Systematic Literature Review on Regression Test Prioritization Techniques

Yogesh Singh
Vice Chancellor, The Maharaja Sayajirao University of Baroda, Gujarat, India
E-mail: ys66@rediffmail.com

Arvinder Kaur, Bharti Suri and Shweta Singhal
University School of Information and Communication Technology, G.G.S.IP.University, Delhi, India
E-mail: arvinder70@gmail.com,bhartisuri@gmail.com, miss.shweta.singhal@gmail.com

Overview paper

Keywords: regression testing, test prioritization, systematic literature review (SLR)

Received: August 31, 2011

The purpose of regression testing is to validate the modified software and detect whether the unmodified code is adversely affected. Regression testing is primarily a maintenance activity. The main motivation behind this systematic review is to provide a ground for advancement of research in the field of Regression Test Prioritization. The existing techniques were compared along with their collected empirical evidences to find if any particular approach was superior to others. 65 papers reporting 50 experiments and 15 case studies were identified. A total of 106 techniques were evaluated for regression test prioritization. Also, a rigorous analysis of the techniques was performed by comparing them in terms of various measures like size of study, type of study, approach, input method, tool, metrics etc. Encouragingly, SLR yielded that almost half of the techniques for regression test prioritization are independent of their implementation language. While on the other hand the future research should focus on bridging the large gaps that were found existing in the usage of various tools and artifacts. During the course of research, preliminary literature survey indicated that to the best of our knowledge, no systematic review has been published so far on the topic of regression test prioritization.

Povzetek: V preglednem članku so opisane regresijske metode testiranja programske opreme.

1 Introduction

Regression test prioritization aims to prioritize the test cases that need to be re-executed during regression testing. The test cases are executed in that order so as to catch the faults at the earliest within minimum time. This is an important activity during maintenance phase as it rebuilds confidence in the correctness of the modified or updated system. This paper presents the systematic review of regression test prioritization techniques. Though a few of these techniques have been evaluated and compared by many researchers [1, 2, 3, 4, 5, 6, 7, 8, 9 etc], a generalized conclusion has not been drawn by any of them. In order to come up with a base for the advancement of future work in the field of Regression Test Prioritization (RTP), a systematic review was conducted to collect and compare some common parameters of the existing techniques and their empirical evidences.

There is a growing number of researches that are being carried out in the field of software engineering. Reviews are the essential tools by which a researcher can keep up with the new evidences in a particular area. There is a need to develop formal methods for systematic reviewing of the studies. In the last decade, the medical research field has successfully adopted the evidence based paradigm [10]. In [10], it is suggested that Evidence Based Software Engineering (EBSE) should be adopted. In [10], they have also discussed the possibility of EBSE using an analogy with the medical practices. EBSE is important as the software intensive systems are taking central place in our day to day life. EBSE can assist practitioners to adopt the appropriate technologies and to avoid the inappropriate ones. The goal of EBSE is "to provide the means by which the current best evidence from the research can be integrated with the practical experience and human values in the decision making process regarding the development and maintenance of a software" [10]. EBSE involves five basic steps [11]: 1) Convert the problem into an answerable question, 2) search the literature for the best available evidence, 3) critically appraise the evidence for its validity, impact, and applicability, 4) combining the critical appraisal with our environment and, 5) evaluating the efficiency of execution of the previous 4 steps and finding ways to improve them for future use. The first three steps constitute a systematic review. The systematic review is a specific research methodology that is aimed at gathering and evaluating the available evidences related to a focused topic area. They evaluate and interpret the relevant research that is available for the particular research questions or topic area [10].

The systematic review should consolidate the empirical studies conducted so far in the field. This review presents an overall report of all the existing regression test prioritization techniques presented till date, along with their properties and the comparisons among a few of them. It makes an attempt in displaying the amount of efforts already been put in to the field. To achieve the same, 65 test case prioritization papers were identified that reported 50 experiments, 15 case studies and 106 techniques of regression test prioritization. A qualitative analysis of the techniques was performed by comparing them with respect to the various measures like size of the study, type of the study, approach, input method, tool, and metrics etc.

2 Related Work

In a systematic review, the main research questions, the methodological steps, and the study retrieval strategies are explicitly defined. In 2004, the procedures for performing a Systematic Literature Review (SLR) in Software Engineering were first proposed by Kitchenham [12]. In the report [12], medical guidelines for performing systematic reviews were adapted to the requirements of software engineering. The first systematic review conducted in the field of software testing was on "the testing technique experiments" published in 2004 [13]. Staples and Niazi [14] shared their experiences while using the guidelines given by Kitchenham [12]. They emphasized more on the clearer and narrower choice of research questions and also on reporting the changes made in the strategy followed during SLR in order to adapt with the respective research scenarios. In addition to this, they [14] also found that reliability and quality assessment was difficult based on the given guidelines [12]. In-spite of these findings they [14] commend the same guidelines [12] to other researchers for performing SLR's. A systematic review in software engineering [15] presented all the systematic reviews conducted during Jan 2004-Jun 2007 in the field. Their SLR on 20 relevant found studies revealed that the topic areas covered by SLR's in software engineering are limited and that European researchers, especially the ones at Simula Labarotory [15] were the leading exponents of SLR's. Another systematic literature survey on regression test selection techniques was presented in 2009 [16]. 27 relevant studies were identified for the SLR[16] and evaluated quantitatively. According to the results obtained after relating various techniques to each other using empirical comparisons, Engström, Runeson and Skoglund [16], found that due to the dependence over varying factors no technique was clearly superior. Also, they identified a need for concept based evaluation of empirical studies rather than evaluations based on small variations in implementations. Engström and Runeson also presented a general industry based survey on regression testing practices in 2010 [17]. The survey was conducted for 15 industry participants and the outcomes were validated by 32 respondents via an online questionnaire. According to the authors [17], the practices were found not to be specific to regression testing and conclusion drawn was that regression testing should not be researched in isolation.

Furthermore, a very rigorous survey on regression test minimization, selection and prioritization was presented by Yoo and Harman [18]. Though it was not a systematic literature review, nonetheless it reported a detailed summary of the current state of art and trends in the field. The number of studies included in their study is almost the same as compared to the size of selected papers for the current research. This is reasonable as 1) their's was not an SLR, thus inclusion of every relevant study is not necessary; 2) the current SLR has been conducted including the studies that were published in the time slot of almost 2.5 years after their their survey was completed. An SLR should be very selective in the inclusion of a study with respect to its research questions. Thus, some of the studies included in the survey by Yoo and Harman for RTP area, got excluded at the study selection stage of our SLR. Also, there are a few additional studies found and included in this SLR that were published during and after the time frame for the survey in [18]. Nonetheless, Yoo and Harman have summed up the various approaches used for RTP, regression test minimization and selection along with the artifacts that have been used by these techniques. The same has also been repeated in this SLR to find whether their findings are correct or not. They had not reported the language dependency, granularity of the technique and the type of input to the technique. These aspects have been reported and used as a basis for the comparison of various techniques in the current research.

3 Difference between Literature Review and Systematic Literature Review (SLR)

Following the recent rise in the number of empirical studies in the field, SLR is a necessity for providing a thorough, unbiased and valuable summary of all the existing information. Systematic reviews require the documentation of not only the search criterions but also of the different databases that are searched. The starting point of a SLR is the review protocol that specifies the focused research question(s) to be addressed and the method to be employed in the process; while in the literature review the questions may be broad in scope. SLR employs a defined search strategy, and an inclusion/exclusion criterion for identifying the maximum possible relevant literature. Traditional review can be accomplished only by a single reviewer; while on the other hand, the systematic review requires a review team to establish the objectivity of literature classification at the very minimal level [19].

4 Research Method

This study presents a rigorous insight to various test case prioritization techniques developed and applied in regression testing area. Following the guidelines given by Kitchenham [12], the course of action undertaken for

this research has been presented in Fig.1. After being motivated for conducting this SLR, finalizing the research questions for the study was the first task to be completed. Once the research questions were reached, various databases were searched based on the search criteria to retrieve the relevant research in the area. The next and the most crucial step of the study was the selection of the most relevant papers based on various finalized parameters (discussed in section 3.3.2). After this step, 65 studies were finalized, and were rigorously examined to find the answers to our research questions. Their data extraction conforming to various parameters led to their empirical evaluation, comparison, appraisal etc., wherever possible And finally the conclusions were reached. The steps undertaken in the Systematic literature review for prioritization techniques are documented in detail in the following sections.

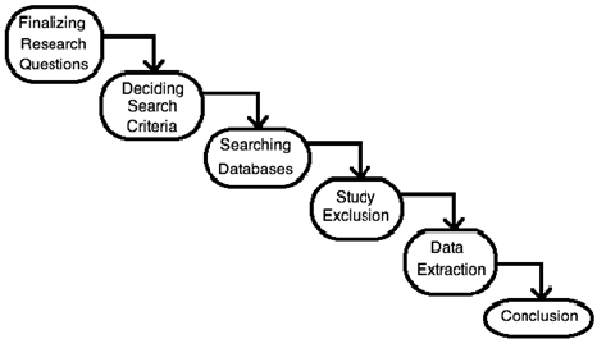

Figure 1: Course of action for this SLR.

4.1 Research questions

The aim is to summarize the current state of art in the RTP research by proposing answers to the set of the following questions:

RQ 1: What are the existing empirical evidences for various approaches followed by the RTP techniques?
RQ 2: Is it possible to prove the independence of various RTP techniques from their implementation languages?
RQ 3: What are the existing gaps in the current research regarding the use of tools, metrics and artifacts for various RTP techniques?
RQ 4: Can a RTP technique be shown superior to others based on a) the level of granularity followed, or b) the type of information used in prioritization?

4.2 Search Process

4.2.1 Sources of information

As suggested by Kitchenham in [19], searching databases gives more wider search space. In accordance with the guidelines, the following six databases were searched rather than the limited set of Journals and Conference proceedings to cover the maximum possible information.

- Inspec (digital-library.theiet.org)
- ACM digital library (dl.acm.org)
- IEEE eXplore (www.ieeexplore.ieee.org)
- Science Direct (www.sciencedirect.com)
- Springer LNCS (www.springerlink.com)
- Google scholar (scholar.google.com)

These electronic sources have been mentioned in [16, 17 and 19] as being relevant to the software engineers. There was an overlapping in the papers resulting from these sources and thus the duplicate papers were excluded manually.

4.2.2 Search Criteria

The initial search string was reached in order to find all the possibly relevant matter in the area of test case prioritization. Engström, Runeson and Skoglund [16] have already presented an SLR on regression test selection techniques. Their SLR is in a field much similar to our topic, thus the search string was reached considering the search string used by them [16] and the requirements for our topic. The keywords used were (((software) <or> (regression)) <and> ((testing) <or> (test)) <and> ((prioritisation) <or> (prioritization))). To make sure that all potentially related literature could be found, the above search string was applied on full text, rather than only on the title or the abstract. The start was set to January 1969 up till February 2011. The earliest paper included was published in the year 1997. Various searching standards are followed by different databases. Hence, the search strategy has to be designed accordingly. Some of the databases do not have the "and" option. In those, we had to search phrase by phrase. Search was carried out in 3 steps for such databases: 1) (software) <or> (regression) 2) (test) <or> (testing) 3) (prioritisation) <or> (prioritization). The search at 2nd step was carried out only on the results from the first step. Similarly, the 3rd step search was computed from the results from the 2nd step. The exclusion criteria during the search process also mentioned for the content **not** from books, standards, magazines, newsletters and educational courses.

4.2.3 Study Selection

The steps followed for the study selection procedure are as in Fig. 2. **Initially**, the study located 12,977 potentially relevant papers from all the sources mentioned in section 4.2.1. Elementary search yielded a huge amount of literature due to the use of the terms 'regression' and 'testing' in the search string. Databases could not differentiate between "statistical regression testing" and "software regression testing", and there exists a huge amount of literature on "statistical regression testing". Similar abundance in initial search results was observed in [16] when SLR was conducted on regression test selection techniques. In the next step, title based exclusions for papers irrelevant to the

software or regression testing were done. Although Dybå [20] has suggested to consider papers irrespective of their language, but we had to exclude the papers in any language other than English. After the **title based** exclusions, we were left with 634 studies.

Step 3 involved rejections based on the abstract for papers lying out of the search field. At this step, studies by both the students and the software professionals were included. The papers about general software testing, selection, reduction, test case generation and hybrid approach were rejected. Only those papers were included that dealt with prioritization. The number of the papers left after exclusions based on reading the **abstracts** were 213.

The final stage of the selection process was text based exclusions. At this stage, we made sure that each paper is selected only if has potential to contribute towards the answers of our research questions[21]. The papers presenting new technique(s) for prioritization, comparing the techniques, reviewing them or empirically validating them were included. The "lessons learned" papers, papers having pure discussion and expert opinion were excluded. Also, the studies included both qualitative and quantitative methods of research. Thus the final number of studies left after the exclusions based on the **full text** were 65 [1-9, 22-79]; these also formed the primary studies for our work (details listed in appendix A: Table A1).

A team of three researchers performed selection of the research papers individually at each stage. The papers were initially selected by two of the researchers that were then checked by the third team member. This process was repeated at each step of study selection (Fig.2). The conflict was mainly on the thoroughness of the works presented in the papers. And this was resolved by the opinion of the third and the fourth authors. Three papers were having conflict out of which two got selected as three authors agreed on the study being relevant while one was rejected. 49 primary studies out of the total 65 were found to report new technique(s), two were extension of the previous work and 14 were re-analyses of the previously reported studies. The same has been listed in Appendix A: Table A1.

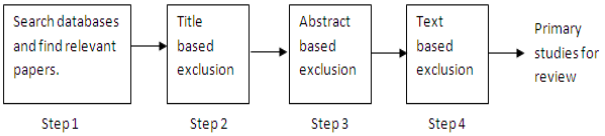

Figure 2: Steps followed in selection procedure for the study undertaken.

