Journal of King Saud University – Computer and Information Sciences 33 (2021) 1041–1054

Contents lists available at ScienceDirect

Journal of King Saud University – Computer and Information Sciences

journal homepage: www.sciencedirect.com

A survey on different approaches for software test case prioritization

Rajendrani Mukherjee, K. Sridhar Patnaik *

Department of Computer Science and Engineering, Birla Institute of Technology, Mesra, Ranchi 835215, India

ARTICLE INFO

Article history:
Received 17 April 2018
Revised 26 July 2018
Accepted 4 September 2018
Available online 3 October 2018

Keywords:
Regression
Prioritization
Techniques
Program
Fault
Coverage

ABSTRACT

Testing is the process of evaluating a system by manual or automated means. While Regression Test Selection (RTS) discards test cases and Test Suite Minimization (TSM) shows diminution in fault detection rate, Test Case Prioritization (TCP) does not discard test cases. Test Case Prioritization techniques can be coverage or historical information based or model based. It can also be cost-time aware or requirement-risk aware. GUI/Web applications need special prioritization mechanism. In this paper, 90 scholarly articles ranging from 2001 to 2018 have been reviewed. We have explored IEEE, Wiley, ACM Library, Springer, Taylor & Francis and Elsevier database. We have also described each prioritization method with their findings and subject programs. This paper includes a chronological catalogue listing of the reviewed papers. We have framed three research questions which sum up the frequently used prioritization metrics, regularly used subject programs and the distribution of different prioritization techniques. To the best of our knowledge, this is the first review with a detail report of the last 18 years of TCP techniques. We hope this article will be beneficial for both beginners and seasoned professionals.

Contents

* Corresponding author.
E-mail address: kspatnaik@bitmesra.ac.in (K.S. Patnaik).
Peer review under responsibility of King Saud University.

Production and hosting by Elsevier

https://doi.org/10.1016/j.jksuci.2018.09.005
1319-1578/

1. Introduction

The main objective of testing is to make sure the deliverable software has no bugs. The testing phase of SDLC is an elaborate task in terms of cost, time and resource. In Test First approach (Erdogmus et al., 2005), test suites are built before the coding starts. A study involving undergraduate students showed that as developers write more tests per unit of programming, higher productivity and consistent quality is ensured (Erdogmus et al., 2005).

Regression testing ensures no new errors have been introduced in the changed application (Chittimalli and Harrold, 2009). However execution of all scheduled test cases is next to impossible because of time, cost, and resource constraint. If a VB application has 5 screens where each screen has 4 combo boxes and each combo box has 5 items then the application will yield $5 * 4 * 5 = 100$ tests. For large systems it is not viable to check every data or input combination. From this point, the concept of RTS (Regression Test Selection), TSM (Test Suite Minimization) and TCP (Test Case Prioritization) gained value among researchers. While test case selection discards several test cases (Rothermel and Harrold, 1996), test suite reduction shows remarkable loss of fault detection rate (Rothermel et al., 1998; Elbaum et al., 2003). Test Case Prioritization techniques do not discard test cases and run the most significant test cases first (Rothermel et al., 1998). TCP is an Improvement Testing mechanism (Juristo et al., 2004) which can be applied at initial phase of testing too.

The test case prioritization problem is addressed in several ways by several researchers. While Coverage aware prioritization techniques focus on maximizing coverage information (statement/branch/function coverage etc.), requirement and risk aware prioritization techniques proposes PORT (Prioritization of Requirements for Testing) (Srikanth et al., 2005). Even though severely criticized as test case selection mechanism, historical record of previous test run data (Kim and Porter, 2002) also helps in prioritizing test cases. Knapsack solvers (Alspaugh et al., 2007) or ILP (Integer Linear Programming) (Zhang et al., 2009a) help in prioritizing test cases in time constrained environment. If time constraint is not applied then the benefit of ordering test cases becomes negligible because TCP techniques have their own implementation overhead. As gathering execution information is a costly effort, Model based TCP is becoming popular nowadays (Korel et al., 2005). Event Driven Software (EDS) like GUI/Web based applications generate bulk amount of test cases from all possible combination of events (mouse click, menu open etc.) and need ordering of test case mechanism.

The ongoing importance of TCP techniques encouraged us to prepare this review article. The paper attempts to make the following contribution:

1. In this paper, we have reviewed 90 scholarly articles from 2001 to 2018 on TCP. As the review work by Yoo and Harman (2007) narrated 47 papers on TCP till 2009, we wanted to bridge the gap by increasing the time frame and paper numbers. The chosen papers in this review are well-sampled as we have considered more than 5 papers in each of the above mentioned prioritization technique.
2. This review paper also describes each prioritization mechanism along with their techniques, results, subject programs and shortcomings in detail. While Catal and Mishra (2013) built a systematic mapping study on TCP, it lacked the technical details of subject programs, results, shortcoming etc. To the best of our knowledge, this is the first detail-oriented review report of the last 18 years of TCP techniques.
3. We have also built a chronological catalogue at the end of each subsection. The catalogue will act as a summarized snapshot.
4. The final motivation of this review article is to help the beginners as well as the experts who are conducting research on TCP domain. Three research questions have been designed and we hope their findings are valuable. This review paper addresses the following RQs-
   (a) RQ1: Which metrics are used often for TCP?
   (b) RQ2: What are the frequently used subject programs in prioritization studies?
   (c) RQ3: Which prioritization methods are commonly explored and what are their proportion?

Our review is structured as follows. Section 2 discusses basic facts and recent trends in Testing. Section 3 presents the definition of Test Case Prioritization problem and its background with related work. Section 4 narrates our search strategy and analyzes the findings from Research Questions. Section 5 describes each of the prioritization methods with their shortlisted papers and chronological catalogue listing. Section 6 discusses the scope of future work and concludes our review.

2. Basics of testing – Facts and trends

Before getting into the details of test case prioritization, in this section we would like to summarize the necessity and types of testing along with some recent trends and developments in testing tools.

Apart from maintaining the quality of a product, testing also ensures proper working of the expanded version of a software. In today's competitive market, a software system should evolve to stay pertinent. As testing by intuition depends on individual performance, rigorous testing by a dedicated team of testing professional is always recommended. Insufficient testing causes missed faults which requires more cost to fix with a noticeable amount of rework. A defect detected at field costs 1000 times more than found in requirement analysis phase (Srikanth et al., 2009). Coders at Google make more than 20 changes per minute in the source code and 100 million test cases get executed per day (Thomas et al., 2014). A test suite is a collection of test cases. A test case is a triplet of input, program state and expected result. Test suites can be reused later and test cases actually determine the success and maturity of testing.

Now we would like to sum up several testing techniques. Functional testing techniques are considered as black box testing because they study the inputs and its corresponding outputs only. Equivalence class partitioning and Boundary value analysis are different forms of functional testing strategy. They do not need knowledge of internal source code. The main advantage of black box oriented technique is that it does not need to collect and store coverage information (Henard et al., 2016). In case of Control Flow testing, detail knowledge of the source code is required. The program flow is analyzed by selecting a set of paths (a code chain that goes from beginning to the end of a program). Statement Coverage, branch/decision coverage, MC/DC are different form of control flow testing strategy. Branch coverage is stronger than statement coverage. MC/DC is stronger than statement and branch coverage but

weaker than condition coverage. Data flow testing techniques also need knowledge of source code. Mutation testing involves producing a series of mutants. Faulty versions of programs are called mutants which are produced by altering program statements. A mutant is considered killed if the test suite can detect the fault. Scalability is an unsolvable issue for mutation testing as big programs can have significant number of mutants. Regression testing ensures that a program modification is not causing existing features to regress. Sometimes modifying a program or fixing a bug may create a new one. Random, Retest-all, Safe (Example- Dj vu, Test Tube, Textual differencing) are several regression testing techniques. The retest-all strategy is applicable for small programs but fails for big programs. A safe regression test selection technique chooses each of the test case that reveals at least one fault. As safe techniques require less time to run it is appropriate for large programs.

