These techniques typically use a certain coverage requirement on modified or modification affected parts of code to decide whether a test case should to be selected. For example, the so-called dataflow-coverage-based techniques. select test cases that exercise data interactions (such as definition-use pair) that have been affected by modifications. These selections techniques are different in two aspects: the coverage requirement they target and the mechanism the use to identify of modification-affected code. For example, Kung et al[10] propose a technique which accounts for the constructs of object-oriented languages. In performing change analysis, their approach takes into account object-oriented notions such as inheritance. The relative performance of these selection techniques tend to vary from program to program, a phenomenon that could be understood only through empirical studies.

Cost Effectiveness

Many empirical studies have evaluated the performance of the test case selection algorithms. In general, these empirical studies show that there is an efficiency-effectiveness (or inclusiveness-precision in terminology) tradeoff between different approaches to selection. Some (such as safe) techniques reduce the size of test suite by a small factor but find most (or all) bugs detectable with existing test cases. Others (such as minimization techniques), reduce the size dramatically but can potentially leave out many test cases that can in fact reveal faults. Other techniques are somewhere in between; they may miss some faults but they reduce the test suite size significantly. The presence of such a tradeoff situation renders the direct comparison of techniques hard.

A meaningful comparison between regression testing techniques requires answering one fundamental question: is the regression effort resulting from the use of a technique justified by the gained benefit? To answer such a question one needs to quantify the notions of costs and benefits associated with each technique. To that goal, researchers have proposed models of cost-benefit analysis. These modelstry to capture the cost encountered as a result of missing faults, running test cases, running the technique itself including all the necessary analysis, etc. The most recent of all these models is that of Do et al[5]. Their approach computes costs directly and in dollars and hence is heavily dependent on good estimations of real costs from the field. An important feature of their model is that it can compare not only test case selection but also prioritization techniques. Most interestingly, it can compare selection techniques against prioritization techniques.

The existence of the mentioned trade-off has also encouraged the researchers to seek multi-objective solutions to the test selection problem. Yoo and Harman[15] have proposed pareto efficient multi-objective test case selection. They use genetic algorithms to find the set of pareto optimal solutions to two different representations of the problem: a 2-dimensional problem of minimizing execution time and maximizing code coverage and the 3-dimensional problem of minimizing time and maximizing both code coverage and history of fault coverage. The authors compare their solutions to those of greedy algorithms and observe that greedy algorithms surprisingly can outperform genetic algorithms in this domain. Coverage information, a necessary input to most existing techniques, can be measured only if the underlying code is available and its instrumentation is cost effectively possible. To be able to address more complex systems, where those conditions do not hold, some recent techniques have shifted their focus to artifacts other than code, such as software specification and component models. These techniques typically substitute code based coverage information with information gathered from formal (or semi-formal) presentations of the software. Orso et al[11]., for example, use component meta data to analyze the modifications across large component-based systems. The trend in current test case selection research seems to be that of using new sources of information or formalizations of a software system to understand the impacts of modifications.

2.1.2 Test Case Prioritization

The regression Test Prioritization (RTP) problem seeks to re-order test cases such that an objective function is optimized. Different objective functions render different instances of the problem, a handful of which have been investigated by researchers. Besides targeted objective functions, the existing body of prioritization techniques typically differs in the type of information they exploit. The algorithm employed to optimize the targeted objective function, also, is another source of difference between the techniques.

Conventional Coverage-based Techniques

Test case prioritization is introduced in [16] by Wong et al. as a flexible method of selective regression testing. In their view, RTP is different from test case selection and minimization in that it provides a means of controlling the number of test cases to run. They propose a coverage-based prioritization technique and specify cost per additional coverage as the objective function of prioritization. Given the coverage information recorded from a previous execution of test cases, this coverage-based technique orders test cases in terms of the coverage they achieve according an specific criterion of coverage (such as the number of covered statements, branches, or blocks). Because the purpose of RTP in their work is selective regression testing, they compare its performance against minimization and selection techniques. The coverage-based approach to prioritization is built upon by Rothermel et al. in [13].