4.2.4 Data extraction strategy

The papers were thoroughly explored to find some common properties which formed the basis of the comparison. These were inspired from the previous work by Engström, Runeson and Skoglund [16] and also from the methods described by Cruzes and Dybå [21]. Each article was studied and appraised to detect the following:

(i) Technique description: The techniques were given the ID's and the names.
(ii) Artifacts used: The artifacts used in the study were noted.
(iii) Type of study: The type of the study can be an "experiment" or a "case study". It might also be possible that a study includes both the "experiment" and the "case study". An "experiment" is a study in which intervention is deliberately introduced to observe its effect [16]. A "case study" investigates within the real life context.
(iv) Comparison: Comparisons mentioned in the study, have been used to analyze and evaluate the studies.
(v) Language Type: It includes the type of the language on which the technique presented in the study is applicable. The language types found were: procedural, binary code, language independent, COTS component based, web designing or object oriented.
(vi) Input method: It includes the type of the input on which the technique can be applied. It can be: Source code, binary form, system model, system, call graph for program structure, or requirements/specifications.
(vii) Approach: The various approaches were found to be: modification based, coverage based, history based, requirement based, fault based, genetic based, composite or other approaches.
(viii) Granularity of approach: It specifies the granularity on which the technique can be applied. The 17 granularities followed in the papers are: Statement level, function level, block of binary form, method, transition in system model, system level, program, process level, event, component, file to be changed, software units, web service, module, configuration of the software system, class level or any. The above nomenclature was being followed by the studies. Some of the granularities seem to be same but they are separately mentioned, as it is not clear from the studies that they are at the same level.
(ix) Metrics: The metrics being used in a study are noted.
(x) Tools: Researchers have been using various tools during their study. The tools being used in each of the study were recorded.

5 Categories of Prioritization Techniques

Regression test prioritization re-orders the test cases so that those with the highest priority (according to some goal) are executed earlier in the regression testing process than the lower priority test cases. To better understand the progress of research in the field of regression test prioritization, eight broad categories were identified. Classification has been made on the basis of the approach followed for prioritization. The discussion presented in the following sections (4.1 – 4.10) also provides an answer to RQ2 by specifying the compared techniques.

5.1 Coverage Based (CB) Approach

Coverage based prioritization is based on the fact that more the coverage achieved by the test suite, more are the chances of revealing the faults earlier in the testing process. Wong et al. [22] initially included prioritization in a hybrid technique. They prioritized the test cases according to the criterion of increasing the cost per additional coverage.

In 1999, Rothermel et al. [23] proposed four coverage based techniques: total/additional statement/branch coverage respectively. The statement level granularity was followed based on source code type of input method. Aristotle program analysis system tool was used for the comparison and the results were measured using Efficacy and APFD metrics. The ordering of the test suite was compared with respect to the faster detection ability of catching faults. On comparing the techniques, Rothermel et al. found that the total coverage prioritization outperforms the additional coverage prioritization.

This work was taken a step further by Elbaum et al. [24] to address the version specific prioritization. Eight techniques were proposed out of which the "total function" and the "additional function" were based on coverage. Rate of fault detection improved by using the version specific test case prioritization. Comparisons among 12 techniques (4 statement level and 8 function level) yielded the worst and the best results for fn-total (function-total) and fn-fi-fep-addtl (function-fault existence/exposure-additional) techniques respectively. A tradeoff was established between the statement and the function level techniques. On one hand the function level techniques were found to be more cost effective and involved less intrusive instrumentation while on the other hand the statement level techniques were preferred if sufficiently high cost of delays are observed in the detection of faults.

Srivastava and Thigarajan [25] introduced the binary code based prioritization approach. A test prioritization system Echelon was built that prioritizes the set of faults based on the changes made to the program. The suggested advantage of the binary form is the elimination of recompilation step for coverage collection etc. making the integration of build process easier in the production environment. The presented case study showed that it is possible to effectively prioritize the test cases using binary matching in a large-scale software development environment.

Do et al. [26] performed a controlled experiment to examine the effectiveness of test case prioritization on programs tested under JUnit. Six block and method level granularity techniques were proposed: Total block coverage, Additional block coverage, Total method coverage, Additional method coverage, Total diff method and Additional diff method. Diff method techniques use modification information. These techniques are for JUnit environment and correspond to the already proposed techniques focusing on C language in [23, 24, 27]. The inference drawn from the comparison was that the level of granularity and the modification information had no effect on the prioritization. The techniques using feedback information (Additional techniques) provided significant improvement in the fault detection rate. On comparing with the previous studies on C, the statement level techniques were found to be better than the function level techniques. Possible reason for this as analyzed by [26] was that the instrumentation granularity for Java differs from C.

Bryce and Memon [25] proposed five new testing techniques for software interaction testing of Event-driven software. The techniques include: interaction coverage based prioritization by length of test (longest to shortest), 3-way interaction, 2-way interaction, unique event coverage and length of test (shortest to longest). The comparison within the proposed five and the random technique resulted in the following findings: test suites including largest percentage of 2-way and 3-way interaction have the fastest fault detection rate; the proposed techniques are useful for test suites having higher interaction coverage.

A graph model based prioritization using fuzzy clustering approach was proposed by Belli et al. [29] in 2006. The paper presented a case study of graph model based approach on the web-based system ISELTA. The complexity of the method has been given as $O(n^2)$. The approach was found to be useful when test suites are ordered within restricted time and method.

The effects of time constraint on the cost benefits of regression testing were studied by Do. et al. [30] by offering four techniques out of which two were based on total/additional coverage and two on Bayesian network approach (discussed in section 4.8.3). Additional technique was found to be more efficient than total technique.

Jiang et al. [31] proposed nine new coverage based Adaptive Random Test (ART) Case Prioritization techniques in 2009. These techniques were broadly classified into three groups namely maxmin, maxavg, and maxmax. For each group the level of coverage information was based on statement, function and branch. The comparison within the proposed techniques and random ordering resulted in the following findings: ART techniques are more effective than random ordering; ART –br-maximum (br-branch) technique is the best among the entire function group of ART techniques; it is more practical and statistically effective than the traditional coverage based prioritization techniques revealing failures.

Maia et al. [32] proposed the use of Reactive GRASP (Greedy Randomized Adaptive Search Procedures) metaheuristics for prioritizing the test cases. The technique uses block, decision and statement coverage criteria. The results were compared to the search algorithms like greedy, additional greedy, genetic and simulated annealing techniques. They found that the proposed technique significantly outperformed genetic, simulated annealing and greedy algorithm. Also, the performance was not worse than the additional greedy algorithm. Proposed solution exhibited more stable behaviour as compared to other solutions.

In 2009, a multilevel coverage model based family of prioritization techniques was proposed by Mei et al. [33] to capture the business process. Mei et al. defined three data coverage levels as CM-1, 2 and 3 where CM implies Coverage Model. Ten proposed techniques (M1 to M10) include: M1: Total-CM1, M2: Addtl-CM1, M3: Total-CM2-Sum, M4:Addtl-CM2-Sum, M5:Total-CM2-Refine, M6:Addtl-CM2-Refine, M7:Total-CM3-Sum, M8:Addtl-CM3-Sum, M9:Total-CM3-Refine, and M10:Addtl-CM3-Refine. They also gave a hierarchy of the proposed techniques to analyze their effectiveness. Except the optimal technique, M6 and M7-M10 were found to be generally better and M1 was found to be the worst among all other techniques. Recently in 2010, Mei et al. [34] also proposed four black box testing techniques for service oriented application in which the regression test cases were reordered using WSDL (Web Service Description Language) information. The techniques comprise: Ascending/Descending WSDL tag coverage prioritization, Ascending/descending WSDL tag occurrence prioritization. Contrasting these four black box techniques with two benchmark (random and optimal), two traditional (Total and Additional-Activity) and two white box (Total and Additional-Transition) prioritization techniques they computed APFD, Boxplots and performed ANOVA analysis. They derived the following outcomes: Black box testing techniques are better than the random ordering in terms of the overall mean APFD. Moreover, white box testing techniques required source code for the services under test while the black box needs only interactive messages. In analogy to traditional functional prioritization techniques, black box testing techniques were able to achieve coverage based on tags. Also, the black box testing techniques achieved higher fault detection rates.

The latest study for the coverage based approach for developing a single abstract model by combining the GUI and the web applications for testing was published in 2011 by Bryce et al. [35]. The prioritization has been accomplished based on Parameter-Value Interaction Coverage, Count, or Frequency criterion. The generic prioritization criterion for both the GUI and the web applications was also defined. The comparisons concluded that both the applications showed similar behaviour when re-casted using the new model. The usefulness of the combined model for two types of event-driven software was indicated by the empirical study.

5.2 Modification Based (MF) Approach

This approach aims to prioritize the test cases based on the modifications made to the program. As already mentioned in the previous sections, the initial paper discussing prioritization was using modification-based approach and was authored by Wong et al. [22]. In 2005, Korel et al. [37] proposed System model based selective test prioritization and Model dependence based test prioritization techniques using Extended Finite State Machine (EFSM) system models. Although the later technique was a little expensive, improvement in prioritization effectiveness was observed using rate of fault detection metrics for both the techniques. Korel et al. [36] proposed five more heuristic based techniques and compared all the seven techniques in 2007. Model dependence based technique and a heuristic technique based on high priority assignment to test cases executing transition that execute least number of times, exhibited best effectiveness out of all the seven techniques. The later is significantly simpler and requires less information about the models than the former.

A model based prioritization approach for selection of test cases relying on traceability links between models, test cases and code artifacts was given by Filho et. al. in 2010 [38]. This technique supports the change based regression testing using timestamps and property based prioritization. They performed the prioritization and the filtering as a part of the process of test generation using test suite modifiers.

5.3 Fault Based (FB) Approach

Fault based prioritization techniques have been proposed initially by Rothermel et al. in [23]. According to it, the ability of a fault to be exposed by a test case not only depends whether a test case executes a particular statement but also on the probability that the fault in the statement will cause failure for that test case. Two techniques (Total fault exposing potential (FEP) and Additional-FEP prioritization) with respect to the fault exposing potential of a test case have been presented in the study. The study also proposed four coverage-based techniques as discussed in the earlier section. Additional – FEP outperformed all the proposed coverage based technique. Total FEP outperformed the same except total branch coverage prioritization. The results shown using Efficacy and APFD suggested that these techniques can improve the fault detection rate and that the results occurred even for the least expensive techniques.

Elbaum et al. presented six function level techniques for prioritizing test cases with respect to faults [24]. Two of the techniques are function level based fault exposing potential (FEP) prioritization; other two are based on fault index that represents fault proneness for that function and two more combine both fault index and fault exposing potential by initially applying total fault index prioritization to all test cases and then applying FEP prioritization to that possessing equal fault index value as secondary ordering. Two more coverage-based techniques presented in the paper have been discussed in the coverage section. Enough statistical evidence has been provided to show that the function level techniques are less effective than the statement level techniques. Fault proneness and FEP estimators have not been found to significantly improve the power of prioritization techniques.

In addition to the above techniques, four function-level prioritization techniques were also proposed by the same authors [27]. The techniques are DIFF-based techniques. These techniques require the computation of syntactic differences between two versions of the program. The degree of change is measured for each of the function present in both the versions by adding the

number of lines inserted, deleted or changed in the output of UNIX diff command applied to both the versions. Two of these four techniques are based only on DIFF and other two combine DIFF with FEP (Fault Exposing Potential). They have compared 18 techniques: two reference techniques (optimal and random), four statement-level and twelve functional-level techniques [23, 24, 27]. Statement level-additional FEP technique performed the best after the optimal. The second best were the function level techniques combining fault proneness measures and FEP. Additional techniques were found to be better than total techniques. Also, the statement level techniques were better than function level technique. Finally, the techniques combining FEP and fault index were better than the rest.

5.4 Requirement Based (RQ) Approach

Srikant et al. [39, 40] proposed a system level technique PORT V 1.0 (Prioritization Of Requirements for Testing) for prioritization based on the requirements and developed a tool to implement the same. The value-driven approach is based on four factors: customer assigned priority of requirements, developer-perceived implementation complexity, requirement volatility and fault proneness of the requirements. The objective is to reveal the severe faults earlier and to improve the customer-perceived software quality. Higher severity faults were mapped with the requirements with higher range of PFV where PFV is the prioritization factor value for a particular requirement computed using their formula. The study showed that the PORT technique could improve the testing efficiency by focusing on the customer's highest value functionality and on improving the severe fault detection rate thereby minimizing field fault occurrence.

Quota constrained strategies (Total and Additional) to maximize the testing requirement coverage were proposed for a service-centric system in [41] by Hou et al. The aim is to maximize the total or the additional testing requirement coverage. It selects a subset of test cases that can satisfy the constraint imposed by the request quotas over a period of time. The comparison of Quota strategies with branch coverage approaches lead to the outcome that the Quota constraint strategies provided better branch coverage.

A model for system level test case prioritization from the software requirement specification was presented to improve user satisfaction and the rate of severe fault detection in [42]. The model prioritized the system test cases based on the following six factors: customer priority, changes in requirement, implementation complexity, usability, application flow and fault impact. Another technique by the same authors has been presented in [43] which only differs in two of the factors affecting the prioritization algorithm. The factors presented in [43] are: customer assigned priority, developer perceived code implementation complexity, changes in requirements, fault impact, completeness and traceability. On comparing the techniques with the total statement and the total method coverage, the rate of detection of severe faults was found to be higher for their technique.

5.5 History Based (HB) Approach

Kim and Porter proposed the first history-based prioritization technique in 2002 [44]. The prioritization performed in the technique is based on the historical execution data. They show that the historical information may be useful in reducing costs and increasing the effectiveness of the regression testing process. The notion of memory full regression testing was incorporated in [44]. The weakness of this approach is that only the effect of last execution of the test cases, especially in the binary manner, is used to calculate the selection probability of test cases. Evaluations yielded that regression testing may have to be done differently in the constrained environments than the non-constrained one. Also, the historical information may be useful in reducing the cost and increasing the effectiveness of a lengthy regression testing process.

A historical value based approach using the historical information to estimate the current cost and the fault severity for cost-cognizant test case prioritization is presented by Park et al. in [45]. It uses the function level gratuity and the historical information of the cost of the test cases and the fault severities of detected defects in a test suite to calculate the historical value of the test case. This value is then used for test case prioritization. In analogy with functional coverage prioritization technique, the technique produced better results in terms of APFDc metric.

Fazlalizadeh et al. [46] modified the history based prioritization technique proposed by Kim and Porter [44] to give faster fault detection in the resource and time constrained environments. The paper presented a new equation that considers the historical effectiveness of the test cases in fault detections, test case's execution history and last priority assigned to the test cases. The proposed technique was compared to random ordering and boxplots were used to visualize the empirical results confirming faster fault detection and stability.

5.6 Genetic Based (GB) Approach

A time aware prioritization technique practicing genetic approach was proposed by Walcott et al. in 2006 [47]. The experiment was conducted at program level granularity on two subjects: Gradebook and JDepend. Emma and Linux process tracking tool were operated on and the results were quantified using the APFD metric. Eventually, GA prioritization realized improvement over no ordering (by 120%), reverse ordering and fault aware prioritization.