Nowadays, agile regression testing is becoming popular. While performing agile testing, test cases get executed after each sprint/delivery window. In Test First approach (Feldt et al., 2016), test suites are built first before the coding starts. Testing becomes an overhead activity if it starts only after coding is complete. Continuous testing overcomes this drawback (Saff and Ernst, 2003; Elbaum et al., 2014; Memon et al., 2017). In Continuous Testing, test scripts are run in between program changes. In case of incremental testing, testing is done very frequently and in between two updates. For batch testing processes, testing window is sufficiently stretched and usually performed at night.

We would also like to enlist some testing tools which made testing easier. Quality Center is a requirement and test management tool offered by Hewlett Packard. UFT (Unified Functional Testing) is another automated testing tool by HPE which performs regression testing for UI based/ non UI based test cases. It was formerly known as QTP (Quick Test Professional). JaBUTi (Java Bytecode Understanding and Testing) is a coverage analysis tool for Java applications. Jumble helps in measuring the coverage of JUnit tests. Jumble operates at test class level. Jester is another open source lightweight tool for testing Java applications which finds uncovered code. JMeter is an open source load testing tool which helps in monitoring performance. Selenium is a capture and replay based automation tool for testing web applications. Selenium provides a portable framework. WinRunner and LoadRunner are another testing tools designed by HPE. Seque's Silk Test and Silk Performer help in doing regression testing and web/mobile application testing respectively. All these testing tools have equipped testing techniques to find more bugs easily.

3. Test case prioritization – Background and related work

In this section we would like to narrate the origin and importance of test case prioritization along with a concise snapshot of several important related work in this field.

Regression testing accounts for 80% of the testing budget (Chittimalli and Harrold, 2009). Implementing changed as well as new requirements, revalidating the software, providing quick bug fix are important part of regression testing. Regression testing is very much time and resource constrained and it is encountered frequently. Regression testing of real time embedded system is heavily time constrained as their simulation environment is very demanding and hosts multiple projects (Kim and Porter, 2002). Retest-All, Regression Test Selection (RTS), Test Suite Minimization (TSM) and Test Case Prioritization (TCP) are the predominant regression testing techniques. Each of these approaches has their own advantages and disadvantages. The Retest-All strategy holds good when the test suite is small. However as test suite scales up, an ordering mechanism becomes necessary. Rothermel and Harrold (1996) discussed several regression test selection techniques. A safe regression test selection technique chooses each of the test case that reveals at least one fault but it still does not ensure safe selection because the criteria under which safety prevails does not always get fulfilled (Rothermel et al., 2001). Unsafe test case selection technique discards several test cases. Wong et al. (1998) showed that Test Suite Minimization shows very minimal decrease (2%–7%) in fault detection rate but several studies contradicted this fact (Elbaum et al., 2003; Rothermel et al., 1998). They argued that in many cases, Test Suite Reduction process ensures 100% coverage of requirements, but does not assure same fault detection ability as the actual test suite. Elbaum et al. (2003) showed remarkable loss (60%) of fault detection rate while running minimization techniques. Test Case Prioritization (TCP) overcomes these drawbacks of selection or reduction mechanism by not discarding test cases. According to TCP, test cases with higher priority gets executed earlier while conducting testing. If conducted offline, TCP will save time and cost and will not become an overhead. Test case prioritization can be of two types – general test case prioritization and version specific test case prioritization. In general test case prioritization, the prioritization ordering is useful for successive modified versions of a program. However for version specific test case prioritization, the ordering is beneficial for one particular version only.

Definition 1. For a test suite T discover T' in such a way that $f(T') \geqslant f(T'')$, where f is the quantifiable performance or goal function, PT is the set derived from all possible prioritization orderings of $T, T' \neq T'', T' \in PT$ and $T'' \in PT$.

Juristo et al. (2004) referred Test Case Prioritization as an Improvement Testing as it can be associated with any other technique to increase fault detection rate. In Test Case Prioritization, test cases are ordered based on some criteria. The goal of test case prioritization can be manifold. It can be like increasing fault detection rate or increasing capture of high priority requirements or decreasing the cost and time of prioritization mechanism. Even though test case prioritization is mainly applied for regression testing, it can also be applied for software maintenance or at initial phase of testing.

In this paper, we have reviewed 90 scholarly articles from 2001 to 2018 on Test Case Prioritization techniques. Several researchers have addressed test case prioritization problem in several ways. Section 5 of this paper narrates each publication with their applied methodology, subject programs, outcomes and shortcomings. Coverage aware prioritization techniques mainly aim at maximizing coverage as an intermediate goal while ultimate goal is improving APFD (Average Percentage of Faults Detected) rate. Historical information based TCP techniques utilizes historical record about previous run of test case or historical fault information. Cost centric prioritization schemes focuses on building cost models by subdividing cost into several parameters like cost of analysis, cost of maintenance, cost of execution etc. A new metric – $APFD_C$ is used which considers varying fault severity and test case cost. Time aware TCP techniques use knapsack solvers or ILP (Integer Linear Programming) to address the issue of time constraint in regression testing. As the main aim of testing is to find out whether the delivered product meets customer's requirement, RBT (Requirement Based Testing) is becoming popular. Clustering of requirements and selecting more test cases from higher priority clusters are also gaining attention. Risk Exposure based prioritization is yielding high fault detection rate. Stallbaum et al. (2008) generated test cases from activity diagrams and prioritized them based on the associated risks. Model based prioritization mechanism is another possibility of addressing TCP problem where specification models (event state diagram/activity diagram) represent system behaviour

and the model execution information is used to order test cases. TCP techniques for GUI/Web based Applications has the edge of using user session data as test data. Bryce and Memon (2007) proposed Interaction Coverage based TCP. Several studies conducted by Srivastava and Thiagarajan (2002), Nardo et al. (2015) proved that TCP techniques scale well for real world systems of million LOC.

In the next section, we would like to explore some Research Questions regarding Test Case Prioritization. The RQs will help the researchers to gain some important insights about the frequently used metric for prioritization, commonly used subject programs for study, prominent researchers in this field etc.

4. Research methodology

4.1. Search strategy

We have searched for papers in the following repository: IEEE Explorer, Wiley Online Library, ACM Digital Library, Springer, Taylor & Francis, Elsevier. We have also selected conference and workshop proceedings. Even though initial search retrieved many studies, after examining the title, abstract, citations and relevancy we have narrowed down our chosen papers to 90. All the 90 papers are read in full length before preparing this review. The sample of 90 chosen papers is rich in diversity as it considers more than 5 papers in each category of prioritization scheme – Coverage aware TCP, Historical information based TCP, Time–Cost centric TCP, Model based TCP, Requirement-Risk aware TCP, GUI/Web Application focused TCP etc.

Inclusion and Exclusion Criteria:

We came across 3 review papers (Catal and Mishra, 2013; Yoo and Harman, 2007; Hao et al., 2016a) on TCP before this. The review work of Yoo and Harman considered 47 papers till 2007 (Yoo and Harman, 2007). We wanted to expand the horizon by selecting 90 papers till 2018. The systematic mapping study done by Catal and Mishra in 2013 did not explore the technical details of each prioritization methodology (Catal and Mishra, 2013). We have tried to explain each prioritization method with their techniques, outcome and subject programs. The chronological catalogue listing of the selected papers at the end of each subsection is another added feature of this review work. To the best of our knowledge, this is the first review work with a summary report of the last 18 years of TCP techniques. Papers narrating test case generation, test case execution are excluded.

Research Questions:

We have also framed 3 research questions. The findings of these RQs will make the review useful for expert as well as beginners who are willing to conduct research on TCP techniques. Table 1 depicts the RQs with their benefit.

Table 1
Research Questions and Motivation.

Research Questions	Benefit from Findings
RQ1: Which metrics are used often for TCP?	This RQ will help in rebuilding the existing metrics while proposing new one.
RQ2: What are the frequently used subject programs in prioritization studies?	New researchers in this field will find it easier to replicate the existing studies with these subject programs. If a new method is proposed it can also be verified using these subjects.
RQ3: Which prioritization methods are commonly explored and what are their proportion?	This RQ will give an idea about which techniques can be explored more.