They refer to early fault detection as the objective of test case prioritization. They argue that RTP can speed up fault detection, an advantage besides the flexibility it provides for selective regression testing. Early detection makes faults less costly and hence is beneficial to the testing process. They introduce Averaged Percentage of Faults Detected (APFD) metric to measure how fast a particular test suite finds bugs. They also introduce many variations of coverage-based techniques, using different criteria for coverage such as branch coverage, statement coverage, and fault-exposing-potential. These coverage-based techniques differ not only on the coverage information they use, but also on their optimization algorithm. When ordering test cases according to their coverage, a feedback mechanism could be used. Here, feedback means that each test case is placed in the prioritized order taking into account the effect of test cases already added to the order. A coverage-based technique with feedback prioritizes test cases in terms of the numbers of additional (not-yet covered) entities they cover, as opposed to total number of entities. This is done using a greedy algorithm that iteratively selects the test case covering the most not-yet-covered entities until all entities are covered, then repeats this process until all test cases have been prioritized. For example, assume we have a system with six elements: $e_1. . . e_6$and the coverage relations between test cases and elements are as follows:$t_1 \rightarrow \{e_2, e_5\}$, $t_2 \rightarrow \{e_1, e_3\}$, $t_3 \rightarrow \{e_4, e_5, e_6\}$. According to a coverage based technique, the first chosen test case is t_3 because it covers three elements, while the others cover two elements each. After selecting t_3, two test cases are left, both of which cover two elements. In the absence of feedback, we would choose randomly between the remaining two. However, we know that $e_5$is already covered by t_3; therefore $t_1$has merely one additional coverage, whereas t_2 has two. After adding t_3, we can update the model of coverage data such that the already tested elements do not effect subsequent selections. This allows choosing $t_2$before $t_1$based on its additional coverage. The notion of using additional coverage is what feedback mechanism provides; techniques employing feedback are often called additional. Many empirical studies have been conducted to evaluate the performance of coverage-based approach [13], most of which use APFD measure for comparison. These studies show that coverage-based techniques can outperform control techniques (including random and original ordering) in terms of APFD but have a significant room for improvement comparing to optimal solutions. They also indicate that in many cases, feedback employing techniques tend to outperform their non-feedback counterparts, an observation which could not be generalized to all cases. Indeed, an important finding of all these studies is that the relative performance of different coverage-based techniques depends on the programs under test and the characteristics of its test suite. Inspired by this observation, Elbaum et al.[6] have attempted to develop a decision support system (using decision trees) to predict which technique works better for what product/process characteristics. Many research works have enhanced the idea of coverage-based techniques by utilizing new sources of information. Srivastava et.al.[1] propose the Echelon frame work for change-based prioritization. Echelon first computes the basic blocks modified from the previous version (using binary codes) and then prioritizes test cases based on the number of additional modified basic blocks they cover. A similar coverage criteria used in the context of aviationindustry called Modified Condition/Decision Coverage (MCDC) is utilized in. Elbaum et al.[6] use metrics of fault-proneness, called fault-index, in order to guide their coverage-based approach to focus on the parts of code more prone to containing faults. Recently, in, Jeffery and Gupta[4] propose incorporating to prioritization a concept extensively used in test selection called relevant slices, modified sections of the code which also impact the outcome of a test case. Their approach prioritizes test cases according to the number of relevant slices they cover. Most recently, Zhang et al.[17] propose a technique which could incorporate varying test coverage requirements and prioritize accordingly. They work also takes into account different costs associated with test cases.

Recent Approaches

Walcott et al.[15] formulate a time-aware version of the prioritization problem in which a limited time is available for regression testing and also the execution time of test cases are known. Their optimization problem is to find a sequence of test cases that could be executed in the time limit and also maximize speed of code coverage. They use genetic algorithms to find solutions to this optimization problem. Their objective function of optimization is based on summations of coverage achieved, weighted by execution times. Their approach could be thought of as a multiobjective optimization problem where most coverage in minimum time is required.

All the code coverage-based techniques assume the availability of source/byte code. They also assume that the available code can be instrumented to gather coverage information. These conditions do not always hold. The code could be unavailable or excessively expensive to instrument. Hence, researchers have explored using other source of information for test case prioritization.