Another Genetic Algorithm (GA) based test suite test case prioritization was proffered by Conrad et al. in 2010 [48]. The paper presented a wide variety of mutation, crossover, selection and transformation operator that were used to reorder the test suite. An experimental study was implemented on 8 case study applications (same as in [49]), using same coverage effectiveness metric [49] and their JUnit test cases at system level. The results

were analyzed with the help of beanplots. On comparison of the proposed technique with random search and hill climbing techniques, GA yielded finer results. Also GA was found to have similar execution times as that of random search and hill climbing. All in all, GA showed a greater variability and is also an upcoming area of research in the field.

5.7 Composite (CP) Approaches

The techniques using two or more of the above (4.1-4.6) and other (4.8) approaches have been categorized under the composite approach.

5.7.1 CB+MF

The introductory study that identified prioritization for regression testing was reported by Wong et al. [22]. They combined modification and coverage approach for their hybrid technique (modification, minimization and prioritization). Though the technique is applied on statement level granularity, it can also be implemented for function level and low level granularity. A combination of modification and minimization was compared with the combination of modification and prioritization techniques. Both were found to serve as a cost effective alternative for faster regression testing in a time and cost constrained environment. The cost effectiveness of techniques was measured using size reduction, recall and precision metrics.

A case study based on the technique incorporating aspects of modification and decision coverage was conducted by Jones and Harrold [50]. The empirical study revealed that the technique significantly reduced the cost of regression testing.

The use of particle swarm optimization (PSO) algorithm for automatic prioritization of test cases based on the modified software units and fitness of the test coverage was proposed in 2008 by Hla, Choi and Park [51]. The total prioritization cost using PSO algorithm was computed to be $O((m*p)kn) < O(mn^2)$. Comparing with the random technique they found that 64% coverage could be achieved against only 47% achieved by the random technique.

5.7.2 CST+FB

Cost-cognizant test case prioritization techniques based on the cost and fault severity were presented by Malishevsky et al. in 2006 [52]. The author adapted and compared their already suggested function level techniques [24, 27] namely fn_total, fn_addtl, fn_diff_total, fn_diff_addtl to the cost cognizant framework. The complexity of the cost cognizant total algorithms was found to be $O(n.m + nlogn)$ while that of additional algorithms was $O(n^2m)$ where n is the size of test suite and m is the number of functions in the system. The proposed techniques were found to be effective only in some of the cases.

5.7.3 MF +SLC

A statement level slice based heuristic combining REG (regular statement/branch) executed by test case, OI (output influencing) and POI (potential OI) was expressed in an experimental study conducted by Jeffery and Gupta [53]. Aristotle Program Analysis tool was used to compare the technique with total statement and branch coverage. It was interpreted that faults were detected earlier in the testing process from the fact that the information about relevant slicing and modifications traversed by each test case is beneficial when used as a part of test case prioritization process.

5.7.4 MF+CB+FB

Mirarab et al. proposed a test case prioritization technique based on Bayesian networks in 2007 [54]. The demonstrated technique is a mixture of three approaches namely modification, fault and coverage based. A comparison was performed among ten prioritization techniques that included three control techniques (Original, Random and Optimal) and six total/additional techniques based on class, method and change overage and the introduced technique. It was observed that all the techniques performed better than random order and original order and that, as the number of faults grew Bayesian network yielded promising results. In 2008, the aforementioned authors presented an enhanced Bayesian networks approach [55]. The technique introduced a new feedback mechanism and a new change information gathering strategy. The results derived from APFD have showed the advantage of using feedback mechanism for some objects in terms of early fault detection.

5.7.5 RQ+HB

A novel prioritization technique for black box testing was brought up by Qu et al. [56]. It is requirement based prioritization approach for which test history and run time information were used as the input method. Moreover, the technique was compared with the random ordering suggesting that the technique improved the test suite's fault detection rate.

5.7.6 CB+IB

A prioritization technique "Combinatorial Interaction Regression Testing (CIT)" combining coverage and interaction approaches has been suggested by Qu et al. [57]. NAPFD metric is used to compare CIT technique with re-generation/prioritization technique where re-generation prioritization techniques are the techniques that are combination of generation and prioritization using interaction testing [58]. The outcome shows that prioritized and re-generated /prioritized CIT test suites were able to find faults prior to unordered CIT test suite.

5.7.7 RQ+CST

Two techniques "total" and "additional" combining "testing requirement priorities" and "test case cost" were set forth by Zhang et al. [59]. They worked on the simulation experiments to empirically compare 24

combinations of the requirement priorities, test cost and test case prioritization techniques. The techniques were compared with the unordered test suite and "additional" technique performed the best among the three. An original metric to evaluate the effectiveness of prioritization based on "units of testing requirement priority satisfied per unit test case cost" was realized.

5.7.8 MF+SVD

A methodology based on Singular value decomposition (SVD) with empirical change records was introduced by Sherriff et al. [60]. The case study compared the presented technique and the regression test selection (RTS) technique [61] with respect to inclusiveness, efficiency and generality. It turned out that the technique was more efficient than the RTS techniques provided the traceability information is readily available.

5.7.9 CB+MF+SLC

Jeffrey and Gupta [62] advanced their earlier proposed technique [53] by adding coverage requirements of the relevant slices to the modification information for prioritization. The two techniques derived from the original technique "REG+OI+POI" [50], were named as "GRP_REG+OI+PI" and "MOD*(REG+OI+PI)". In comparison with the statement and branch coverage techniques, the extended MOD*(REG+OI+POI) proved to be an improvement over the REG approach on the grounds of the fault detection rate of prioritized test suites.

5.7.10 CB+MF+FB+PS

A prioritization technique by Ma and Zhao [63] based on coverage, modification, fault and program structure was presented and compared with four other techniques: total and additional method coverage, total and additional different method coverage. It came forth that the technique performed better than original, random, total method coverage, additional method coverage, total different method coverage and additional different method coverage by 30%, 62%, 13%, 11%, 31% and 24% respectively.

5.7.11 CF+DF+CB+MF

Chen et al. [64] reported a test case prioritization technique in 2010 for the web service regression testing using WS-BPEL language. The paper presented a case study of an ATM example and a weighted graph was constructed that help to identify modification affected elements using impact analysis. The study was based on combination of four approaches: control flow, data flow, coverage and modification. Two techniques that were used to prioritize test cases included total and additional techniques. The main goal of prioritization was to cover the elements with the highest weight. The approach gave appropriate reasons for fake dependence in BPEL process and also gave solutions for their elimination.

5.7.12 HB+GA+CST

A cost-cognizant technique utilizing the historical records and the genetic algorithms to carry out the prioritization process was instigated by Huang et al. in 2010 [65]. A combination of three approaches (history, genetic and cost based) was used by the version specific test case prioritization technique. GA_hist was compared with a genetic based [47], two history based [45], a cost cognizant based, a function coverage based, random and optimal techniques. The results highlight greater mean APFDc value for the GA_hist than other techniques. It was also revealed that the proposed technique improved the effectiveness of cost-cognizant test case prioritization without taking into account the source code, test case cost and uniformity of the fault severities. The greater the number of generations, more effective is the proposed technique.

5.8 Other (O) Approaches

The approaches for which only single technique was available in the literature have been listed in the 'Other' category.

5.8.1 Data flow based (DF)

Rummel et al. [66] proposed a data flow based prioritization technique in 2005. It is based on the definition and use of the program variables by employing the all-DU's test adequacy criteria. The discussed technique was compared with the random ordering. It was found that the time and space overhead increase with the size of the application. Also, it was concluded that the test suites can be prioritized according to the all-DU's with minimal time and space overheads. Finally, the data flow based prioritization were not found to be always effective than the random order.

5.8.2 Inter Component Behaviour (ICB)

In 2007, Mariani et al. [67] gave a new technique to prioritize the test cases that provided an improvement of the efficiency of the regression testing of the COTS components. The proposed techniques followed inter component behaviour approach. The technique helped in discovering many faults after the execution of a small amount of high priority test cases. It was also observed that less than 10% of the high priority test cases revealed all the faults for all the considered configurations except in one of the configurations.

5.8.3 Bayesian Network Approach (BN)

Two class level Bayesian network based techniques were described by Do et al. [30] in addition to the two coverage based techniques (discussed under CB approaches). The effectiveness of the block level and the class level techniques were contrasted against the original and the random ordering. It emerged that the effect of time constraint on differences between the cost benefits increased as the time constraint level increased. As mentioned earlier, feedback techniques (additional) were found to be more effective than their non-feedback

counterparts. Overall, it was found that the BN techniques tended to have lower cost on an average than the coverage based techniques.

5.8.4 Cost Based Approach (CST)

A prioritization technique for Multiple Processing Queues applying task scheduling methods was proposed by Qu et al. [68]. The technique was compared with the random approach providing an improvement in parallel testing scenario with respect to the fault detection.

5.8.5 Graph based Approach (GPH)

Ramanathan et al. presented a graph based test case prioritization in 2008 [69]. A weighted graph was constructed in which the test cases denoted the nodes and the edges specified user defined proximity measures between the test cases. The de clustered linearization of nodes in the graph led to the prioritization of the test cases. Fielder (spectral) and greedy ordering approaches were used and were implemented using PHALANX framework.

5.8.6 Configuration Aware Approach (CA)

A paper addressing the issue of providing configuration aware regression testing for evolving software was presented by Qu et al. [70]. A combinatorial interaction testing technique was used to generate the configuration samples that were used in the regression testing. The comparison highlighted that the median fault finding ability and NAPFD of the technique is higher than original ordering and has better fault detection capability than random ordering.

5.8.7 Classification Tree Based Approach

Yu et al. [71] proposed an annotated classification tree based prioritization technique in 2003. The annotation to the classification tree is made with additional information of selector expression, occurrence tags and weight tags. The annotated classification tree was used to prepare prioritized test suite and this process was automated using EXTRACT (Extracting black boX Test cases fRom Annotated Classification Tree).

5.8.8 Knapsack Based Approach (KB)

Knapsack solvers were exploited in the time aware prioritization by Alspaugh et al. in 2007 [72]. The test suites were prioritized using seven algorithms: Random, Greedy by ratio, Greedy by value, Greedy by weight, Dynamic Programming, Generalized tabular and Core. The effectiveness of each of the algorithm to prioritize was measured using code coverage, coverage preservation and order-aware coverage metrics. The comparisons revealed that Dynamic programming, Generalized tabular and Core do not always create more effective prioritization. Moreover, if correctness had utmost importance, overlap prioritizers with higher time overhead were found to be appropriate.

5.8.9 Failure Pursuit Sampling (FPS)

Simons et al. [73] proposed a distribution based prioritization technique called Failure Pursuit Sampling that was previously used for prioritization of tests in general [5]. The original technique was modified by improving the clustering and the sampling phases of FPS using the fault matrix computed from the execution of test on the previous versions. It was accrued that the technique has higher rate of efficiency than the original FPS.

5.8.10 Search Algorithm based (SA)

Search algorithms have been used as the basis for prioritization technique or comparisons. Some of the studies [32, 48, 65, 72] using the search algorithms have been discussed in the previous sections as they followed genetic, composite or other approaches. The papers exclusively based on search algorithms have been discussed here. All the recorded search algorithm for RTP have been summarized in Appendix A. (Table A3).

Li et al. [74] applied five search algorithms (Hill climbing, Genetic algorithm, greedy, additional greedy and 2-optimal greedy) to prioritization and compared them by empirical evaluation. Greedy algorithms enhance the initially empty test suite incrementally using some heuristics. The greedy algorithms are also compared with respect to their cost of prioritization. If m is the number of statements and n is the number of test cases, the cost of prioritization for greedy, additional greedy and 2-optimal greedy was found to be O(mn), $O(mn^2)$ and $O(mn^3)$ respectively. The results exhibited that Additional Greedy and 2-Optimal were the finest and along with Genetic Algorithm, these 3 always outperformed the Greedy Algorithm.

An extension and empirical evaluation of greedy algorithm, 2-optimal greedy algorithm and delayed greedy algorithms was presented by Smith and Kapfhammer in 2009 [49]. They incorporate the test case cost, the test coverage and the ratio of coverage to cost in the algorithm. For each of the eight observed case studies, a decrease in the testing time and the coverage of the test requirements was observed.

Lately in 2010, Sihan Li and his teammates [75] performed a simulation experiment for studying the same [74] five search algorithms for RTP. The test requirements based on statement, decision, block and other coverage criteria were measured. The results concluded that the Additional and the 2-Optimal greedy algorithm performed better in most of the cases, which is in conformance to the results of the previous study. Also, the overlap of test cases affected the performance of these algorithms with respect to the test requirements.

5.9 Comparison Studies

Elbaum et al. in 2001 [1] proposed a new cost cognizant metric APFDc (adapted from APFD) that was used for measuring the rate of fault detection and included varying test cases and fault costs. A case study was performed to analyze the impact of test cost and the fault

severity of the prioritization techniques (random, additional statement coverage, additional function coverage and additional fault index). The additional fault index prioritization resulted better than the other techniques. All the four techniques were found to be better than the random technique.

In addition to the above three techniques, three more techniques (Total statement/ function coverage and fault index) and optimal (instead of random) techniques were analyzed in terms of APFD (initially explained in [20]) by Elbaum et al. [2]. The task was accomplished by exploring the impact of certain factors of the various prioritization techniques on the fault detection rate. The conclusion drawn by them was that a new technique incorporating information provided by the metric APFD can be developed.

Nine techniques were described and compared by Rothermel et al. in 2001 [3]. The techniques were: original order; random order; optimal; total/additional statement coverage; total/additional branch coverage; total/additional fault exposing potential prioritization. The results showed that all the techniques performed better than the original and the random order prioritization. Also, the additional fault exposing potential prioritization performed the best. Moreover, the branch coverage techniques were better than the corresponding statement coverage techniques.

Elbaum et al. [4] examined two techniques, total/additional function coverage along with the random and the optimal ordering to understand the effect of change on the cost effectiveness of the regression testing techniques. They made use of a large number of measures to accomplish the comparative case study. The analysis found that the change attributes played a significant role in the performance of the techniques. Also, the additional function coverage technique outperformed the total function prioritization technique regardless of the change characteristics. The total technique gave varied results and was sometimes worse than random prioritization.

An empirical comparison among four different prioritization techniques was put forward by Leon et al. in 2003 [5]. These techniques included test suite minimization, prioritization by additional coverage, cluster filtering and failure pursuit sampling (FPS). The former two techniques were broadly classified as coverage based and the latter two as distribution based. The comparisons yielded the following findings: when the sample sizes are small, basic coverage maximization can detect the facts efficiently; one per cluster sampling achieves comparably good results and at the same time does not achieve full coverage; for large sampling sizes, FPS is more efficient than cluster sampling. APFD demonstrated that the random ordering outperformed the repeated coverage maximization for GCC while not for Jikes and Jvac. The results also suggested that both the coverage based and the distribution based techniques were complimentary in finding different defects.