4.2. RQ1 – Which metrics are used often for TCP?

Average Percentage of Fault Detected (APFD) is the most dominant metric for addressing prioritization. Catal and Mishra (2013) have stated that 34% of the chosen papers have used APFD metric. However apart from APFD, researchers have proposed Savings Factor (SF), Coverage Effectiveness (CE), – $APFD_C$ (APFD per cost), NAPFD (Normalized APFD) etc. for measuring the effectiveness of prioritization mechanism. Table 2 tabulates several useful metrics.

4.3. RQ2 – What are the frequently used subject programs in prioritization studies?

This RQ investigates commonly used subjects (programs, applications) for experimentation of TCP techniques. We have come across several C/JAVA programs which are repeatedly used for TCP studies. Certain case study applications or web based applications are also chosen recurrently. Table 3 enlists the findings for RQ2.

4.4. RQ3 – Which prioritization methods are commonly explored and what are their proportion?

There is a wide variety of available prioritization techniques. The aim of this research question is to categorize them and find out their proportion. We have chosen 90 publications from last 18 years (2001–2018). Section 5 of this paper details each of this prioritization scheme.

1. Coverage aware methods are most prevalent (10%) while requirement based methods (7.77%) are second best one.
2. Time, cost and historical information based TCP techniques are also becoming popular (5.5% in each cases).
3. 4.4% of the chosen papers involved real time systems.
4. Search algorithms based ordering is explored in 4.4% and risk aware ranking is observed in 3.3% of the total selected papers.

Table 2
Metrics Used by Several Prioritization Studies.

Abbreviation	Full Form	Explanation
APFD	Average Percentage of Fault Detected	APFD Value ranges between 0 to 100 where higher value indicates better fault detection.
SF	Savings Factor	SF translates APFD into benefit scale. SF 1000 indicates 1% of APFD gain creates a savings of 1000 dollars.
CE	Coverage Effectiveness	CE value ranges between 0 to 1. CE considers the cost and coverage of individual test case. If test requirements are fulfilled then a high CE is achieved.
CI	Convergence Index	CI helps in deciding when to stop testing.
$APFD_C$	APFD per cost	Units of fault severity detected per unit of test cost is calculated.
NAPFD	Normalized APFD	This metric considers both fault detection and time of detection.
ASFD	Average Severity of Faults Detected	ASFD for requirement i is the ratio of the summation of severity values of faults detected for that requirement divided by TSFD (Total Severity of Fault Detected).
RP	Most Likely Relative Position	An average relative position of the first failed test case that finds a defect is used. This metric is primarily used for model based TCP.
APDP	Average Percentage of Damage Prevented	APDP is used to measure the effectiveness of Risk Based Test Case Derivation and Prioritization.

Table 3
Subject Programs Used by Several Prioritization Studies.

Program/Application Name	Language/Platform
tcas (138LOC), schedule2 (297LOC), schedule (299 LOC), tot_info (346 LOC), print_tokens (402 LOC), print_tokens2 (483 LOC), replace (516 LOC), space (6218 LOC)	These are 8 C Programs from SIR (Software Artifact Infrastructure Repository) which is publicly available. First 7 are called Siemens Program and the eight program is a program from European Space Agency.
grep (7451 LOC), flex (9153 LOC), sed (10 KLOC),	UNIX utility program
ant (80.4 KLOC), jmeter (43.4 KLOC), xml-sec (16.3 KLOC), jtopas (5.4 KLOC),	JAVA programs
Siena ($\geq$ 2.2 KLOC)	JAVA program
Bash ($\geq$ 50 KLOC)	A shell program that provides interface to Unix services
Empire (63 KLOC)	Implemented in C. Open source software – a game played between several players.
GradeBook, JDepend	Case Study Application. Performs grading and Generates metrics.
TerpOffice ($\geq$ 90 KLOC)	Includes TerpWord, TerpPaint, TerpCalc etc.

5. Even though coverage aware methods are dominant, 2.2% of the papers researched algorithms without coverage information.
6. If one technique is used in only one paper, then that technique is shown in others category.

Fig. 1 charts the distribution of several TCP techniques.

We have also charted a year wise distribution of papers in Fig. 2. Test case prioritization studies gained peak attention from 2007 to 2010. It also surged during 2016. After seeing the year wise distribution we can conclude that interest of researchers in test case prioritization is very much active.

5. Review of different approaches for test case prioritization

In this section we would like to highlight 90 prominent research publications from each avenues of test case prioritization. Test case prioritization techniques can be of several approaches -.

1. Coverage based Prioritization
2. Historical Information based Prioritization
3. Cost Aware Prioritization
4. Time Aware Prioritization

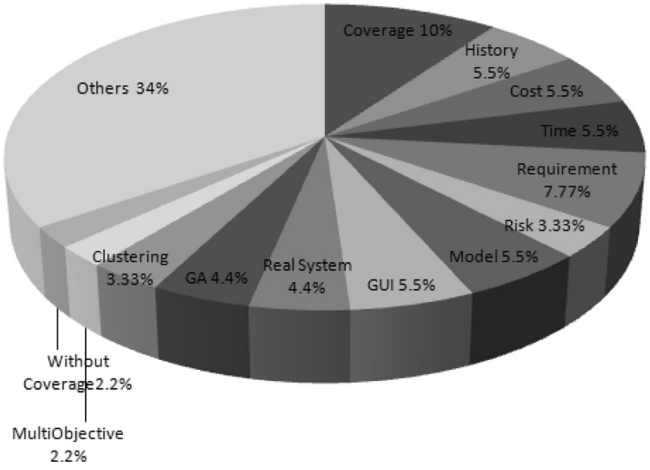

Fig. 1. Distribution of Prioritization Techniques.

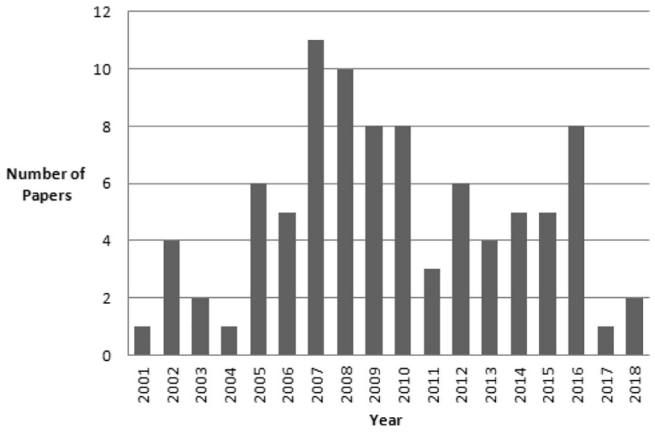

Fig. 2. Year wise Paper Publications on Test Case Prioritization.

5. Requirement and Risk Aware Prioritization
6. Model based Prioritization
7. GUI/Web Application focused Prioritization
8. Other Special Approaches (Ant Colony Optimization Techniques, Search Algorithms, Clustering Techniques etc.)

For each publication, methods applied for ordering test cases, subject programs of experimentation and the outcome along with shortcomings are detailed in a catalogue listing in chronological order. The catalogue will also act as a summarized snapshot of the discussed publications. A separate subsection is devoted for prioritization techniques applied on real world/industrial case studies.

5.1. Coverage aware test case prioritization techniques

Coverage aware prioritization techniques aim to maximize coverage of program elements (statement/ branch/ methods etc.) by a test case and need detail knowledge of source code. It basically follows white box/structural testing approach (Fig. 3). Achieving maximum coverage may be visualized as an intermediate goal while the ultimate goal is enhanced fault detection rate.