Srikanth et al. [7] have proposed PORT framework which uses four different requirement-related factors for prioritization: customer-assigned priority, volatility, implementation complexity, and fault-proneness. Although the use of these factors is conceptually justifiable and based on solid assumptions, their subjective nature (especially the first and third factors) make the outcome dependent on the perceptions of customers and developers. While it is hard to evaluate or rely on such approaches, it should be understood that it is the subjective nature of requirement engineering that imposes such properties. Also, their framework is not concerned with specifics of regression testing but prioritization in general.

Bryce et al. have proposed a prioritization technique for Event-Driven Software (EDS) systems. In their approach, the criteria of t-way interaction coverage is used to order test cases. The concept of interactions is defined in terms of events and the approach is tested on GUI-based systems and against traditional coverage based systems. Based on a similar approach, Sampath et al[1]. target prioritization of test cases developed for web applications. Their technique prioritizes test cases based on different criteria such as test lengths, frequency of appearance of request sequences, and systematic coverage of parameter-values and their interactions. Taking a different approach from coverage-based techniques, Kim and Porter[9] propose using history information to assign a probability of finding bugs to each test case and prioritize accordingly. Their approach, inspired by statistical quality control techniques, can be adjusted to account for different history-based criteria such as history of execution, history of fault detection, and history of covered entities. These criteria, respectively, give precedence to test cases that have not been recently executed, have recently found bugs, and have not been recently covered. From a process point of view, history-based approach makes the most sense when regression testing is performed frequently, as opposed to a one-time activity. Kim and Porter evaluate their approach in such a process model (i.e., considering a sequence of regression testing sessions) and maintain that comparing to selection techniques and in the presence of time/resource constraints, it finds bugs faster.

Most recently, Qu et al.[2] use the history of test execution for black-box testing and build a relation matrix between test cases. This matrix is used to move the test cases up or down in the final order. Their approach also includes some algorithms for building and updating such a matrix based on outcome of test cases and types of revealed faults. In addition to research works addressing the stated prioritization problem directly, there are research closely related to this area but from different perspectives. Saff and Ernst use behavior modeling to infer developers' beliefs and propose a test reordering schema based on their models. They propose running test cases continuously in background while software is being modified. They claim their approach leads to reducing the wasted time of development by approximately 90%. Leon and Podgurski[3] compare coverage-based techniques of regression testing with another family called distribution-based. Distribution-based approaches look at the execution profile of test cases and use clustering techniques to locate test cases that can reveal faults better. The experiments indicate that distribution based approach can be as efficient or more efficient compared to coverage-based. Leon and Podgurski, then, suggest combining these two approaches and report achieved improvement using that strategy.

3. Probabilistic Modeling and Reasoning

The probability theory provides a powerful way of modeling systems. It is especially useful for situations where the effects of events in a system are not fully predictable and a level of uncertainty is involved. The behaviors of large complex software systems are sometimes hard to precisely model and hence probabilistic approaches to software measurement have gained attention.

In the center of modeling a system with probability theory is to identify events that can happen in the system and model them as random variables. Moreover, the distribution of these random variables also needs to be estimated. The events in the real systems and hence the corresponding random variables can be dependent on each other. Bayes theorem provides a basis for modeling the dependency between the variables through the concept of conditional probability. The probability distribution of random variables could be conditioned on others. This makes modeling systems more elaborate but also more complex. Different modeling techniques have been developed to facilitate such a complex task.

Probabilistic graphical model are one family of such modeling techniques. A probabilistic graphical models aims to make modeling system events more comprehensible by representing independencies among random variables. A probabilistic graphical model is a graph in which each node is a random variable, and the missing edges between the nodes represent conditional independencies. Different families of graphical models have different graph structures. One well-known family of graphical networks, used in this research work, is Hidden Morkov Model Networks.

3.1 Hidden Morkov Model Networks

Hidden Morkov Model Networks (HMMN) is a special type of probabilistic graphical model. In a HMMN, like all graphical models, nodes represent random variables and arcs represent probabilistic dependency among those variables. The missing edges from the graph, hence, indicate that two variables are conditionally independent. Intuitively, two events (e.g., variables) are conditionally independent if knowing the value of some other variables makes the outcomes of those events independent. The conditional independence is a fundamental notion here because the idea behind the graphical models is to capture these independencies.