Rothermel and Elbaum [6] experimented and studied the effect of test suite granularity and test input grouping on the cost and the benefit of regression testing methodologies. An analogy was established among the three prioritization techniques: optimal, additional and additional-modified function coverage prioritization. It revealed that the test suite granularity affected several cost-benefit factors for the methodologies and at the same time the test input grouping had limited effect. As the granularity level decreased, higher APFD values were observed. It emerged that the finer granularity precisely discriminates between the test cases. The results were recorded to be consistent with [27].

Elbaum et al. [7] thoroughly analyzed the fault detection rates of five prioritization techniques (random order, total/additional function coverage prioritization; total/ additional binary diff. function coverage prioritization) on several programs and their versions to help the practitioners chose a technique for a particular scenario. The generalized results showed that the techniques using feedback gave better results. They suggested that since the performance of the technique varied significantly with the scenarios (programs, test cases and modifications), it was therefore necessary to choose the appropriate technique. They also stressed that choosing a technique with higher APFD is oversimplifying and may not always imply a better technique. The two strategies proposed by them for the practitioners include: Basic instance-and-threshold strategy (to choose a technique that is successful for largest number of times) and Enhanced instance-and-threshold strategy (that adds attribute of the scenario using metric and then selecting the technique by building classification tree). The results suggested, like many others, that the techniques using feedback were better.

A small experimental study was performed for comparing the simple code based and the model based test prioritization method with respect to the early fault detection effectiveness in the modified system by Korel et al. [76]. The study focused on the source code faults. The results expressed that the model based test prioritization may improve the average effectiveness of early fault detection significantly when compared to code-based prioritization. The model based prioritization was less expensive but was sensitive to the information provided by the tester or the developer.

Block and method level prioritization techniques for the total and the additional coverage were assessed using the mutation faults by Do and Rothermel in 2005 [8]. They also examined the consistency of the results with the prior study [26] of Java System using hand seeded faults. The levels of coverage had no effect on the rate of fault detection whereas the additional techniques proved better over the total techniques.

The same authors along with Kinner [9] empirically performed the cost benefit analysis on the same artifacts. The comparisons were accomplished on the same techniques as mentioned above and also the method_diff total and the additional techniques. They found that the functions and the statement level in C correspond to the method and the block level in Java respectively. It hailed from the experiment that the statement level techniques were superior to the function level in C. But the block

level techniques were not found to be very different from the method level techniques in Java. This is because the block level is not as sensitive as the statement level. The cost benefit analysis also revealed that the method and the block level additional techniques resulted in the highest cost savings.

Do and Rothermel [77] further conducted an empirical assessment of the same techniques as in [8]. Same results with respect to the level of coverage were recorded. Due to large sampling errors produced using the mutation faults, they were found to have better rate of fault detection over the hand seeded faults.

Aforementioned authors [78] also put forth an improved Cost-Benefit model incorporating the context and the life-time factors and compared two prioritization techniques (total/additional block coverage) and two regression test selection techniques. Time constraint proved an important factor for the relative benefits hailing from the tradeoff between the cost of the additional tests without missing the faults and the cost of reduced testing missing the faults.

A series of controlled experiments was conducted by Do et al. in 2010 [79]. These were used to assess the effects of time constraint on the cost and the benefits of six prioritization techniques. The techniques included two control (random/original) and four heuristic techniques (two feedback and two non-feedback). The results showed that heuristic techniques(Bayesian network based and conventional code coverage based) were useful when no time constraint were applied and the software contained a large number of faults. The results also revealed the cost effectiveness of the feedback (additional) prioritization techniques over their non-feedback counterparts. In addition, the feedback techniques again performed unvaryingly better with the increase in the time constraint levels.

6 Results and Analysis

The study resulted in the selection of 65 RTP research papers for the literature survey. 106 new prioritization techniques were identified from 49 of the studies, whereas rest 16 studies were based only on the comparative analysis.

Publication trends (Fig. 3) were observed from 1969 till the search of studies for the survey was carried out (Feb 2011). The first technique (composite) was recorded in the year 1997. Over the years, many more techniques were logged and an increasing publication trend has been observed. Maximum number of published papers appeared in 2007 and 2008 (11each) accounting for 35 of the techniques. Though most of the techniques were documented in 2009 (21), the number of studies were only six. This is due to the fact that many studies presented more than one technique in the same paper.

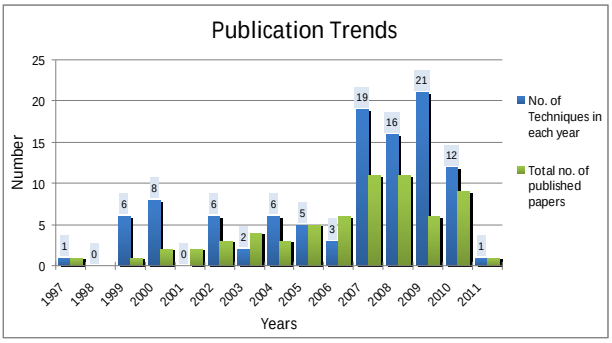

Figure 3: Publication trends.

6.1 Advent and usage of approaches (RQ1)

The techniques were broadly categorized under eight approaches as already discussed. The advent of these approaches has been illustrated graphically in the Fig. 4. The height of the bars in fig.4 represents the recentness of the use of the particular approach to regression test prioritization. The year 1999 saw the advent of the Fault based approach for prioritization. It also proved to be the main motivation behind many evaluation measures such as APFD etc. After these, the year 2002 experienced the use of the feedback (History based) approach for test suite prioritization [44]. Generally, the errors are concentrated in the primary stages of the software development process. Realizing this fact, Requirement based techniques emerged for the first time in 2005 [40]. Genetic algorithms based techniques are an upcoming approach documented primarily in 2006 [47] for the use in prioritization. The approaches introduced after 2006 have been included in the 'Others' category along with the approaches that have not been used more than once in the RTP field. The earliest approaches, Coverage and Modification based (composite), came in 1997 [22]. Almost half of the recognized techniques were only coverage based (44%) followed by the composite, the fault based and other approaches as depicted by the pie chart in Fig. 5.

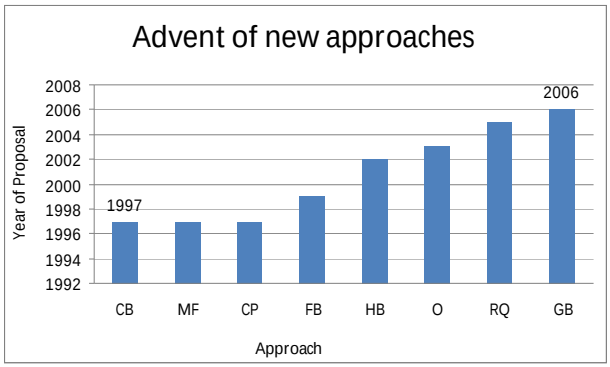

Figure 4: Advent of new approaches; CB-Coverage based, MF-Modification based, CP – Composite, FB-Fault based, HB- History based, O-Others, RQ-Requirements based, GB- Genetic based.

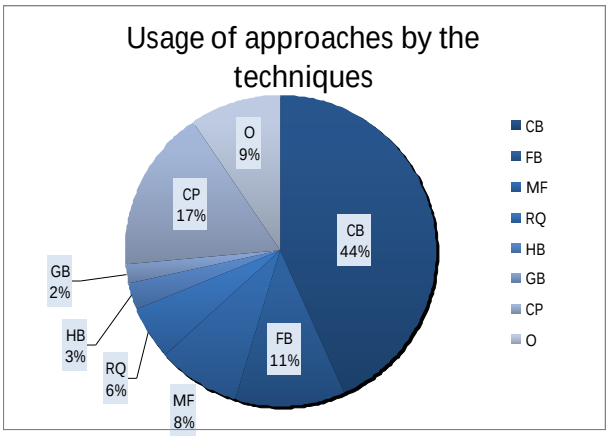

Figure 5: Usage of approaches by the techniques; CB-Coverage based, MF-Modification based, CP – Composite, FB- Fault based, HB- History based, O-Others, RQ- Requirements based, GB- Genetic based.

6.2 Are RTP techniques independent of their implementation language? (RQ2)

The groundwork culminated in determining 17 levels of Granularity utilized by the 106 techniques. It emerged that System level, Web services, Statement level and Function level granularity were largely utilized (Fig. 6). The input method used by majority of the techniques was found to be Source Code (as clearly shown in Fig. 7). This is justified as the majority of the prioritization techniques have been applied in the later stages of the software development life cycle (SDLC), i.e., after the source code is available. Also it can be inferred that the System Models were the next in majority to be used as the input method. System Models were mainly utilized by the techniques that came after the introduction of requirement based prioritization techniques. Thus we can observe an increase in the use of the prioritization techniques in the earlier stages of SDLC also. The distribution of the type of languages used by the techniques has been depicted in Fig. 8. About half of the techniques were found to be Language Independent, suggesting their compatibility over many languages. Approximately one-fourth of the techniques worked for Procedural languages only. An increasing use of the recent techniques for Web designing languages (16%) was noticed. Another major used language type was Object Oriented languages (11%). Binary code based and COTS component based languages also formed the basis of a few techniques.

It can be inferred from the above data that in-spite of huge variations in 1) the level of granularity at which an RTP technique is applied, and 2) Source code being majorly used as an input for an RTP technique; almost half of the RTP techniques were still found to be language independent. Although this is not sufficient to prove the independence of various RTP techniques from their implementation language, it encourages the current and future research in the field to be more language independent. This would allow various researchers to use each other's technique and will surely lead to better quality research and its assessment.

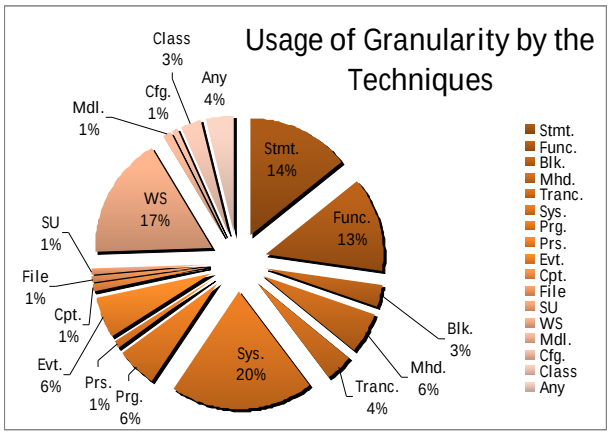

Figure 6: Usage of granularity by the techniques; Stmt.-Statement level, Func.-Function level, Blk.-Block of binary form, Mhd.-Method level, Tranc.-Transaction in System Model, Sys.-System level, Prg.-Program, Prs.-Process level, Evt.-Event, Cpt.-Component, File-file to be changed, SU-Software units, WS-Web services, Mdl.-Module, Cfg.-Configuration of software system, Class-class level, Any-any level.

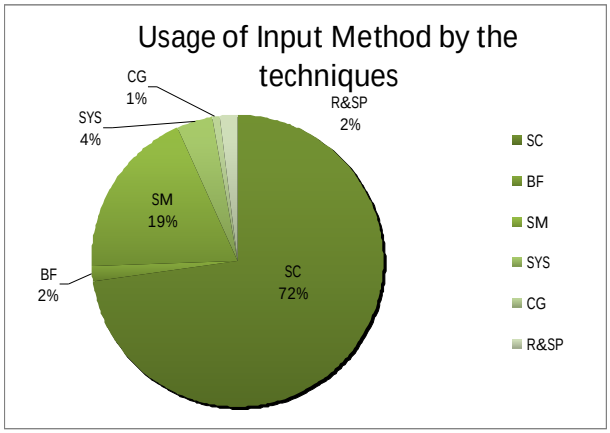

Figure 7: Usage of input method by the techniques; SC-Source code, BF-Binary code, SM-System level, SYS-System, CG-Call graph for program structure, R&SP-Requirements and specifications.

6.3 Identifying the gaps in the usage of Artifacts, Tools and Metrics in RTP (RQ3)

6.3.1 Artifacts

Artifacts are the pre-requisites for accomplishing controlled experiments on the testing techniques. Artifacts might comprise of software, test suites, fault data, coverage data, requirements, history information etc. depending on the type of experiment utilizing the artifacts. A thorough investigation of the artifacts used by the various regression testing techniques has already been presented by Yoo and Harman in [18]. They emphasized more on the size of Subject Under Test

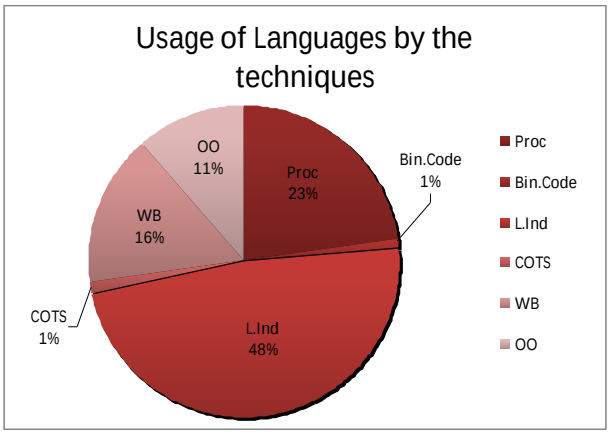

Figure 8: Usage of Languages by the Techniques;Proc - Procedural Language, Bin.Code- Binary code, L.Ind-Language independent, COTS-COTS component based, WB-Web designing language, OO-Object oriented language.

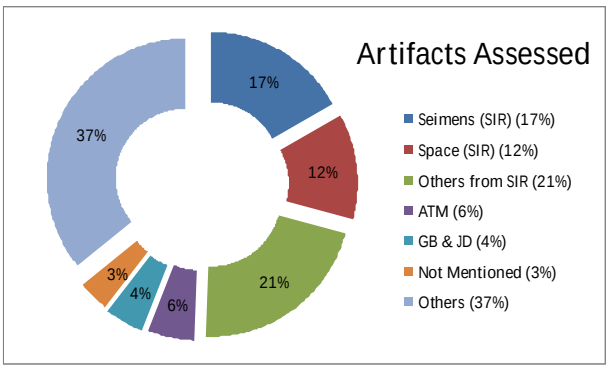

Figure 9: Artifacts assessed.

(SUT) and the test suites studied, and thus it has not been replicated here. They found that 60% of the researches used SUT's less than 10 KLOC, while on the other hand 70% of the studies have benefited from test suites with less than a 1000 test cases. Regarding the usage of artifacts, our results conform to those mentioned in [18], i.e. more than half of the artifacts have been freely procured from the Software Infrastructure Repository (SIR) [80]. The same has been demonstrated in the Fig. 9 having 50% share from SIR only. The research culminated in spotting 89 artifacts mentioned in 62 studies while 3 studies [51, 71, 76] did not mention any artifacts.