Now we would like to highlight 9 research papers which have added a new dimension to code coverage based prioritization techniques. Coverage based test case prioritization techniques came into attention in 2001 as Rothermel et al. showed that better coverage yields better fault detection rate (Rothermel et al., 2001). Nine prioritization techniques were applied on eight C programs from SIR (Software Artifact Infrastructure Repository). The most insightful outcome of this study revealed that for the first seven programs (called as Siemens Program) the total techniques outperformed the additional technique but for the eighth program, Space, which is a real and huge program from European Space agency, the additional techniques performed better. The additional techniques are called feedback or greedy strategy as they iteratively first selects a test case of highest coverage, then adjusts the coverage details of the remaining test cases for elements (statement/branch etc) not yet covered. The shortcoming of this study was that it considered all faults of same severity and did not incorporate cost factor. In continuation with Rothermel's study, Elbaum et al. conducted another empirical assessment which classifies the prioritization techniques as coarse granularity (function level techniques) and fine granularity (statement level techniques) (Elbaum et al., 2002). The concept of SF (Savings Factor) was introduced in this work. SF is a metric that translates APFD measure into benefit scale. SF 1000 indicates 1% APFD gain creates a savings of 1000 dollars. The study concludes that the fine granularity tech-

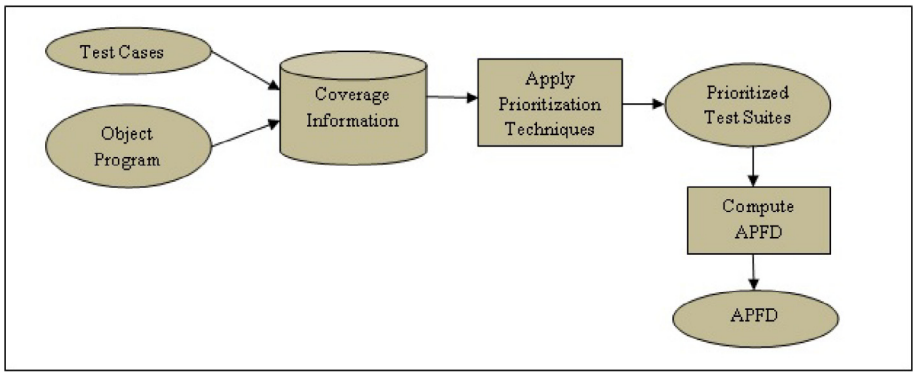

Fig. 3. Methodology of Coverage Aware TCP Technique.

niques outperforms the coarse granularity techniques by a very little edge. This marginal gain is not beneficial as for larger systems statement level techniques are too tedious and expensive while function level techniques are less expensive. Jones and Harrold (2001) for the first time applied Test Case Prioritization techniques on MC/DC test suite. The authors proposed a build up technique (test cases are getting added to an empty test suite) and a break down technique (removing low contribution test cases) by identifying essential and redundant test cases.

All of the previous studies mainly used C programs for implementing the prioritization prototypes. We would like to bring up the work conducted by Do et al. (2006) which for the first time focused on prioritizing JUnit test cases. For JUnit framework, test cases were categorized into two segments – test cases at test class stage and test cases at test method stage. Apart from analyzing JUnit test cases for the first time, this study also added a new viewpoint in terms of time and money. If the prioritization method involves feedback strategy then the cost of prioritization increases. However this study neither considered other test suites nor it incorporated object oriented features (inheritance, encapsulation, polymorphism etc.) while testing Java programs. We would like to mention another experimentation executed by Do and Rothermel (2006) which studied the same open source Java Programs of the previous work with mutation faults. This study eliminated the shortcoming of the previous work by considering TSL (Test Specification Language) test suite. In another research conducted in 2007, Kapfhammer and Soffa, introduced a metric CE (Coverage Effectiveness) for ordering test suites. If test requirements are fulfilled quickly then a high CE is achieved (Kapfhammer and Soffa, 2007). Fang et al. (2012) executed another experimentation which focused on the term logic coverage testing. Branch coverage, modified condition/decision coverage are several logic coverage testing method. Logic coverage is widely accepted for safety – critical software. The authors coined the term CI (Convergence Index) which is used to decide when to stop testing. A quick convergence indicates low testing cost and rapid fault detection rate.

In almost all the previous studies, the authors mentioned about significant performance margin between prioritized orderings and optimal orderings. However, we would like to highlight a recent work by Hao et al. (2016a) which implemented Optimal coverage based prioritization technique by ILP (Integer Linear Programming) and ruled out that fact. The study concluded that the Optimal technique is notably worse than additional technique in terms of fault detection rate or execution time. Table 4 represents a concise snapshot of the above papers.

Though there is a logical establishment that greater coverage indicates greater effectiveness, coverage is just one parameter for prioritizing test cases. A maximum coverage does not always ensure that all faults will be covered. As the main goal of testing is to ensure the delivery of a quality product within a stipulated time and budget, TCP techniques should be time, cost and requirement centric. Leon and Podgurski (2003) compared the differences between coverage based and distribution based technologies (how the execution profile of test cases are distributed) and proved that distribution based techniques may be more efficient than coverage

Table 4
Summary of Publications on Coverage Aware Prioritization Techniques.

Authors and Year	Prioritization Technique	Results	Subject Program
Rothermel et al. (2001)	Nine prioritization techniques (total/ additional etc.) were introduced.	Prioritization of test cases help in increasing APFD.	8 C programs (Siemens and Space program)
Jones and Harrold (2001)	Prioritization techniques were applied on MC/DC test suite.	For larger systems build up technique is beneficial as it requires less time.	Tcas, Space
Elbaum et al. (2002)	Concept of coarse granularity (function level), fine granularity (statement level) were introduced.	Fine granularity techniques outperforms the coarse granularity techniques by a very little edge.	Real time embedded system QTB, UNIX utilities flex and grep
Do et al. (2006)	Coverage based techniques were applied for JUnit test cases for the first time.	In some cases (ant and xml-security) non-control techniques performed better.	Four Java programs – ant, jmeter, xml-security and jtopas
Do and Rothermel (2006)	Use of Mutation faults were studied.	For ant the non-control techniques performed better while for jtopas it showed no improvement.	Two Java programs- galileo and nanoxml in TSL test suite and Four Java programs ant, xml-security, jmeter, jtopas in JUnit
Kapfhammer and Soffa (2007)	Coverage Effectiveness was used to rank test cases.	A high CE is achieved if test requirements are fulfilled.	
Fang et al. (2012)	The concept of logic coverage and convergence index were introduced.	Quick convergence is necessary for low cost and rapid fault detection.	3 Java Programs – Tcas, NanoXML, Ant
Hao et al. (2016b)	ILP was used to implement Optimal Coverage based prioritization.	Optimal method is worse than additional technique.	Siemens and Space program, 2 Java programs (jtopas and siena)

based techniques. Several Event Driven Software (GUI/Web Applications) and real world case studies from industry also have special challenges for prioritizing test cases. All these understanding has forced the test case prioritization researchers explore other options. In the subsequent subsections, we would be narrating other test case prioritization schemes – history based TCP, time/cost aware TCP, model based TCP, requirement based TCP etc. one by one.

5.2. Historical information based test case prioritization techniques

Most of the proposed prioritization techniques are memory less. However as regression testing is not one time activity some kind of memory function should be associated with it. In this subsection, we would like to highlight 5 research publications which consider test case execution history.

Kim and Porter (2002) proposed a TCP method using historical information about test case performance record. The authors assign a selection probability for each test case which is described in Table 5. The concept of fault age was also introduced by this work. If a fault remains undetected by a test run, then the fault age increases. However this technique is strongly criticized by many as it is essentially a test case selection mechanism.

Park et al. (2008) criticized the existing TCP techniques as value-neutral. The existing methods consider all faults of same severity and all test cases of same test cost. However this is not the real case. The authors introduced a Historical Value-Based technique with a cost centric approach. A historical information repository module was used to keep record of test case's previous cost and fault severity data. Fazlalizadeh et al. (2009) mentioned that finding an optimal execution order of test cases does not have any deterministic solution. Kim and Baik (2010) proposed FATCP (Fault Aware Test Case Prioritization) that utilized historical fault information by using fault localization technique. Fault Localization Technique is actually a debugging activity which points out the location of a fault or a set of faults in a program. FATCP outperformed branch coverage and statement coverage prioritization methods for most of the Siemens program. Huang et al. (2012) proposed MCCTCP (Modified Cost Cognizant TCP) based on test case execution history. MCCTCP does not need analysis of source code and it feeds historical information of each test case to a GA (genetic algorithm). Table 6 represents a concise snapshot of the above papers.

Table 5
Test case Selection Parameters.