What differentiates a HMMN from other types of graphical models (such as Markov Nets) is that it is a Directed Acyclic Graph (DAG). That is, each edge has a direction and there should be no cycles in the graph. In a HMMN, in addition to the graph structure, the Conditional Probability Distribution (CPD) of each variable given its parent nodes should be specified. These probability distributions are often called the “parameters” of the model. The most common way of representing CPDs is using a table called Conditional Probability Distribution Table (CPT) for each variable (node). Each possible outcome of the variable forms a row, where each cell gives the conditional probability of observing that outcome, given a combination of the outcomes of the parents of the node. That is, these tables include the probabilities of outcomes of a variable given the values of its parents .The inference problem can get very hard in complex networks. Two types of inference, forward (causal) inference, an inference in which the observed variables are parent of the query nodes. The inference could be done backwards(diagnostic), from symptoms to causes. The inference algorithms typically perform both type of inference to propagate the probabilities from observed variables to the query variables. Researchers have studied the inference problem in depth. It is known that in general case the problem is NP-hard. Therefore,

researchers have sought different algorithms that perform better for special cases. For example, if the network is a polytree, inference algorithms exist that run in linear time with the size of the network. Also approximate algorithms have been proposed which use iterative sampling to estimate the probabilities. The sampling algorithms sometimes run faster but do not give the exact right answer. Their accuracy is dependent on the number of samples and iterations, a factor which in turn increases the running-time.

Designing a HMMN model is not a trivial task. There are two facets to modeling a HMMN, designing the structure and computing the parameters. Regarding the first issue, the first step is to identify the variables involved in the system. Then, the included and excluded edges should be determined. Here, the notions of conditional independence and casual relation can be of great help. It is important to make sure that conditionally independent variables are not connected to each other. One way to achieve that is to design based on causal relation: an edge from a node to another is added if and only if the former is a cause for the latter. For computing the parameters, expert knowledge, probabilistic estimations, and statistical learning can be used. The learning approach has gained much attention in the literature due to its automatic nature. Here, learning means using an observed history of variable values to automatically build the model (either parameters or the structure). There are numerous algorithms proposed to learn a HMMN based on history data, some of which are resented in.

One situation faced frequently when designing a HMMN is that one knows the conditional distribution of a variable given each of its parents separately, but does not have its distribution conditioned on all parents. In these situations, Noisy OR assumption can be helpful. The Noisy-OR assumption gives the interaction a graph with at most one undirected path between any two vertices. between the parents and the child a causal interpretation and assumes that all causes (parents) are independent of each other in terms of their influence on the child.

4. CONCLUSION

This paper presented a novel framework for regression testing of software using Hidden Morkov Model Networks (HMMN). The problem of software regression test optimization is targeted using a dynamic Bayesian network. The framework models regression fault detection and as a set of random variables that interact through conditional dependencies. In future Software measurement techniques are used to quantify those interactions and Hidden Morkov Model Networks are used to perform probabilistic inference on the distributions of those random variables. The inference gives the probability of each test case finding faults; this data can be then used to optimize the test suite for regression.