The seven C programs (printtokens, printtoken2, replace, schedule, schedule2, tcas, and totinfo) developed by Siemens Corporate Research and available on SIR [80] constitute 17% of all the artifacts used in [2, 3, 23, 24, 27, 31, 32, 44, 46, 50, 53, 62, 69, 73, 74]. A single 'Space' program (from SIR) of 6218 LOC, alone makes up 12% of the total artifacts exercised in [1, 2, 3, 22, 24, 27, 32, 44, 46, 50, 74]. SIR has also been used for some more programs accounting for 21% artifacts mentioned in all the studies [4, 6, 7, 8, 9, 26, 27, 30, 36, 37, 45, 54, 55, 57, 63, 70, 77, 78, 79]. Another single example of 'ATM' has been used in 5 researches [34, 36, 37, 64, 76]. A few studies [47, 48, 49, 72] have also made use of the JDepend (JD, tool for creating design quality metric for Java programs) and the Gradebook (GB, program performing tasks associated with creating and maintaining grade-book system for a course). Rest of the studies [5-7, 25, 27-29, 31, 33-37, 39-43, 48, 49, 52, 56, 59, 60, 65-68, 75-78] comprised of the artifacts developed using their own examples or the artifacts that have been sparsely touched upon by others. These 'other' artifacts amount to be a vast 37% of all. Hence it can be incurred that except the ones from SIR, no other major artifacts were found to be utilized unanimously by the RTP researchers.

The size of all the artifacts have not been mentioned by all of the respective studies and also the information about some artifacts was so less that they could not be assessed. Most (53) of the total artifacts mentioning their size were in KLOC's (i.e. over 1000 lines of codes) while a handful (14) were having size less than 1 KLOC. While 16 other artifacts had their size mentioned in terms of classes, methods or transitions rather than in terms of lines of codes. One possible reason behind this could be the usage of source code as input by majority of the RTP techniques developed till date. Once the source code is known, LOC becomes the size measure for that artifact. On the other hand the other size metrics used for artifacts correspond to the different types of input methods required by those RTP techniques. This was also confirmed by the fact that all the artifacts used for one particular RTP technique had the same size metric used (LOC or classes or method or transitions). This gap in the usage of artifact would remain also because various techniques follow various approaches for RTP. It does not make sense to calculate size of an artifact in LOC for being applied to requirements based approach, as it would not be possible to have the source code at the requirements analysis stage of SDLC.

6.3.2 Tools

There is an abundance of tools available nowadays, providing a fruitful means to the researchers for quick implementation and automatic analysis of their works. At the same time it is also the reason behind the unavailability of standard and worldwide accepted tools. In addition, many researchers need to develop their own tools to meet their particular requirements. Thus, various practitioners use various tools for their research instead of any single standard tool.

Though it can be perceived from Fig. 10 that 'Aristotle Program Analysis System' tool was used by 11 of the studies, which is the highest of all the tools used; it was used primarily by the same authors in different studies [2-4, 6, 23, 24, 27, 52, 53, 57, 62]. This tool was first used by Rothermel et al. in [23] for providing the test coverage and the control-flow graph information. Another tool used by five of the studies [30, 45, 77, 78, 79] was 'Sofya'. It helped in gathering the coverage information and the fault data of the test cases. 'Emma' tool has been utilized by 4 of the studies [43, 47, 53, 54] all by different authors. Emma is an open source toolkit for reporting Java code coverage. Few studies [2, 6, 31, 47, 52] also used 'UNIX based' tools such as UNIX Diff tool, UNIX utilities etc. for process tracking, collecting

dynamic coverage information or to show which lines were inserted or deleted from the basic version. 'SPSS' or Statistical package for Social Sciences is an upcoming data analysis tool used in the later researches [28, 71, 76, 79]. Another very promising tool used by 3 of the studies [37, 63, 72] is 'MATLAB'. It is an analysis and programming tool developed by Matrix Laboratory and is extensively used by many more applications; we expect it to be used more in the area of software testing also. One of the earlier used tools was 'Proteum Mutation System' to obtain the mutation scores for use in the Fault Exposing Potential (FEP) prioritization. Initially used in context of test case prioritization by Rothermel et al [23], it was further used in [3, 24, 27]. Its use was not spotted in any of the studies after 2002. The tools mentioning Java in their names (Junit Adaptor, Filter, JTester and byte code mutant generator) were grouped under the 'Java based' tools. Exploited in five studies [9, 47, 66, 77, 78], these tools varied in the purpose of their use but had a common language background, Java.

A couple of studies [8, 9] made use of the 'Galileo' system for acquiring coverage information by running test cases on the instrumented object programs in Java. 'Sandmark' is a watermarking program that provides change track algorithms employed by a handful of 3 studies [30, 54, 55] only. To comply with the specific requirements, seven studies mentioned their own created tools [23, 24, 27, 44, 63, 71]. Mostly the tools were created to automate their own proposed techniques. Some of the tools like Vulcan, BMAT, Echelon, déjà vu, GCOV, testrunner, winrunner, Rational test suite, bugzilla and Canatata++ etc. are only experienced in one study each. These all have been grouped under the Others category accounting for 31 such tools mentioned in [4-7, 22, 24-27, 30, 34, 35, 38, 39, 42-44, 46, 48, 52, 57, 69, 74, 75]. Exact details of the studies and tools used by them are available in the Appendix (Table A2).

None of the tools was discovered to be used by more than 13% of all the studies. This also generally results in the final outcomes that are not in a form comparable with the outcomes obtained using the other tools. Thus, we observed a wide range of tools used by all groups of the researchers without any particular standard being followed.

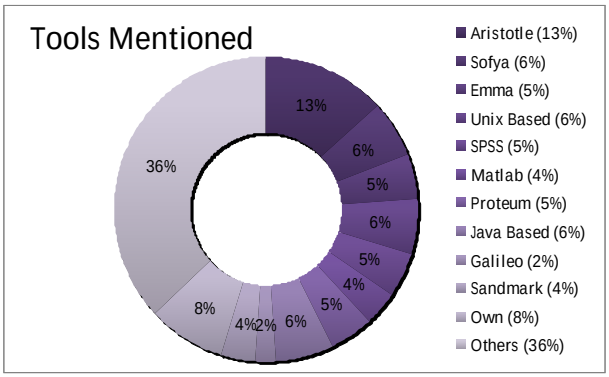

Figure 10: Tools mentioned.

6.3.3 Metrics

To properly understand the effects and the outcomes of any case study or experiment, one needs to quantify the results or analyze them with respect to the measures, well known in the software testing field as **metrics**. Unlike the scattered distribution seen in the tools usage, many of the diverse researchers tend to use the similar type or exactly the same metrics. Fig. 11 presents the final outcomes that commenced from scrutinizing the test case prioritization field in the view of the metrics used. We noticed a total of **97** metrics utilized by 60 research papers while 5 studies [29, 38, 60, 67, 71] did not mention any used metrics.

As clearly outlined in fig. 10, APFD came forth as a striking measure for computing the Average Percentage of Faults Detected and a massive of 29 studies [2-9, 23, 24, 26-28, 31, 33-35, 41, 46, 47, 53-55, 57, 62, 69, 73, 77] took advantage of the metric directly. This metric was originally set forth by the by a group of researchers in [23] and later used immensely by other groups of researchers as well. APFD metric denotes the weighted average of the percentage of the faults detected [2]. APFD values range from 0 to 100; higher numbers imply faster (better) coverage rates. It denotes how fast a prioritized test suite detects the faults. APFD is also being used in its mutant form as APFDc, APFDp, ASFD, WPFD, TSFD, APBC, APDC, APSC, NAPFD, APMC, TPFD, APRC, and BPFG. These have been put under the 'APFD alike' category shown in the graph [fig. 8]. 'APFD alike' are basically the metrics which are calculating average percentage of faults detected with some variations in the calculation method. APFDc is the modified APFD to include the costs of faults and is utilized by 5 researches [1, 52, 45, 63, 65]. Again a vast number of 10 studies [32, 39, 40, 42, 43, 57, 63, 64, 68, 74] benefited from the APFD alike metrics. These all sum up to more than 50% of the metrics availed by all the studies to be APFD or its mutants. It has now become a more or less standard in measuring the rate of fault detection achieved by the RTP techniques. We say so because, almost all the comparisons, whether between 2 or at most 18 techniques, given in the 65 studies were recorded to be based on the APFD (or its mutants) metrics. A meager of 5 studies [3, 6, 24, 27, 79] also made use of Bonferroni metric for analyzing their data. Bonferroni test provides a means of multiple comparisons in the statistical analysis. Various other metrics, whether available or self developed, such as PTR, RFFT, ATEI, ckjm, FDD, Kruskal Wallis Test, size reduction, precision, recall, efficacy, LOC count, and distance etc. have also been taken advantage of by a ide range of researchers [4, 5, 22, 23, 25, 26, 30, 31, 35-37, 41, 43, 44, 48-51, 56, 59, 63, 66, 70, 72, 75, 76, 78, 79]. Nonetheless, APFD and the other metrics provide a useful insight to the in-depth analysis of the techniques.

Explaining all the metrics along with their differences is beyond the scope of the current SLR, although there might be an SLR in future only on the software metrics being used for RTP that could include

the complete explanation and comparison for each technique.

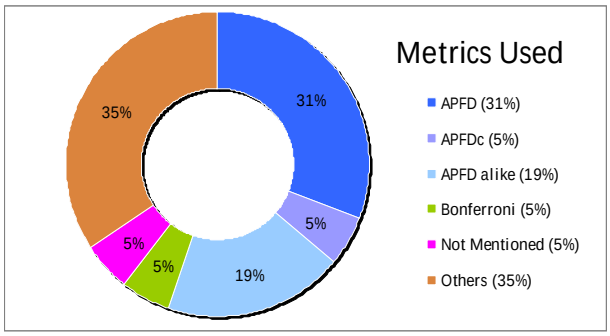

Figure 11: Metrics used.

6.4 Summarized effects of the Comparisons with respect to the granularity and the type of information used (RQ4)

A handful of articles [1, 6, 23] found the 'additional-fault exposing potential' technique to be better than all the compared ones. But these were only preliminary studies in the test case prioritization area. Among all the studied papers, following highlights were accrued:

6.4.1 With respect to the level of granularity (RQ4-(a))

Out of the 17 determined levels of granularity followed by the 106 techniques, the System level, Web services, Statement level and Function level granularity were the most utilized (as explained earlier in 6.2 and Fig. 6). None of the comparisons was detected to be based on the system level or the web services granularity. The techniques using the C language have only been tested for statement and function level granularity and it worked out that the statement level techniques were more advantageous over the function level techniques [9, 24, 26, 27]. The techniques for Java (Junit) environment showed no effect of granularity on the prioritization; the possible cause of this was suggested to be the difference in instrumentation granularity for Java and C [26].

6.4.2 With respect to the type of information (RQ4-(b))

An intricate scrutiny of the 65 research papers emanated the superiority of the additional techniques over the total and the other techniques. The additional techniques are the ones that are based on extra feedback information used in the process of test case prioritization. The comparisons in an enormous amount of 17 studies [1, 3, 4, 7, 8, 9, 23, 24, 26, 27, 30, 32, 33, 59, 74, 75, 79] turned out to produce similar outcomes: the additional techniques are more cost effective.

7 Conclusion

As the number of publications in the field of software testing is increasing, there is a need for a method that can summarize a researcher about the particular field. Systematic review is a tool that can be used to formally present the research made so far in a particular field. A systematic review on regression test prioritization techniques is presented in this paper which evaluates and interprets all the research work related to the area. It presents a concise summary of the best available evidences. The research identified over a hundred RTP techniques proposed since 1969. These were further classified based on the utilized approaches, and almost half of the recognized techniques came under the coverage based approach followed by composite, fault based and other approaches respectively. The paper summarizes the research papers along with the techniques they compared and the artifacts they processed. The tools and metrics being used in the research were also identified.

The input method used by majority of the techniques was computed to be the Source Code. This is justified by the use of the majority prioritization techniques in the final stages of SDLC, i.e., after the source code is available. After 2002, we also observed a general increase in the use of the prioritization techniques in the earlier stages of SDLC. Furthermore, an increasing use of the recent techniques for Web designing languages (16%) was detected.

Noticeably, it incurred that except the ones from SIR, no other major artifacts were found to be utilized unanimously by the researchers in the RTP field. No standard or sound majority could be established in the usage of tools. It lead to the results that were not in a form comparable with the outcomes obtained using the other tools. On the other hand, an analysis of the metrics used resulted in substantial findings. All the metrics spotted in the studies availing APFD or its mutants summed up to be more than 50%. Remarkably, we noticed that almost all the comparisons performed in all the selected studies were recorded to be based on the APFD (or its mutants) metrics. But failing to find the specific APFD values evaluated in the comparisons except for a few studies, it could not be possible to contrast all the techniques in general.

Though at most only 18 techniques were found to be compared in a single study, the results obtained provided useful insights into the RTP field. The inference drawn from the comparisons was that the additional techniques provided significant improvement in the fault detection rates. The level of granularity and modification information had no effect on prioritization for Java environment in general. Statement level techniques were found to be better than the function level techniques. Almost all the techniques were found to be better than the random technique. Many papers also presented comparisons with the optimal ordering, but since all the optimal orderings are defined according to the technique followed, it was not feasible to compare the optimal for different techniques.

The SLR finally highlighted that even after different approaches being followed by the various techniques, the prime goal of test case prioritization emerged as the increase in the rate of fault detection. Since no general

technique exists, there is a need to perform empirical comparisons among the existing techniques that are made to work on the same concept, implementation, metric and artifacts.

References

[1] S.Elbaum, A.Malishevsky, and G.Rothermel. "Incorporating varying test costs and fault severities into test case prioritization", Proceedings of the International Conference on Software Engineering, May 2000.

[2] S.Elbaum, D.Gable, and G.Rothermel, "Understanding and measuring the sources of variation in the prioritization of regression test suites", Proceedings of the International Software Metrics Symposium, pp. 167-179, Apr. 2001.

[3] G. Rothermel, R.H.Untch, C.Chu, and M.J.Harrold. "Prioritizing test cases for regression testing", IEEE Transactions on Software Engineering, Vol.27, No. 10, pp. 929-948, Oct.2001.

[4] S.Elbaum, P.Kallakuri, A.Malishevsky, G.Rothermel and S.Kanduri, "Understanding the effects of changes on the cost-effectiveness of regression testing techniques", Journal of Software Testing, Verification, and Reliability, Vol.12, No.2, pp.65-83, 2003.