Variables	Description
T_C	Test Case
T	Time instant
H_{tc}	Time ordered observations $\{h_1, h_2, \ldots, h_t\}$ obtained from previous run
α	Weight factor for Individual Observation • Lower value indicates older observation • Higher value indicates recent observation
$P_{tc}, t(H_{tc}, \alpha)$	Selection Probability of each test case

5.3. Cost cognizant test case prioritization techniques

While discussing coverage aware prioritization prototypes, we have talked about Savings Factor (SF). Determination of SF is basically an attempt to convert APFD into a benefit scale. Even though test case prioritization may cause savings, it has its own execution/implementation overhead. Building a cost model is henceforth necessary and in this subsection we would like to focus on 5 publications which calculates cost benefit tradeoffs for TCP techniques.

Malishevsky et al. (2002) subdivided cost into several sub parameters. Table 7 shows the proposed cost categorization.

Another study conducted by Elbaum et al. (2004) introduced the concept of cost-benefit threshold which indicates a borderline percentage which needs to be crossed in order to produce a positive/effective APFD gain. Malishevsky et al. (2006) proposed a cost aware TCP technique that overcomes the shortcomings of APFD metric. They introduced –$APFD_c$ which considers varying fault severity and test case cost. This new metric is termed as 'units-of-fault-severity-detected-per-unit-of-test-cost'. While plotting –$APFD_c$, the x-axis indicates percentage of total test case cost incurred instead of percentage of test suite executed (as in APFD). The y-axis indicates percentage of total fault severity detected instead of percentage of faults detected. We would like to bring up another study conducted by Smith and Kapfhammer (2009). The experimentation included 8 real world case study applications to show the effect of cost on building smaller and faster fault detection worthy test suite. Testing system configurable software was experimented by Srikanth et al. (2009) and they have shown that cost of configuration and set up time plays an important role. Table 8 represents a concise snapshot of the above papers.

Table 7
Cost parameters for Prioritizing Test Cases.

Cost Variable	Explanation
$Ca(T)$	Cost of Analysis
$Cm(T)$	Cost of Maintenance
$Ce(T)$	Cost of Execution
$Cs(T)$	Cost of Selection
$Cc(T)$	Cost of Result Checking
$Cf(F(T) \setminus F(T'))$	Cost of missing faults by not selecting T∩T′ where T′ is the Selected Test Suite (set difference – set of elements present in F(T) but not in F(T′))
$Cf(F_k(T) \setminus F_k(T'))$	Cost of omitting faults where $F_k(x)$ is the set of regression faults on version v_k detected by Test Suite x

Table 6
Summary of Publications on History based Prioritization Techniques.

Authors and Year	Prioritization Technique	Results	Subject Program
Kim and Porter (2002)	A TCP method using historical information about test case was proposed.	The authors assign a selection probability for each test case.	8 C Program (Siemens and Space)
Park et al. (2008)	A Historical Value-Based technique with a cost centric approach was proposed.	Test case execution cost and fault severity are important factor.	8 versions of open source Java program ant.
Fazlalizadeh et al. (2009)	The priority of a test case in previous regression test session was considered.	Optimal execution of test cases do not have deterministic solution.	8 C Program (Siemens and Space)
Kim and Baik (2010)	FATCP (Fault Aware Test Case Prioritization) was used.	FATCP outperformed branch coverage and statement coverage method.	8 C Program (Siemens and Space)
Huang et al. (2012)	MCCTCP (Modified Cost Cognizant TCP) was utilized for prioritization.	Improvement in fault detection was observed.	Two UNIX utilitis • sed • flex

Table 8
Summary of Publications on Cost Cognizant Prioritization Techniques.

Authors and Year	Prioritization Technique	Results	Subject Program
Malishevsky et al. (2002)	Cost was subdivided into several sub parameters.	Equations for calculating cost for selection, reduction, and prioritization mechanism was built.	bash (size > 50 KLOC)
Elbaum et al. (2004)	The concept of cost-benefit threshold was introduced.	The preferred technique changes as the cost benefit threshold increases.	8 C programs (bash, emp-server, sed, xearth, grep, flex, make, gzip)
Malishevsky et al. (2006)	The authors introduced $APFD_c$ which considers varying fault severity and test case cost.	In 74.6% of cases, $APFD_c$ based method showed a worse rate of fault detection.	Empire (a game played between several players)
Smith and Kapfhammer (2009)	The concept of cumulative coverage function was introduced.	Cost has effect on building smaller and faster fault detection worthy test suite.	8 real world case study applications were studied.
Srikanth et al. (2009)	Cost of configuration and set up time plays an important role.	History of escaped faults help in prioritizing system configurations.	Two releases of a large legacy system were studied.

5.4. Time aware test case prioritization techniques

Regression testing has severe time and resource constraints. Not only for regression testing, nightly build and batch techniques are also severely time pressed. Extreme programming techniques require frequent and fast execution of test cases. We would like to narrate 5 publications which explored time aware prioritization with an application of Genetic Algorithms or ILP (Integer Linear Programming) or Knapsack solvers.

Time aware TCP techniques were first introduced by Walcott et al. (2006). This is the first study which concretely incorporates testing time budget. GAPrioritize is the proposed Genetic Algorithm which identifies the tuple with maximum fitness. The drawback of this study is that it considers each test case to be independent and have no execution ordering dependency. Alspaugh et al. (2007) showed the usage of 0/1 knapsack solvers to finish the testing activity within a stipulated amount of time. A knapsack with maximum fixed capacity (maximum time limit for executing the test suite) should contain distinct items (test cases) each with its own value (percentage of code coverage of each test case) and weight (execution time of each test case). Zhang et al. (2009a) used ILP (Integer Linear Programming) Techniques for prioritizing test cases in time-constrained environment. This is the first attempt to apply ILP for time-aware TCP. The only drawback of ILP based approach is that as it requires more analysis time it appears time consuming for large test suite. Do and Mirarab (2010) cited an example of a software development organization which has a regression test suite of 30,000 test cases. It takes 1000 machine hours to run all the test cases. Apart from execution time, significant time is needed for setting up test bed, monitoring results etc. The authors raised an important fact that if no time constraint is placed, prioritization becomes non cost effective. An economic model EVOMO (EVOlution-aware economic MOdel) was used for analyzing several cost factors. Effect of individual test case cost was explored by You et al. (2011). Five time budgets (5%, 25%, 50%, 75% and 100% of execution time of entire test suite) were set and the authors concluded that although it is slightly better to track individual test case cost, the benefit is marginal. Table 9 represents a concise snapshot of the above papers.

5.5. Requirement and risk oriented test case prioritization techniques

Mogyorodi (2001) of Starbase Corporation, overviewed Requirement Based Testing (RBT) process with CaliberRBT which can design minimum number of test cases from requirements. The author has stated that a problem was presented with 137,438,953,472 test cases and CaliberRBT solved the problem with only 22 test cases. The cost to fix an error is lowest if it is found in the requirements phase. The distribution of bug report indicates that 56% of all bugs are rooted in the requirements phase while design and coding phase yields 27% and 7% respectively (Mogyorodi, 2001). RBT helps in conducting testing in parallel with development. In this case, testing is not a bottleneck. A study conducted by Uusitalo et al. (2008) showed that testing and requirements engineering are strongly linked. In this subsection, we would like to focus on 7 key publications which analyzed requirement based TCP techniques.

The benefits of requirement based test case prioritization was explored since 2005 through several studies. If all requirements are given equal importance then a value neutral approach gets created. To overcome this neutrality, the authors designed PORT (Prioritization of Requirements for Testing) which is value driven and system level prioritization scheme (Srikanth and Williams, 2005). PORT considers four factors – CP (Customer Assigned Priority), IC (Requirement Implementation Complexity), RV (Requirement Volatility) and FP (Fault Proneness) of requirements (Fig. 4). CP is a measure of significance of requirement from customer's business value point of view and RV indicates how many times the requirement has changed. For industrial projects RV is high while for stable projects RV is low. Implementation Complexity ranges from 1 to 10 while larger value indicates higher complexity. FP includes both number of field failures and developmental failures while

Table 9
Summary of Publications on Time Aware Prioritization Techniques.