References

1. Amitabh Srivastava and Jay Thiagarajan, "Effectively Prioritizing Tests in Development Environment", In *Proceedings of the International Symposium on Software Testing and Analysis*, pages 97-106, 2002.
2. Bo Qu, Changhai Nie, Baowen Xu and Xiaofang Zhang, "Test Case Prioritization for Black Box Testing", 31st Annual
3. International Computer Software and Applications Conference (COMPSAC 2007), 2007.
4. David Leon and Andy Podgurski, "A Comparison of Coverage-Based and Distribution-Based Techniques for Filtering and Prioritizing Test Cases", *Proc. Int'l Symp. Software Reliability Eng.*, pp. 442-453, 2003.
5. Dennis Jeffrey and Neelam Gupta, "Test Case Prioritization Using Relevant Slices", In *Proceedings of the 30th Annual International Computer Software and Applications Conference*, Volume 01, 2006, pages 411-420, 2006.
6. Do.H, Rothermel.G, and Kinneer.A, "Empirical studies of test case prioritization in a JUnit testing environment", In Proc. Of 15th ISSRE, pages 113-124, 2004.
7. Elbaum.S, Mailshevsky. A.G., and Rothermel. G., "Prioritizing Test Cases for Regression Testing," *Proc. Int'l Symp. Software Testing and Analysis,* ACM Press**,** 2000, pp. 102–112.
8. Hema Srikanth and Laurie Williams, "Requirements-Based Test Case Prioritization", North Carolina State University, ACM SIGSOFT Software Engineering, pages 1-3, 2005.
9. Jung-Min Kim, Adam Porter and Gregg Rothermel, "An Empirical Study of Regression Test Application Frequency", ICSE2000, 2000.
10. Jung-Min Kim and Adam Porter, "A History-Based Test Prioritization Technique for Regression Testing in Resource Constrained Environments", In *Proceedings of the International*
11. *Conference on Software Engineering (ICSE)*, pages 119–129. ACM Press, 2002.
12. Kung, D., Suchak, N., Hsia, P., Toyoshima, Y., and Chen, C., " On object state testing", In *Proceedings of COMPSAC'94,* IEEE Computer Society Press, 1994.
13. Orso, A., Harrold, M. J., Rosenblum, D., Rothermel, G., Soffa, M. L., and Doo, H., "Using Component Metadata to support the regression testing of component-based software", In Proceedings of the International Conference on Software Maintenance (ICSM2001), pp 716-725, November, Florence, Italy, 2001.
14. Rothermel. G., Untch . R. H.Chu,.C and Harrold. M. J., "Test case prioritization: An empirical study", In *Proceedings ICSM 1999*, pages 179–188, Sept. 1999.
15. Rothermel.G et al., "On Test Suite Composition and Cost-Effective Regression Testing," *ACM Trans. Software Eng. and Methodology*, vol. 13, no. 3, 2004, pp. 277–331.
16. Shin Yoo and Mark Harman (2007), "Pareto efficient multi-objective test case selection ", proceeding of 2007 International symposium on software testing and analysis, ISBN 978-1-59593-734-6
17. Walcott.K.R, Soffa. M. L., Kapfhammer G. M. and Roos. R. S., "Time-Aware Test Suite Prioritization", In *Proceedings of the International Symposium on Software testing and Analysis*, pages 1-12, 2006.
18. Wong W.E. Horgan .J. R., London. S., and Agrawal. H., "A Study of Effective Regression Testing in Practice," *Proc. 8th Int'l Symp. Software Reliability Eng.,* 1998, pp. 264–274.
19. Xiaofang Zhang, Changhai Nie, Baowen Xu and Bo Qu, "Test Case Prioritization based on Varying Testing Requirement Priorities and Test Case Costs", *Proceedings of Seventh International Conference on Quality Software (QSIC'07)*, 2007.

Brief Bio-data of P.Thenmozhi

P.Thenmozhi has completed her M.Phil degree in Mother teresa women's University kodaikanal in 2004.She has completed 9 years of service in teaching. Currently she is Assistant Professor, Department of Computer Science, Kongu Arts and Science college, Tamilnadu, INDIA. She has guided 2 M.Phil students.She has presented 5 papers in various conferences.

Brief Bio-data of Dr.P.Balasubramanie

Dr.P.Balasubramanie has completed his M.Phil degree in Anna University Chennai in 1990. He has Qualified for National level eligibility test conducted by Council of Scientific and Industrial Research(CSIR) and Joined as a Junior Research Fellowship(JRF) in Anna University, Chennai. He has completed his Ph.D degree in Theoretical Computer Science in 1996. He has completed 15 years of service in teaching. Currently he is Professor, Department of Science & Engineering, Kongu Engineering College, Tamilnadu, INDIA. He is the recipient of Best Staff Award consecutively for two years in Kongu Engineering College. He is also the recipient of Cognizant-Technology Solutions(CTS) Best faculty award 2008 for outstanding performance. He has published more than 80 research articles in International and National Journals. He has authored 7 books with the reputed publishers. He has guided 6 part time Ph.D scholars and number of scholars is working under his guidance on various topics like image processing, data mining, networking and so on. He has organized several AICTE sponsored National seminar/ workshops.