[5] D. Leon,A. Podgurski, "A Comparison of coverage based and distribution based techniques for filtering and prioritizing test cases", In Proceedings of the 14th International Symposium on software reliability engineering(ISSRE 03),pp 442-453,2003.

[6] G. Rothermel, S.G.Elbaum, A.G.Malishevsky, P.Kallakuri, and X.Qiu. "On test suite composition and cost-effective regression testing", ACM Transaction Software Engineering Methodology, Vol.13, No.3, pages 277-331, July 2004.

[7] S.Elbaum, G.Rothermel, S.Kanduri and A.G.Malishevsky, "Selecting a cost-effective test case prioritization technique", Software Quality Journal, Vol.12, no.3, pp. 185-210, September 2004.

[8] H. Do, G.Rothermel. "A controlled experiment assessing test case prioritization techniques via mutation faults", Proceedings of the International Conference on Software Maintenance (ICSM), pp.411-420, 2005.

[9] H. Do, G. Rothermel and A. Kinner, "Prioritizing JUnit Test Cases: An Empirical Assessment and Cost-Benefits Analysis", An International Journal of Empirical Software Engineering, Vol. 11, No. 1, pp 33-70, March 2006.

[10] B.A. Kitchenham, T. Dybå, M. Jorgensen, "Evidence-based Software Engineering", in Proceedings of the International Conference of Software Engineering, 2004.

[11] T. Dybå, B.A. Kitchenham, M. Jorgensen, "Evidence-based Software Engineering for Practitioners", IEEE Software, Vol. 22, No. 1, pp. 58-65, 2005.

[12] B.Kitchenham, "Procedures for undertaking Systematic Reviews", Joint Technical Report, Computer Science Department, Keele University (TR/SE-0401) and National ICT Australia Ltd (0400011T.1), July 2004.

[13] N. Juristo, A.M. Moreno, S. Vegas, "Reviewing 25 years of testing technique experiments", Empirical Software Engineering Journal, Vol. 1, No. 2, pp. 7-44, 2004.

[14] M. Staples, M. Niazi, "Experiences using Systematic review guidelines," The Journal of Systems and Software, Elsevier Science Inc. USA, Vol. 80, No. 9, pp. 1425-1437, Sept 2007.

[15] B. Kitchenham, O.P. Brereton, D. Budgen, M. Turner, J. Bailey, S. Linkman, "Systematic literature reviews in software engineering – a systematic literature review", Journal of Information and Software Technology, Vol. 51, No. 1, pp. 7-15, 2009.

[16] E. Engström, P. Runeson, M. Skoglund, " A systematic review on regression test selection technique", Journal of Information and Software Technology, Vol. 52, Issue 1, pp. 14-30, 2010.

[17] E.Engström, P.Runeson, "A Qualitative Survey of Regression Testing Practices", Lecture Notes on Computer Science (LNCS), Springer Verlag, pp. 3-16, 2010.

[18] S.Yoo, M.Harman, "Regression Testing Minimization, Selection and Prioritization: A Survey", Software Testing, Verification and Reliability, Wiley Interscience, 2010.

[19] B.A. Kitchenham, "Guidelines for performing Systematic Literature reviews in Software Engineering Version 2.3". Technical Report S.o.C.S.a.M. Software Engineering Group, Keele University and Department of Computer Science University of Durham, 2007.

[20] T. Dyba, T. Dingso yr, G.K. Hanssen, Applying systematic reviews to diverse study types: an experience report, in: First International Symposium on Empirical Software Engineering and Measurement, 2007 (ESEM 2007), 2007, pp. 225–234.

[21] D. S. Cruzes and T. Dyba, "Research synthesis in software engineering: A tertiary study," Information Software Technology, Vol. 53, No. 5, May 2011, pp. 440-455. doi: 10.1016/j.infsof.2011.01.004.

[22] W.E. Wong, J.R.Horgan, S.London, and A.Aggarwal. "A study of effective regression testing in practice", Proceedings of the Eighth International Symposium Software Reliability Engineering, pp. 230-238, Nov. 1997.

[23] G. Rothermel, R.Untch, C.Chu, and M.J.Harrold, "Test case prioritization: An empirical study", Proceedings of International Conference Software Maintenance, pp. 179-188, Aug. 1999.

[24] S. Elbaum, A.Malishevsky, and G.Rothermel, "Prioritizing test cases for regression testing", Proceedings of the International Symposium on Software Testing and Analysis, pp. 102-112, Aug.2000.

[25] A.Srivastava, and J.Thiagarajan, "Effectively prioritizing tests in development environment",

Proceedings of the International Symposium on Software Testing and Analysis, pp.97-106, July 2002.

[26] H. Do, G.Rothermel, and Kinner. “Empirical studies of test case prioritization in a JUnit testing environment”, Proceedings of the International Symposium on Software Reliability Engineering, pp.113-124, Nov. 2004.

[27] S.Elbaum, A.G.Malishevsky, and G.Rothermel, “Test case prioritization: A family empirical studies”, IEEE Transactions on Software Engineering, Vol. 28, No. 2, pp. 159-182, Feb.2002.

[28] R.C. Bryce, A.M. Menon, “Test Suite Prioritization by Interaction coverage”, Proceedings of the workshop on domain specific approaches to software test automation (DOSTA), ACM, pp. 1-7, 2007.

[29] F.Belli, M.Eminov, N.Gokco. “Coverage-Oriented, Prioritized Testing-AFuzzy Clustering Approach and Case Study”. In :Bondavalli.A.,Brasileiro, F., Rajsbaum, S.(eds.) LADC 2007, LNCS, Springer, Heidelberg, Vol. 4746, pp. 95-110, 2007.

[30] H. Do, S. Mirarab, L. Tahvildari, G. Rothermel, "An Empirical Study of the effect of time constraints on the cost benefits of regression testing" Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering,pp71-82,2008.

[31] B.Jiang, Z.Zhang, W.K.Chan, T.H.Tse, “.Adaptive Random test case prioritizatation.” In Proceedings of International Conference on Automated Software Engineering, pp:233-243, 2009.

[32] C. L. B. Maia,R. A. F. do Carmo, F. G. de Freitas,G. A. L. de campos,and J. T. de Souza, "Automated test case prioritization with reactive GRASP,” In Proceedings of Advances in Software Engineering, pp.1-18, 2010.

[33] L.Mei, Z.Zhang, W.K.Chan, T.H.Tse, "Test case prioritization for regression testing of service oriented Business Applications", In Proceedings of the 18th International World Wide Web Conference (WWW 2009), pp. 901-910, 2009.

[34] L.Mei, W.K.Chan, T.H.Tse., R.G.Merkel, “XML-manipulating test case prioritization for XML-manipulating services,” Journal of Systems and Software, pp.603-619, 2010.

[35] R.C.Bryce, S.Sampath, A.M.Memon, "Developing a Single Model and Test Prioritization Strategies for Event Driven Software", IEEE Transactions on Software Engineering, pp.48-63, 2010.

[36] B. Korel, G. Koutsogiannakis, L.H. Talat, “Model-based test suite prioritization Heuristic Methods and Their Evaluation”, Proceedings of 3rd workshop on Advances in model based testing (A – MOST), London, UK, pp. 34-43, 2007.

[37] B. Korel, L. Tahat, M. Harman, "Test Prioritization Using System Models", In the Proceedings of 21st IEEE International Conference on Software Maintenance (ICSM'08), pp. 247-256, 2005.

[38] R.S.S.Filho, C.J.Budnik, W.M.Hasling, M.M.Kenna, R.Subramanyam, "Supporting concern based regression testing and prioritization in a model driven environment", In Proceedings of 34th Annual IEEE Computer software and Applications conference Workshops (COMPSA 10), pp.323-328, 2010.

[39] H. Srikanth, L. Williams, J. Osborne, "System Test Case Prioritization of New and Regressionb Test Cases", In the Proceedings of International Symposium on Empirical Software Engineering (ISESE), pp. 64-73, Nov. 2005.

[40] H. Srikanth, L.Williams, "On the Economics of requirements based test case prioritization", In Proceedings of the Seventh International conference on Economics Driven Software Engineering Research (EDSER 05), pp1-3, 2005.

[41] S. Hou, L. Zang, T. Xie, J. Sun, "Quota-Constrained Test Case Prioritization for Regression Testing of Service-Centric Systems", Proceedings of International Conference on Software Maintenance, pp. 257-266, 2008.

[42] R.Krishnamoorthi, S.A.Mary, “Incorporating varying requirement priorities and costs in test case prioritization for new and regression testing”, Proceedings of International Conference on Computing, Communication and Networking (ICCN), pp.1-9, 2008.

[43] R.Krishnamoorthi, S.A.S.A.Mart, “Factor oriented requirement coverage based system test case prioritization of new and regression test cases”, Journal of information and software technology, Vol. 51, pp. 799-808, 2009.

[44] J.M.Kim, A.Porter. “A History-Based Test Prioritization Technique for Regression Testing in Resource Constrained Environment”, Proceedings of the 24th International Conference Software Engineering, pp.119-129, May.2002.

[45] H. Park, H.Ryu, J.Baik, “Historical value-based approach for cost-cognizant test case prioritization to improve the effectiveness of regression testing”, Proceedings of second International Conference on Secure System Integration and reliability, Improvement, pp. 39-46, 2008.

[46] Y.Fazlalizadeh, A.Khalilian, H.A.Azgomi, S.Parsa, “Incorporating historical test case performance data and resource constraints into test case prioritization”, Lecture notes in Computer Science, Springer, Vol. 5668, pp. 43-57, 2009.

[47] K.R.Walcott, M.L.Soffa, G.M.Kapfhammer and R.S. Roos. “Time aware test suite Prioritization”, Proceedings of International Symposium on software Testing and Analysis (ISSTA), pp. 1-12, July 2006.

[48] A. P.Conrad,R. S.Roos, "Empirically Studying the role of selection operators during search based test suite prioritization", In the Proceedings of the ACM SIGEVO Genetic and Evolutionary Computation Conference, Portland, Oregon, 2010.

[49] A.M.Smith, G.M.Kapfhammer, “An empirical study of incorporating cost into test suite reduction and prioritization”, Proceedings of ACM Symposium on Applied Computing, pp. 461-467, 2009.

[50] J.A. Jones, and M.J. Harrold, “Test suite reduction and prioritization for modified condition/decision coverage”, Proceedings of the IEEE Transactions on Software Engineering, Vol.29, No.3, March, 2003.

[51] K.H.S. Hla, Y. Choi, J.S. Park, "Applying Particle Swarm Optimization to Prioritizing Test Cases for Embedded Real Time Software Retesting", Proceeding of 8th International Conference on Computer and Information Technology Workshops, pp. 527-532, 2008.

[52] A.G.Malishevsky, J.Ruthruff, G. Rothermel and S.Elbaum, “Cost-cognizant test case prioritization”, Technical Report TR-UNL-CSE-2006-0004, University of Nebraska-Lincoln, 2006.

[53] D.Jeffrey, N.Gupta. “Test-case Prioritization using relevant slices”, Proceedings of the 30th annual International Computer Software and Applications (COMPSAC), Chicago, USA, pp.18-21, September 2006.

[54] S. Mirarab and L. Tahvildari, "A Prioritization Approach for Software Test Cases on Bayesian Networks", FASE, Lecture Notes in Computer Science, Springer, 4422-0276, pp. 276-290, 2007.

[55] S.Mirarab, L.Tahvildari. “An Empirical study on Bayesian Network-based approach for test case prioritization”, Proceedings of International conference on software testing verification and validation, pp. 278-287, 2008.

[56] B.Qu, C.Nei, B.Xu, X. Zhang, ”Test case prioritization for black box testing”, In Proceedings of 31st Annual International Computer Software and Applications Conference (COMSAC 2007), vol. 1, pp. 465-274, 2007.

[57] X. Qu, M. B. Cohen and K.M. Woolf, "Combinatorial Interaction Regression Testing: A Study of Test Case Generation and Prioritization", In the Proceedings of International Conference on Software Maintenance, pp. 255-264, Oct., 2007.

[58] R.Bryce, C.Colbourne. "Prioritized interaction testing for pair-wise coverage with seeding and constraints", Journal of Information and Software Technology, Vol. 48, No. 10, pp. 960-970, May 2006.

[59] X. Zhang, C. Nie, B. Xu, B.Qu, "Test Case Prioritization based on Varying Testing Requirement Priorities and Test Case Costs", Proceedings of the 7th International Conference on Quality Software, pp. 15-24, 2007.

[60] M. Sherriff, M. Lake, L. Williams, "Priroitization of Regression Tests using Singular Value Decomposition with Empirical Change Records", The 18th IEEE International Symposium on Software Reliability Engineering, Trllhattan, Sweden, pp. 82-90, Nov-2007.

[61] G.Rothermel, M.Harrold, "Analyzing Regression Test Selection Techniques", IEEE Transactions on Software Engineering, Vol. 22, pp. 529-551, Aug 1996.

[62] D. Jeffrey, N. Gupta, "Experiments with Test Case Prioritization using Relevant Slices", Journal of Systems and Software, Vol. 81, No. 2, pp. 196-221, 2008.

[63] Z. Ma, J.Zhao, “Test Case Prioritization based on analysis of program structure”, In the Proceedings of 15th Asia-Pacific Software Engineering conference, pp. 471-478, 2008.

[64] L. Chen,Z. Wang,L. X.u,H. Lu,B. Xu, "Test Case prioritization for web service regression testing", In Proceedings of the Fifth International Symposium on servive oriented system engineering, pp. 173-178, 2010.

[65] Y. C. Huang,C.Y. Huang,J.R. Chang,T.Y. Chen, "Design and Analysis of cost cognizant test case prioritization using genetic algorithm with test history", In proceedings of 34th Annual IEEE Computer Software and Applications Conference (COMSAC 2010), pp. 413-418, 2010.

[66] M.J. Rummel, G.M. Kapfhammer, A. Thall, "Towards the Prioritization of Regression Test Suites with Data Flow Information", Proceedings of the 2005 ACM Symposium on Applied Computing, pp. 1499-1504, March 13-17, 2005.

[67] L. Mariani, S. Papagiannakis and M. Pezze, “Compatibility and Regression Testing of COTS-Component-Based Software”, Proceedings of the 29th International Conference on Software Engineering (ICSE), USA, pages 85-95, May 2007.

[68] B.Qu, C.Nie, B.Xu, “Test case prioritization for multiple processing queues”, Proceedings of International Symposium on Information Science and Engineering, pp. 646-649, 2008.