Authors and Year	Prioritization Technique	Results	Subject Program
Walcott et al. (2006)	The first study which includes testing time budget.	GA based algorithms performed extremely well (up to 120% improvement).	Two case study applications JDepend and Gradebook
Alspaugh et al. (2007)	Knapsack solvers were used	Testing activity can be finished within a stipulated amount of time.	JDepend and Gradebook
Zhang et al. (2009a)	ILP was used to prioritize test cases.	All the ILP based techniques outperformed GA based techniques.	JDepend and JTopas
Do and Mirarab (2010)	EVOMO (EVOlution-aware economic MOdel) was proposed.	Test case prioritization becomes non cost effective if there is no time constraint.	Java programs – ant, xml-security, jmeter, nanoxml, and galileo
You et al. (2011)	Effect of individual test case cost was explored.	Benefit of individual test case cost tracking is marginal.	Siemens and Space program

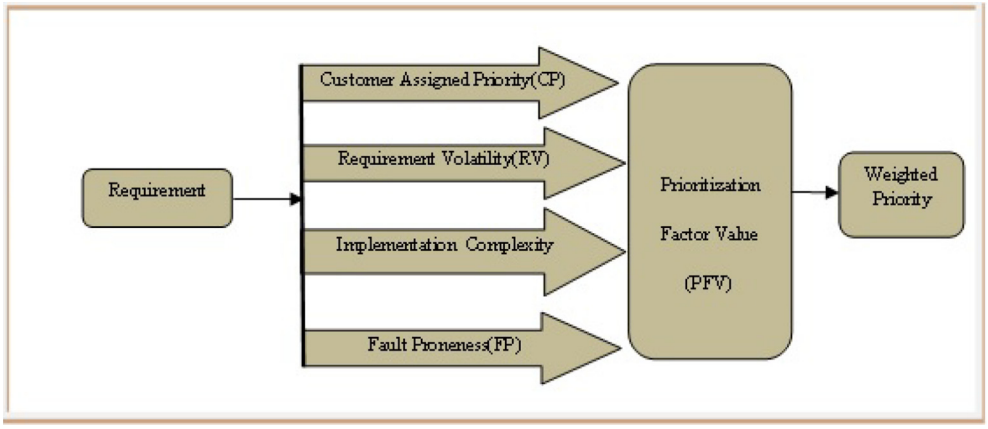

Fig. 4. Prioritization of Requirements for Testing (PORT).

coding a requirement. Based on these 4 values, PFV (Prioritization Factor Value) is computed for each requirement and it is used to derive Weighted Priority (WP) of each associated test cases.

Zhang et al. (2007) presented a metric 'units-of-testing-require ment-priority-satisfied-per-unit-test-case-cost'. As testing requirement priority changes frequently and test case costs also vary this metric becomes necessary. Another study conducted by Krishnamoorthi and Sahaaya (2009) proposed two more factors, Completeness and Traceability, as regression test case factors. The theory of generation of test cases from requirements with the help of GSE (Genetic Software Engineering) was projected by Salem and Hassan (2011). GSE uses a visual semi-formal notation called BT (Behaviour Tree) to model the requirements. Arafeen and Do (2013) evaluated whether test cases can be clustered based on similarities found in requirement. The requirement clusters are prioritized and more test cases are selected from higher priority clusters. Table 10 represents a concise snapshot of the above papers.

Next, we would like to discuss some risk management based prioritization strategies. Evaluating risk helps in early damage prevention. Stallbaum et al. (2008) proposed an automatic technique RiteDAP (Risk Based Test Case Derivation And Prioritization) which generates test cases from Activity Diagrams and then prioritizes them based on associated risk. Srivastava (2008) mentioned that risk prone software elements should be tested earlier. A new technique was structured by Yoon (2012) that measures risk exposure values of different risk items which originated from each requirements and then prioritizes test cases based on that. Table 11 represents a concise snapshot of the above papers.

5.6. Model based test case prioritization techniques

Gathering execution information is a costly effort in terms of time, money and resource. Also as source code changes with respect to new requirements, execution information needs timely upgrades making maintenance difficult. TCP techniques based on

Table 10
Summary of Publications on Requirement Aware Prioritization Techniques.

Authors and Year	Prioritization Technique	Results	Subject Program
Srikanth et al. (2005), Srikanth and Williams (2005), Srikanth et al. (2013)	The authors designed PORT (Prioritization of Requirements for Testing).	80% of defects got revealed in first 3 weeks.	JAVA projects and an industrial case study from IBM
Zhang et al. (2007)	Changing testing requirement priority and varying test case costs were considered.	New metric 'units-of-testing-requirement-priority-satisfied-per-unit-test-case-cost' was built.	A series of simulation experiments were performed.
Krishnamoorthi and Sahaaya (2009)	A new system level TCP technique similar to PORT was designed.	Two more factors, Completeness and Traceability, were proposed.	5 J2EE application projects of size approximately 6000 LOC
Salem and Hassan (2011)	Test cases were generated from requirements with the help of GSE.	Genetic Software Engineering was used to model the requirements.	Microwave oven case study
Arafeen and Do (2013)	Test cases were clustered based on similarities found in requirement.	More test cases are selected from higher priority clusters.	Capstone (online examination) and iTrust (medical record keeper)

Table 11
Summary of Publications on Risk Aware Prioritization Techniques.

Authors and Year	Prioritization Technique	Results	Subject Program
Stallbaum et al. (2008)	RiteDAP (Risk Based Test Case Derivation And Prioritization).	APDP (Average Percentage of Damage Prevented) was designed.	Program flow chart of the income tax calculation
Srivastva et al. (2008)	Both requirement priority and risk exposure value were considered.	Risk prone software elements should be tested earlier.	Sample case study to count the frequency of a word in a file
Yoon (2012)	RE (Risk Exposure) based prioritization	RE (Risk Exposure) based prioritization yielded 94% APFD.	All Siemens program

models gain interest in such cases. Model based TCP is basically a grey box oriented approach where specification models (state diagrams/activity diagrams etc) are utilized to represent expected behaviour of the system. Each test case is linked to an execution path in the model. The term grey-box is used as the information about the internal data structure and architecture of the system is required but the source code is not required. As execution of model is fast compared to actual system, model based TCP is a profitable option. Model based testing (MBT) is also gaining interest as an test automation approach. In this subsection, we would like to highlight 5 noteworthy publications which focused on model based test case prioritization.

Korel et al. (2005) focused on MBT (Model based testing) for system models. System models help in understanding system's behaviour. State based models are executed with a test suite and the execution information is used to prioritize tests. The authors use EFSM (Extended Finite State Machine) as the modelling language which consists of states and transitions between states. Three system models (ATM model, Cruise Control model and Fuel Pump model) averaging 7 to 13 states and 20 to 28 transitions were experimented and results indicated promising improvement in prioritization. Faults were seeded in the models. A very important conclusion came from the study that monitoring only modified transitions is not that effective, deleted transitions should also be taken care of. Five heuristics were formulated by Korel et al. (2007) for model based testing. A model based TCP technique taking into account several object oriented features like inheritance, polymorphism, aggregation etc. was proposed by Panigrahi and Mall (2010). The authors proposed EOSDG (Extended Object Oriented System Dependence Graph). Fig. 5 indicates different steps of this technique.

A similarity based test case selection approach was proposed by Hemmati et al. (2013). In similarity based selection it is hypothesized that more diverse test cases have higher fault revealing ability. Table 12 represents a concise snapshot of the above papers.

5.7. Test case prioritization techniques for GUI/Web based applications

In the previous subsections, we have portrayed several strategies for Test Case Prioritization for C and JAVA programs. However, increasing usage of internet is making reliable web applications on demand. Web applications run on web server and consists of several static (same content for all users) and dynamic (content depends on user input) web pages. While testing web applications, sequence of events (clicking a button, opening a menu etc.) which are performed by users are recorded. Web applications and Graphical User Interface are examples of EDS (Event Driven Software). Large number of possible combination of events creates huge number of test cases for EDS and poses a challenge. Web application testing has the advantage that user session data can be recorded and used as test data. In this subsection, we narrated 5 research publications which explored prioritization of Web application/GUI application test cases.