[69] M.K.Ramanathan, M.Koyuturk, A.Grama, “PHALANX : A Graph-Theoretic Framework for Test Case Prioritization”, Proceedings of ACM Symposium on Applied Computing (SAC), pp. 667-673, March 2008.

[70] X.Qu, M.B.Cohen, and G.Rothermel. “Configuration-aware regression testing: An empirical study of sampling and prioritization”, Proceedings of the ACM/SIGSOFT International Symposium on Software Testing and Analysis (ISSTA), pp.75-85, 2008.

[71] Y.T.Yu,S.P. Ng,E.Y.K.Chan, "Generating,Selecting and Prioritizing test cases from Specifications with tool support", In the Proceedings of Third International Conference on quality software (QSIC 03), pp. 83, 2003.

[72] S.Alspaugh, K.R. Walcott, M.Belanich, G.M.Kapfhammer, M.L.Soffa.”Efficient time aware prioritization with knapsack solvers”, Proceedings of the 1st ACM international workshop on empirical assessment of software engineering languages and technologies: held in conjunction with the 22nd IEEE/ACM International Symposium on Automated Software Engineering(ASE) 2007, pp.13-18, Nov.2007.

[73] C. Simons,E. C. Paraiso,"Regression Test cases Prioritization Using Failure Pursuit Sampling",In Proceedings of Tenth International Conference on Intelligent Systems Design and applications, pp.923-928, 2010.

[74] Z. Li, M. Harman and R.M. Hierons, "Search Algorithm for Regression test case prioritization", IEEE TSE, Vol. 33, No. 4, 2007.
[75] S. Li,N. Bian,Z. Chen,D. You,Y. He, "A simulation on some search algorithms for regression test case prioritization", In Proceedings of 10th international conference on Quality software, pp. 72-81, 2010.
[76] B. Korel,G. Koutsogiannakis, "Experimental comparison of code based and model based test prioritization", In Proceedings of the IEEE International Conference on Software Testing, Verification, and Validation Workshops(ICSTW '09), pp. 77-84, 2009.
[77] H. Do, G. Rothermel, "On the use of Mutation Faults in Empirical assessment of Test Case Prioritization Techniques", IEEE Transaction of Software Engineering, pp. 733-752, Sep-2006.
[78] H. Do, G. Rothermel, "An Empirical Study of Regression Testing Techniques Incorporating Context and Lifetime Factors and Improved Cost-Benefit Models", Proceedings of the ACM SIGSOFT Symposium on Foundations of Software Engineering, pp. 141-151, Nov. 2006.
[79] H. Do,S. Mirarab,L. Tahvildari,G. Rothermel, "The Effects of Time Constraints on Test case Prioritization :A series of Controlled Experiments", IEEE Transactions on Software Engineering, vol. 36, no. 5, pp. 593-617, 2010.
[80] http:// sir.unl.edu/portal/index.php

Appendix A

Table A1: List of primary studies with their authors and techniques.

Research Paper ID	Authors	Techniques
RP1	Wong et. al	T1
RP2	Rothermel et. al	T2, T3, T4, T5, T6, T7
RP4	Elbaum et. al	T8,T9,T10,T11,T12,T13,T14,T15
RP3	Elbaum et. al	No new technique
RP5	Elbaum et. al	No new technique
RP6	Rothermel et. al	No new technique
RP7	Elbaum et. al	T16, T17, T18, T19,
RP9	Kim and Porter	T20
RP8	Srivastava and Thiagarajan	T21
RP11	Jones and Harrold	T22
RP10	Elbaum et. al	No new techniques
RP12	Yu et al	T22.1
RP13	Leon et al	No new technique
RP15	Elbaum et. al	No new technique
RP14	Rothermel et. al	No new technique
RP16	Do et. al	T23, T24, T25, T26, T27, T28
RP18	Srikanth et al	T28.1
RP19	Do and Rothermel	No new technique
RP20	Korel et. al	T29, T30
RP17	Rummel et. al	T31
RP21	Srikanth et. al	T32
RP25	Do and Rothermel	No new technique
RP27	Do and Rothermel	No new technique
RP22	Do et. al	No new technique
RP23	Malishevsky et. al	T33
RP26	Jeffrey and Gupta	T34
RP24	Walcott et. al	T35
RP36	Alspaugh et. al	T36
RP38	Belli et. al	T37
RP33	Bryce and Memon	T38 to T42
RP31	Korel et. al	T43 to T47
RP32	Qu et al	T47.1
RP28	Li et. al	A1 to A5
RP30	Mariani et. al	T48
RP29	Mirarab and Tahvildari	T49
RP34	Qu et. al	T50
RP37	Sherrif et. al	T51
RP35	Zang et. al	T52, T53
RP43	Hla et. al	T54
RP46	Hou et al.	T55, T56
RP39	Jeffrey and Gupta	T57, T58
RP40	Ramanathan et. al	T59

RP48	Ma and Zhao	T60
RP41	Mirarab and Tahvildari	T61 (enhanced T51)
RP42	Park et. al	T62
RP49	Qu et. al	T63
RP44	Qu et.al	T64
RP45	Ramasamy and Mary	T65
RP47	Do et al	T65.1 – T65.4
RP50	Smith and Kapfhammer	Extension of algorithms in RP29(A3, A5) and delayed greedy algorithm
RP54	Fazlalizadeh et. al	T66
RP51	Krishnamoorthi and Mart	T67
RP52	Mei et al	T67.1 – T67.10
RP53	Korel et al	No new technique
RP55	Jiang et al	T68.1 – T68.9
RP56	Maia et al	T69
RP57	Chen et al	T70.1, T70.2
RP58	Filho et al	T71.1,T71.2
RP59	Li et al	No new technique
RP60	Huang et al	T72
RP61	Conrad et al	T73
RP62	Do et al	No new technique
RP63	Simons et al	T74
RP64	Mei et al	T75.1 – T75.4
RP65	Bryce et al	T76

Table A2: List of Research papers, techniques, artifacts, tools and metrics used by them.

Research Paper ID	**Technique(s)**	**Artifacts**	**Tools**	**Metrics**
RP1 [19]	T1	Space program (SIR)	ATAC (Automatic Testing Analysis Tool)	Size reduction, Precision, Recall
RP2 [20]	T2, T3, T4, T5, T6, T7	7 C programs from siemens corporate research (SIR)	Aristotle Program Analysis System, Prioritization tool (created own), Proteum Mutation System	Efficacy, APFD
RP3 [1]	No new technique	Space program (SIR)	N.M	APFDc
RP4 [21]	T8,T9, T10,T11,T12, T13,T14,T15	7 C programs from siemens and 1 space program (SIR)	Aristotle Program Analysis System, Prioritization tool (created own), Proteum Mutation System, Source code measurement tool, Comparator, Fault Index generater	APFD, Bonferroni analysis
RP5 [2]	No new technique	7 siemens program and 1 space program (SIR)	Aristotle program Analysis System, UNIX Diff tool	APFD
RP6 [3]	No new technique	7 siemens program and 1 space program (SIR)	Aristotle Program Analysis System, Proteum Mutation System	APFD, Bonferroni Means Separation Test
RP7 [24]	T16, T17, T18, T19,	8 SIR programs (7 siemens and 1 space program), 3 case studies : 2 open source UNIX utilities from SIR (grep and flex), 1 embedded real time subsystem of a level 5 RAID storage system	Aristotle Program Analysis System, Prioritization tool(created own), Proteum Mutation System, Source code measurement tool, Comparator, Fault Index generater	APFD, Bonferroni analysis
RP8 [22]	T21	Two versions of large office productivity application	Vulcan (Rich Binary Modification Infrastructure), BMAT(Binary mathching tool build using System	Coverage of impacted blocks

			Vulcan), Echelon	
RP9 [41]	T20	7 siemens program and 1 space program (SIR)	déjà vu tool, Created tool for minimizatio n, Created tool for Random technique	Total testing effort, Average fault age
RP10 [4]	No new techniq ue	bash , grep, flex, gzip (SIR)	Aristotle Tool Suite, TSL tool for generating test suite, Tool to implement their process	Percentage of changed LOC, functions and files, APFD, Probability of execcution of changed function, Average no. of LOC changed per function, Average percentage of tests executing changed functions
RP11 [47]	T22	tcas (siemens) and one space program (SIR)	N.M	Time to perform prioritizatio n
RP12 [68]	T23	N.M.	Created EXTRACT tool for prioritization	N.M.
RP13 [5]	No new techniq ue	Three large programs: GCC,Jikes and Javac Compilers	GCOV tool for profiling	APFD, Dissimilarit y Mertic
RP14 [6]	No new techniq ue	bash (SIR) and emp_server	Aristotle Program Analysis System, Clic Instrumenter and monitor, Unix utilities	APFD, Annova analysis, Bonferroni test
RP15 [7]	No new techniq ue	bash, flex, grep, gzip, make, sed from SIR and emp_server, xearth	TSL Tool	APFD
RP16 [23]	T24, T25, T26, T27,	ant, XML-security, Jmeter, Jtopas from	Selective Testrunner	APFD, Boxplots, ANNOVA analysis
	T28, T29	SIR		
RP17 [63]	T33	3 applications in JAVA: Bank, Identifier and Money	Soot 1.2.5(Java Optimization Framework)	APFD, PTR(Percen tage of test suite that must be executed to find all defects)
RP18 [37]	T30	5 projects developed by students	N.M	ASFD, TSFD
RP19 [8]	No new techniq ue	ant, XML-security, Jmeter, Jtopas from SIR	Mutation tool, Galileo system for coverage information	APFD
RP20 [34]	T31, T32	3 system models : ATM model, cruise control model (SIR), fuel pump model	N.M	Rate of fault detection
RP21 [36]	T34	4 Java projects developed by students	PORT tool, TCP tool	Weighted percentage of fault detected (WPFD)
RP22 [9]	No new techniq ue	Ant , XML-security, Jmeter, Jtopas from SIR	Galileo system for coverage information, Junit adaptor, JunitFilter, TestRunner	APFD
RP23 [49]	T35	emp_server portion of Empire software	UNIX Diff tool, Aristotle program Analysis System, Tools to prioritize test cases	APFDc
RP24 [44]	T37	Gradebook and Jdepend	Emma tool, Linux Process tracking tool, JTester	APFD
RP25 [74]	No new techniq ue	Ant , XML-security, Jmeter, Jtopas, nanoxml from SIR and Galileo	Sofya system, Junit adaptor	APFD
RP26 [50]	T36	7 siemens program	Aristotle Program Analysis Tool	APFD

RP27 [75]	No new technique	Ant , XML-security, Jmeter, nanoxml from SIR and Galileo	Sofya system, Java bytecode Mutant generator	Cost and Benefit
RP28 [71]	A1 to A5	print_tokens, print_tokens 2, schedule, schedule2 from siemens and space, sed from SIR	Canatata++, SPSS	APBC(Average percentage of block coverage), APDC (Average percentage of Decision coverage), APSC (Average Perecentage of Statement Coverage)
RP29 [51]	T52	Apache Ant (SIR)	ckjm, Emma, Sandmark	APFD
RP30 [64]	T51	15 configurations of Ginipad Java Editor version 2.0.3 including 316 Java classes	N.M	Not Mentioned
RP31 [33]	T45 to T49	cruise control model from SIR, ATM Model, fuel pump model, TCP model, ISDN model	N.M	Most likely relative position of test case
RP32 [53]	T50	Software for Microsoft Word and Power point(checks the performance when opening malicious documents)	N.M	Speed Of fault detection
RP33 [25]	T40 to T44	TerpCalc, TerpPaint, TerpSpeadsheet, TerpWord	N.M	APFD
RP34 [54]	T53	flex and make from SIR	SSLOC tool, Aristotle Coverage tool	APFD, NAPFD
RP35 [56]	T55, T56	Simulation experiments	N.M	Rate of units of testing requirement
				priority satisfied per unit test case cost
RP36 [69]	T38, A7, A8, A9	Gradebook and Jdepend software	Emma tool, Linux Process tool	Code coverage, Coverage preservation, Order aware coverage
RP37 [57]	T54	3 minor releases of IBM software system	MATLAB	Not Mentioned
RP38 [26]	T39	web based system ISELTA(Isik's System for Enterprise Level web centric Tourist Applications)	N.M	None
RP39 [59]	T60, T61	7 C pprograms from Siemens	Aristotle Program Analysis Tool	APFD
RP40 [66]	T62	7 C pprograms from Siemens	MATLAB, PIN tool	APFD
RP41 [52]	T64 (enhanced T54)	Ant , XML-security, Jmeter, nanoxml from SIR and Galileo	Sandmark tool	APFD
RP42 [42]	T65	ant (SIR)	Sofya system	APFDc
RP43 [48]	T57	N.M	N.M	Coverage
RP44 [67]	T67	Vim from SIR	Mutation testing tool	Block Coverage, Fault Detection, Change across faults, Change across tests
RP45 [39]	T68	5 J2EE application projects developed by students and 2 set of Industrial project (one VB and one PHP)	Rational Test Suite, Tbreq-Requirement tracability tool	TSFD (Total Severity of Faults Detected)
RP46 [38]	T58, T59	Travel agent system having 12	N.M	Total branch coverage,

		web services with 17 methods in Java		APFD
RP47 [27]	T69 – T70	Ant, XML_security, Jmeter, nanoxml from SIR and Galileo	Sandmark Tool, Sofya System, Ckjm Tool	Boxplots, Chidamber, Kemerer(ckjm metrics)
RP48 [60]	T63	XML-security, Jtopas from SIR	Apros tool	APFDc, APMC (Average percentage of fault affected modules cleared per test case)
RP49 [65]	T66	Microsoft Power Point 2003(11.8106.8107)	N.M	APFDp(for parallel scenario)
RP50 [46]	Extension of algorithms in RP29(A3, A5) and delayed greedy algorithm	8 case study application: DS (Data Stucture application), GB (Grade Book Application), JD (Jdepend), LF (LoopFinder), RM (Reminder), SK (Sudoko), TM(Transaction Manager), RP (Reduction and Prioritization package)	N.M	RFFT (Reduction factor for time)
RP51 [40]	T74	5 J2EE application projects by students, two industrial projects (one in VB, one in PHP) and two industrial case studies (java)	Emma tool, Winrunner testing tool	ASFD (Average Severity of Faults Detected), ATEI (Average Test Effort Index), TPFD (Total Percentage of Failures Detected)
RP52 [30]	T75 – T84	Business process Hotelbooking taken from the Triphandling project	N.M	APFD
RP53 [73]	No new technique	Seven models from the Infrastucture: ATM model, Cruise Control, Fuel Pumps, Fuel Pumps, TCP-Dialer, ISDN, Print Token, Vending Machine	SPSS Tool	Relative Position Metric
RP54 [43]	T73	7 siemens program and 1 space program	SAS 9.1.3	APFD
RP55 [28]	T85 – T93	Seven Siemens programs and 4 mediun sized unix utility programs	SPSS Tool, Unix Tool Gcov to collect dynamic coverage information	APFD Metric, Distance Metric called Jaccard distance
RP56 [29]	T94, A2, A3, A4, A6	Four siemens programs: print_tokens, print_tokens 2, schedule, schedule2 and one space program from SIR	N.M	APBC(Averge Percentage Block Coverage), APDC(Average percentage Decision Coverage), APSC(Average percentage statement Coverage)
RP57 [61]	T95, T96	ATM example	N.M	BPFG
RP58 [35]	T97,T98	N.M.	TDE/UMl model Editor	N.M.
RP59 [72]	A1 to A5	One simulated program	Microsoft Visual Studio 2008	APRC(Average percentage Requirement coverage
RP60 [62]	T99	Five versions of sed(unix utility program)	N.M	APFDc
RP61 [45]	T100	8 case study application: DS (Data Stucture application), GB (Grade Book Application)	Beanplots	Coverage effectiveness Score metric