Memon and Xie (2005) proposed a framework called DART (Daily Automated Regression Tester) that retests GUI applications frequently. An office suite TerpOffice was developped by undergraduate students of University of Maryland and was used as a subject for testing DART. TerpOffice has TerpWord, TerpSpreadSheet, TerpPaint, TerpCalc and TerpPresent. Other than TerpSpreadSheet, all the subjects showed large number of fault detection with DART. The authors suggested a future work which will make DART execute certain uncovered part of the code. User session based testing of web applications was introduced by Sprenkle et al. (2005). Converting usage data into test cases is called user session based testing. The authors proposed an approach called Concept which clusters user sessions that depicts similar use cases. Bryce and Memon (2007) proposed TCP by interaction coverage. An example of a pair wise interaction for a library system may be – {Member Type = Student, Discount Status = High}. Sampath et al. (2008) proposed several prioritization

Table 13
Test Case Interaction with Web pages.

Test Cases	Login.jsp	Search.jsp	Select.jsp	Order.jsp	Payment.jsp
T1	X	X			
T2		X	X	X	
T3	X		X	X	X
T4		X			

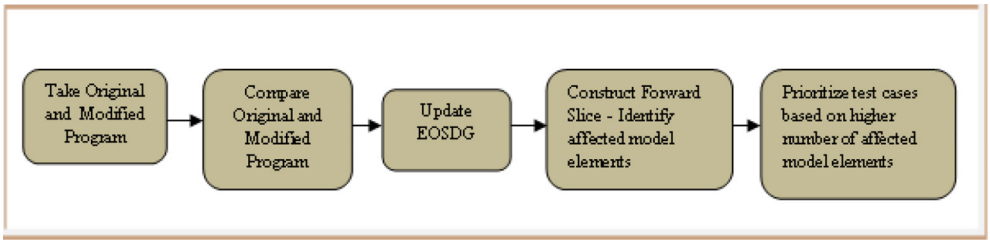

Fig. 5. Model based TCP technique using EOSDG.

Table 12
Summary of Publications on Model based Prioritization Techniques.

Authors and Year	Prioritization Technique	Results	Subject Program
Korel et al. (2005)	Selective prioritization and model dependence based prioritization were proposed.	Modified and Deleted transitions should be taken care of.	Three system models (ATM model, Cruise Control model and Fuel Pump model)
Korel et al. (2007, 2008)	Several heuristics were formulated for MBT.	Transition frequency or number may not have positive influence in early fault detection.	Two systems – ISDN and TCP-Dialler.
Panigrahi and Mall (2010, 2014)	Object oriented features were taken into account.	Almost 30% improvement in bug detection	ATM, Library System, Elevator Controller and Vending Machine
Hemmati et al. (2013)	A similarity based test case selection approach was followed.	Test case diversity increases scalability of MBT.	Subsystem of a video conference system, subsystem of a safety critical system

strategies (frequency of appearance, coverage of parameter values etc.) after evaluating them with three web applications. Table 13 shows the prioritization order (T3-T2-T1-T4) based on length of interaction. T3 covers four interactions while T4 covers only one.

All the previous work considered GUI and web based application as separate object of study. Bryce et al. (2011) proposed methods for testing web applications and GUI applications together. Table 14 represents a concise snapshot of the above papers.

5.8. TCP techniques applied on real world systems

While most of the conducted studies in the TCP domain are based on seeded faults so far, it is necessary to study the effect of TCP on real faults. Seeded faults are easier to inject and may be available in large numbers while real regression faults might be handful and tough to locate. Elbaum et al. (2002) studied real time embedded system QTB. However additional studies need to be done for checking the applicability of TCP techniques in real domain. We have come across 4 more remarkable work which studied real time systems with real faults.

Srivastava and Thiagarajan (2002) proved that TCP works for large systems (1.8 million LOC) by building Echelon. Haymar and Hla (2008) used PSO (Particle Swarm Optimization) technique which prioritizes test cases based on their new best position in the test suite. The authors applied PSO technique for real time embedded system and showed that 64% coverage can be achieved after running 10 test cases. Nardo et al. (2015) conducted another case study with an industrial system (name withheld and mentioned as NoiseGen) with 37 real regression faults. The study showed that modification information does not help in increasing fault detection rate. In 2016, a very interesting study conducted by Lu et al. (2016) reflected a new thought process regarding prioritization approach. Very few studies talk about test suite change. The authors concluded that expansion in test suite level did degrade the efficiency of TCP techniques. Table 15 represents a concise snapshot of the above papers.

5.9. Other test case prioritization approaches

As we have detailed our representative publications from each avenue of test case prioritization techniques in the previous subsections, we would like to highlight some other distinct prioritization approaches. Search algorithm based prioritization, ant colony optimization based prioritization, clustering oriented prioritization, multi-objective prioritization, and prioritizations without coverage information are gaining interest.

Data flow information based testing (Rummel et al., 2005), user knowledge based ranking of test cases (Tonella et al., 2006), dynamic runtime behaviour based test case clustering (Yoo et al., 2009) are revealing more defects than control flow based criteria. Test cases are also ranked based on classified events (Belli et al., 2007) and clustering approach (Chen et al., 2018). Siavash Mirarab and Ladan Tahvildari built Bayesian Networks (BN) based prioritization approach (Mirarab and Tahvildari, 2007). The prospect of CIT (Combinatorial Interaction Testing) (Qu et al., 2007; Qu et al., 2008) and construction of call tree based prioritization approach (Smith et al., 2007) look very promising as prioritized test suite takes 82% less time to execute. P R Srivastava applied Non homogeneous Poisson process for optimizing testing time window (Srivastava, 2008).

Li et al. applied meta heuristic and evolutionary algorithms (Li et al., 2007) for TCP. Li et al. conducted simulation studies on Search Algorithms for test case prioritization. Five search algorithms (Total Greedy, Additional Greedy, 2- Optimal Greedy, Hill Climbing and Genetic Algorithms) were studied (Li et al., 2010). Conrad et al. built a new framework called GELATIONS (GEnetic aLgorithm bAsed Test suIte priOritizatioN System) (Conrad et al., 2010). A study to compare the effectiveness of search based and greedy prioritizers were conducted (Williams and Kapfhammer, 2010). ACO (Ant Colony Optimization) technique can also be used for prioritizing test cases (Singh et al., 2010; Agrawal and Kaur, 2018).

Dennis Jeffrey and Neelam Gupta prioritized test cases based on coverage of requirements in the relevant slices of outputs of each

Table 14
Summary of Publications on GUI/Web Application Focused Prioritization Techniques.

Authors and Year	Prioritization Technique	Results	Subject Program
Memon and Xie (2005)	DART (Daily Automated Regression Tester) framework was built.	Other than TerpSpreadSheet, all subjects showed large number of fault.	An office suite TerpOffice
Sprenkle et al. (2005)	User session based testing of web applications was introduced.	Concept analysis based reduction has greater fault detection rate.	Application Bookstore and Course Project Manager
Bryce and Memon (2007)	Test case prioritization by interaction coverage was introduced.	2-way prioritization showed best APFD	Four GUI applications from TerpOffice
Sampath et al. (2008)	Several strategies (frequency of appearance, coverage of parameter) were followed.	Found most of the faults at first 10% of tests executed.	Web based Applications (Book, CPM, MASPLAS)
Bryce et al. (2011)	Unified model for testing web applications and GUI applications together was built.	2-way based prioritization showed promising results for all cases.	4 GUI projects (TerpOffice) and 3 Web Applications (Book, CPM, MASPLAS)

Table 15
Summary of Publications on Prioritization Techniques applied for Real World System.

Authors and Year	Prioritization Technique	Results	Subject Program
Srivastava and Thiagarajan (2002)	The weight of a test is equal to the number of impacted blocks it covers.	TCP works for large systems (1.8 million LOC).	Two versions of a production program
Haymar and Hla (2008)	PSO (Particle Swarm Optimization) technique was proposed.	64% coverage can be achieved after running 10 test cases.	Real time embedded system
Nardo et al. (2015)	The behaviour of real regression faults were studied.	Modification information does not help.	NoiseGen of 59–73 KLOC
Lu et al. (2016)	The study talks about test suite augmentation.	Expansion in test suite level degrade the efficiency.	8 real world Java projects

test case. The study showed that if a test case is traversing a modification, it will not necessarily expose a fault (Jeffrey and Gupta, 2008). ART (Adaptive Random Testing) (Jiang et al., 2009) and xml tags of WSDL (Web Service Description Language) document were also used to order the test suite (Mei et al., 2009).