		, JD (Jdepend), LF (LoopFinder), RM (Reminder), SK (Sudoko), TM(Transaction Manager), RP (Reduction and Prioritization package)		
RP62 [76]	No new techniq ue	Five java Programs namely ant,xml security,jmeter,nanoxml from SIR and galileo	Sofya system, SPSS Tool	EVOMO and LOC Model, Bonferroni Analysis, Kruskal Wallis Test(non parametric one way analysis)
RP63 [70]	T101	schedule (Siemens)	N.M	APFD
RP64 [31]	T102 – T105	A set of WS-BPEL applications: Atm, Buybook, Dslservice, Gymlocker, Loan Approval, Marketplace, Purcharse, TripHandlin g	MATLAB Tool, PTS Box chart Utility	APFD, Boxplots, Annova Analysis
RP65 [32]	T106	Four GUI Applications :Terpcalc, TerpCalc,TerPaint,TerpSpredsheet and Terpword which is an open soutce Office suite developed at University of Maryland and Three web based Applications :Book,CPM and Masplas	Bugzilla(Bug Tracking Tool)	APFD, FDD(fault detection density)

Table A3: List of techniques with the language type, input method, approach and granularity.

T. no	**Technique**	**Language type**	**Input Method**	**Approach**	**Granularity**
T1	Hybrid technique combining modification, minimization and prioritization	Proc.	SC	MF and CB	Stmt.
T2	Total branch coverage prioritization	Proc.	SC	CB	Stmt.
T3	Additional branch coverage prioritization	Proc.	SC	CB	Stmt.
T4	Total statement coverage prioritization	Proc.	SC	CB	Stmt.
T5	Additional statement coverage prioritization	Proc.	SC	CB	Stmt.
T6	Total fault-exposing potential (FEP) prioritization	Proc.	SC	FB	Stmt.
T7	Additional fault-exposing potential (FEP) prioritization	Proc.	SC	FB	Stmt.
T8	fn_total (prioritize on coverage of functions)	Proc.	SC	CB	Func.
T9	fn_addtl (prioritize on coverage of functions not yet covered)	Proc.	SC	CB	Func.
T10	fn_fep_total (prioritize on probability of exposing faults)	Proc.	SC	FB	Func.
T11	fn_fep_addtl (prioritize on probability of exposing faults, adjusted to consider previous test	Proc.	SC	FB	Func.

	cases)				
T12	fn_fi_total (prioritize on probability of fault existence)	Proc.	SC	FB	Func.
T13	fn_fi_addtl (prioritize on probability of fault existence, adjusted to consider previous test cases)	Proc.	SC	FB	Func.
T14	fn_fi_fep_total (prioritize on probability of fault existence and fault exposure)	Proc.	SC	FB	Func.
T15	fn_fi_fep_addtl(prioritize on probability of fault existence and fault exposure adjusted to previous coverage)	Proc.	SC	FB	Func.
T16	fn_diff_total (prioritize on probability of fault existence based on DIFF)	Proc.	SC	FB	Func.
T17	fn_diff_addtl (prioritize on probability of fault existence adjusted to consider previous test cases based on DIFF)	Proc.	SC	FB	Func.
T18	fn_diff_fep_total (prioritize on combined probability of fault existence with fault exposure based on DIFF)	Proc.	SC	FB	Func.
T19	fn_diff_fep_addtl (prioritize on combined probability of fault existence with fault exposure, adjusted on previous coverage based on DIFF)	Proc.	SC	FB	Func.
T20	Prioritization based on history-based on test execution history in resource constrained environment	Proc.	SC	HB	Stmt.
T21	Binary code based prioritization	Bin. Code	BF	CB	Blk.
T22	Prioritization incorporating aspects of MC/DC	Proc.	SC	CB + MF	Stmt.
T23	Annotated Classification Tree based prioritization	L.Ind.	R&SP	O (CTB)	Class
T24	block_total(prioritization on coverage of blocks)	OO	SC	CB	Blk.
T25	block_addtl(prioritization on coverage of blocks not yet covered)	OO	SC	CB	Blk.
T26	method_total (prioritization on coverage of method)	OO	SC	CB	Mhd.
T27	method_addtl(prioritization on coverage of methods not yet covered)	OO	SC	CB	Mhd.
T28	method_diff_total(prioritize on coverage of method and change information)	OO	SC	CB	Mhd.
T29	method_diff_addtl(prioritize on coverage of method and change information adjusted to	OO	SC	CB	Mhd.

	previous coverage)				
T30	PORT(V1.0)	L.Ind.	R&SP	RQ	Sys.
T31	System model based selective test prioritization	L.Ind.	SM	MF	Tranc.
T32	Model dependence based test prioritization	L.Ind.	SM	MF	Tranc.
T33	Data flow based prioritization	OO	SC	DF	Stmt.
T34	Prioritization Of Requirements for Test(PORT)	L.Ind.	SYS	RQ	Sys.
T35	Cost-Cognizant TCP	L.Ind.	SC	CST & FB	Func.
T36	Prioritization using relevant slices(REG+OI+POI approach)	Proc.	SC	MF & SLC	Stmt.
T37	Prioritization using genetic algorithm	OO	SC	GB	Prg.
T38	Time aware prioritization using Knapsack solvers	OO	SC	KB	Stmt.
T39	Graph model based approach for prioritization	L.Ind.	SM	CB	Prs.
T40	Interaction coverage based prioritization by length of test(longest to shortest) on event driven softwares	L.Ind.	SC	CB	Evt.
T41	Interaction coverage based prioritization by 3-way interaction on event driven softwares	L.Ind.	SC	CB	Evt.
T42	Interaction coverage based prioritization by 2-way interaction on event driven softwares	L.Ind.	SC	CB	Evt.
T43	Interaction coverage based prioritization by unique event coverage on event driven softwares	L.Ind.	SC	CB	Evt.
T44	Interaction coverage based prioritization by length of tests(shortest to longest) on event driven softwares	L.Ind.	SC	CB	Evt.
T45	Model based heuristic#1 prioritization	L.Ind.	SM	MF	Sys.
T46	Model based heuristic#2 prioritization	L.Ind.	SM	MF	Sys.
T47	Model based heuristic#3 prioritization	L.Ind.	SM	MF	Sys.
T48	Model based heuristic#4 prioritization	L.Ind.	SM	MF	Sys.
T49	Model based heuristic#5 prioritization	L.Ind.	SM	MF	Sys.
T50	Test case prioritization for black box testing based on requirements and history	L.Ind.	SYS	RQ+HB	Sys.
T51	Priritizing test cases for COTS components	COTS	SC	ICB	Cpt.
T52	Bayesian network based test case prioritization	OO	SC	MF+CB+FB+BN	Prg.
T53	Combinatorial Interaction regression testing based	Proc.	SC	CB & IB	Prg.

	prioritization				
T54	Prioritization using Singular Value Decomposition (SVD) with empirical change records	L.Ind.	SC	MF & SVD	File
T55	Prioritization based on testing requirement priorities and test case cost	L.Ind.	SC	RQ & CST	Any
T56	Prioritization based on additional testing requirement priorities and test case cost	L.Ind.	SC	RQ & CST	Any
T57	Particle Swarm Optimization based prioritization	L.Ind.	SC	MF + CB	SU
T58	Quota constrained test case prioritization	L.Ind.	SC	RQ	WS
T59	Quota constrained additional test case prioritization	L.Ind.	SC	RQ	WS
T60	Prioritization using heuristic REG_OI_POI with grouping (GRP_REG+OI_POI)	L.Ind.	SC	CB&MF&SLC	Stmt.
T61	Prioritization using heuristic REG_OI_POI with modification (MOD * REG+OI_POI)	L.Ind.	SC	CB&MF&SLC	Stmt.
T62	Graph theoretic framework for test case prioritization	L.Ind.	SC	GPH	Any
T63	Prioritization based on analysis of program structure	Proc.	CG	CB&MF&PS&FB	Mdl.
T64	Enhanced Bayesian network based approach	OO	SC	MF+CB+FB+BN	Prg.
T65	Historical value based approach for prioritization	L.Ind.	SC	HB	Func.
T66	Test case prioritization for multiple processing queue	L.Ind.	SC	CST	Any
T67	Configuration aware regression testing	L.Ind.	SC	CA	Cfg.
T68	Test case prioritization for varying requirement priorities and cost	L.Ind.	SC	RQ	Sys.
T69	totalCC (prioritize on coverage of blocks)	L.Ind.	SC	CB	Class
T70	totalBN (prioritize via Bayesian Networks)	L.Ind.	SC	O (BN)	Mhd.
T71	additionalCC (prioritize on coverage of blocks with feedback mechanism)	L.Ind.	SC	CB	Class
T72	additionalBN (prioritize via Bayesian Network with feedback mechanism)	L.Ind.	SC	O (BN)	Mhd.
T73	Prioritization for resource constrained environment using historical test performance data	L.Ind.	SC	HB	Prg.
T74	Factor orented requierement coverage based prioritization	L.Ind.	SC	RQ	Sys.
T75	Total CM-1 (CM=Coverage Model, total workflow	WB	SM	CB	WS

	branches)				
T76	Addtl CM-1 (Additional CM-1, cumulative workflow branch coverage)	WB	SM	CB	WS
T77	Total-CM2-Sum (total workflow and XRG branches)	WB	SM	CB	WS
T78	Addtl-CM2-Sum (cumulative workflow and XRG branches)	WB	SM	CB	WS
T79	Total-CM2-Refine (Total workflow branches, descending order of XRG branches to break tie)	WB	SM	CB	WS
T80	Addtl-CM2-Refine (Additional CM2 Refine)	WB	SM	CB	WS
T81	Total-CM3-Sum (total workflow branches, XRG branches and WSDL elements)	WB	SM	CB	WS
T82	Addtl-CM3-Sum (cumulative workflow, XRG and WSDL elements)	WB	SM	CB	WS
T83	Total-CM3-Refine (same as Total-CM2-Refine except descending order of WSDL elements to break tie)	WB	SM	CB	WS
T84	Addtl-CM3-Refine (Additional CM3 Refine)	WB	SM	CB	WS
T85	ART-st-maximin (Statement level, min dij = max {min dij})	L.Ind.	SC	CB	Sys.
T86	ART-st-maxavg (Statement level, avg dij = max {avg dij})	L.Ind.	SC	CB	Sys.
T87	ART-st-amxmax (Statement level, max dij = max {max dij})	L.Ind.	SC	CB	Sys.
T88	ART-fn-maximin (Function level, min dij = max {min dij})	L.Ind.	SC	CB	Sys.
T89	ART-fn-maxavg (Function level, avg dij = max {avg dij})	L.Ind.	SC	CB	Sys.
T90	ART-fn-amxmax (Function level, max dij = max {max dij})	L.Ind.	SC	CB	Sys.
T91	ART-br-maximin (Branch level, min dij = max {min dij})	L.Ind.	SC	CB	Sys.
T92	ART-br-maxavg (Branch level, avg dij = max {avg dij})	L.Ind.	SC	CB	Sys.
T93	ART-br-amxmax (Branch level, max dij = max {max dij})	L.Ind.	SC	CB	Sys.
T94	Reactive GRASP (Greedy Randomized Adaptive Search Procedures)	L.Ind.	SC	CB	Stmt.
T95	Total technique to	WB	SYS	CB+MF+CF+	WS

	prioritize test cases			DF	
T96	Additional technique to prioritize test cases	WB	SYS	CB+MF+CF+DF	WS
T97	Model based prioritization	L.Ind.	SM	MF	Tranc.
T98	Concern Based Prioritization	L.Ind.	SM	MF	Tranc.
T99	GA_hist (Genetic Algorithms and History based test case prioritization)	L.Ind.	SC	HB+GB+CST	Prg.
T 100	GELAITONS (Genetic aLgorithm bAsed Test suIte priOritizatioN Systems)	OO	SC	GB	Sys.
T 101	Test case prioritization using Failure Pursuit Sampling	L.Ind.	BF	O (DTB)	Sys.
T 102	Ascending-WSDL-tag coverage prioritization	WB	SC	CB	WS
T 103	Descending-WSDL-tag coverage prioritization	WB	SC	CB	WS
T 104	Ascending-WSDL-tag occurrence prioritization	WB	SC	CB	WS
T 105	Descending-WSDL-tag occurrence prioritization	WB	SC	CB	WS
T 106	GUI and web based test case prioritization	WB	SC	CB	Evt.

Language Type:	Proc - procedural, Bin.Code - binary code, L.Ind - language independent, COTS - COTS component based, WB - web designing or OO - object oriented.
Input Method:	SC – Source Code, BF – Binary Form, SM – System Model, SYS – System, CG – Call graph for program structure, R&SP - Requirements/Specifications.
Approach:	CB – Coverage Based, MF – Modification Based, RQ – Requirement Based, FB – Fault Based, HB – History Based, GB – Genetic Based, CP – Composite, O – Others.
Granularity	Stmt-Statement level, Func-function level, Blk-block of binary form, Mhd-method, Tranc-transition in system model, Sys-system level, Prg-program, Prs-process level, Evt-event, Cpt-component, File-file to be changed, SU-software units, WS-web service, Mdl-module, Cfg-configuration of the software system, Class-class level or any.

Table A4: Search Algorithms.

ID of search algorithm	Search Algorithm
A1	Hill Climbing
A2	Genetic Algorithm
A3	Greedy Algorithm
A4	Additional Greedy Algorithm
A5	2-Optimal greedy algorithm
A6	Simulated Annealing
A7	Greedy by ratio
A8	Greedy by value
A9	Greedy by weight