Prioritizing test cases without coverage information is becoming popular. Mei et al. introduced JUPTA (JUnit Test Case Prioritization Techniques operating in the Absence of coverage information) (Mei et al., 2012). In another study, JUnit test cases were prioritized without using coverage information (Zhang et al., 2009b). Static black box oriented TCP (Thomas et al., 2014), string distance based TCP (Ledru et al., 2012) do not require code or specification detail and test suites are prioritized without the execution of source code or specification models. Aggarwal et al. formulated multiple parameter based prioritization model which actually originates from SRS (Software Requirement Specification) (Aggarwal et al., 2005).

In 2014, Rothermel et al. formulated unified strategy that combined total and additional (not yet covered) techniques together (Hao et al., 2014). Fang et al. designed similarity based test case prioritization technique where the execution profiles of test cases were utilized. Test case diversity yields better performance (Fang et al., 2014). In 2015, Epitropakis et al. clubbed three objectives average percentage of coverage, average percentage of coverage of changed code and average percentage of past fault covered (Epitropakis et al., 2015). Marchetto et al. proposed multi objective prioritization technique (Marchetto et al., 2016). Eghbali et al. developed a lexicographical ordering technique for breaking ties. Vector x is considered to have higher lexicographical rank than Vector y if the first unequal element of x and y has a greater value of x (Eghbali and Tahvildari, 2016). The method proved to be beneficial when more than one test case achieves the same coverage.

We have described the common prioritization mechanisms (coverage/time/cost/history/model etc.) with their methodology, subject programs, results and shortcomings in the previous subsections. However, Test Case Prioritization is dynamically evolving and that is, why we have summarized these other upcoming prioritization viewpoints.

6. Conclusion and future work

As Test Driven Developments (TDD) are generating great profits, testing is not considered as an overhead activity anymore. The test case prioritization techniques actually help in orderly execution of test cases based on some performance or goal function (coverage/time/cost etc.). We have read all the chosen 90 papers in full length before preparing this review. We would like to conclude:

1. Even though APFD (Average Percentage of Faults Detected) is the most used metric for prioritization, Savings Factor, Coverage Effectiveness, – $APFD_C$(APFD per cost), NAPFD (Normalized APFD) etc. are also considered as valuable metrics.
2. The SIR (Software Artifact Infrastructure Repository) library hosts several recurrently used subject programs (tcas, schedule, replace etc.) for TCP studies. UNIX utility programs (grep, flex), JAVA programs (ant, jmeter) and several other case study applications are also selected in repeated endeavour.
3. Coverage aware prioritization methods are dominant while requirement based methods are second best one. Model based TCP and search algorithm based TCP is also finding more attention nowadays.

We would also like to recommend several scope for future work –.

1. Many researchers have addressed the effect of source code modification on test case prioritization, however the study regarding the alteration (augmentation, deletion etc.) in test suite level needs more exploration. Prioritization techniques for ordering multiple test suites can also be designed.
2. Even though TCP techniques without coverage information has shown promising results, it is still not that popular. Acquiring coverage information is tedious and costly in many cases. In this context, techniques without coverage information (Petke et al., 2015; Parejo et al., 2016) should be researched more.
3. Luo et al. (2016) indicated a combination of static techniques (includes source code intervention) and dynamic techniques (uses run time execution data) might be more beneficial for prioritizing test cases. However it is an open end which combination works best. Among the static techniques call graph based method, topic model based method are popular. Total-additional strategy, search based approach, adaptive random testing (ART) are common dynamic techniques. Researchers might delve into this fact that which combination of these above mentioned techniques work best.
4. Researchers have also indicated TCP based on program change information outperforms static or dynamic techniques (Saha et al., 2015). This scenario can open an entire new avenue which will be beneficial for large regression test suite.
5. Further exploration is needed for large software product line (SPL) related TCP. Even though t-wise combinatorial testing helps in reducing the number of test cases, it is mostly limited to small values of t. Henard et al. (2014) proposed a similarity based approach which can be an alternative to t-wise approach. In context of this, realistic approaches for large SPLs should be explored more.
6. Bertolino et al. (2015) opened up a new angle addressing TCP with access control. They indicated similarity criteria is useful for prioritization of access control tests. To the best of our knowledge, we do not see any other study exploring this view. So future work might investigate access control test case prioritization in depth.

We hope this article will be beneficial for both beginners and experienced professionals who have far-reaching interest in TCP domain. We can confide this is the first review with a thorough report of the last 18 years of TCP techniques.

References

Aggarwal, K.K., Singh, Y., Kaur, A., 2005. A multiple parameter test case prioritization model. J. Stat. Manage. Syst. 8, 369–386.
Agrawal, A.P., Kaur, A., 2018. A comprehensive comparison of ant colony and hybrid particle swarm optimization algorithms through test case selection. Data Eng. Intell. Comput., 397–405
Alspaugh, S., Walcott, K.R., Belanich, M., Kapfhammer, G.M., Soffa, M.L., 2007. Efficient time-aware prioritization with knapsack solvers, Proc. – 1st ACM Int. Workshop on Empirical Assessment of Software Engineering Languages and Technologies, WEASELTech 2007, Held with the 22nd IEEE/ACM Int. Conf. Automated Software Eng. ASE 2007, pp. 13–18.
Arafeen, M.J., Do, H., 2013. Test case prioritization using requirements-based clustering, Proceedings – IEEE 6th International Conference on Software Testing, Verification and Validation, ICST 2013, pp. 312–321.
Belli, F., Eminov, M., Gökçe, N., 2007. A fuzzy clustering approach and case study 1 introduction: motivation and related work. Dependable Comput., 95–110
Bertolino, A., Daoudagh, S., El Kateb, D., Henard, C., Le Traon, Y., Lonetti, F., Marchetti, E., Mouelhi, T., Papadakis, M., 2015. Similarity testing for access control. Inf. Softw. Technol. 58, 355–372.
Bryce, R.C., Memon, A.M., 2007. Test suite prioritization by interaction coverage, Workshop on Domain specific approaches to software test automation in conjunction with the 6th ESEC/FSE joint meeting – DOSTA '07, pp. 1–7.
Bryce, R.C., Sampath, S., Memon, A.M., 2011. Developing a single model and test prioritization strategies for event-driven software. IEEE Trans. Software Eng. 37, 48–64.
Catal, C., Mishra, D., 2013. Test case prioritization: a systematic mapping study. Software Qual. J. 21, 445–478.

Chen, J., Zhu, L., Yueh, T., Towey, D., Kuo, F.c., Huang, R., 2018. Test case prioritization for object-oriented software: an adaptive random sequence approach based on clustering. J. Syst. Software 135, 107–125.
Chittimalli, P.K., Harrold, M.J., 2009. Recomputing coverage information to assist regression testing. IEEE Trans. Software Eng. 35, 452–469.
Conrad, A.P., Roos, R.S., Kapfhammer, G.M., 2010. Empirically studying the role of selection operators duringsearch-based test suite prioritization, Proceedings of the 12th annual conference on Genetic and evolutionary computation – GECCO '10, p. 1373.
Do, H., Mirarab, S., 2010. The effects of time constraints on test case prioritization: a series of controlled experiments. IEEE Trans. Software Eng. 36, 593–617.
Do, H., Rothermel, G., 2006. On the use of mutation faults in empirical assessments of test case prioritization techniques. IEEE Trans. Software Eng. 32, 733–752.
Do, H., Rothermel, G., Kinneer, A., 2006. Prioritizing JUnit test cases: an empirical assessment and cost-benefits analysis. Empirical Software Eng. 11, 33–70.
Eghbali, S., Tahvildari, L., 2016. Test case prioritization using lexicographical ordering. IEEE Trans. Software Eng. 42, 1178–1195.
Elbaum, S., Kallakuri, P., Malishevsky, A., Rothermel, G., Kanduri, S., 2003. Understanding the effects of changes on the cost-effectiveness of regression testing techniques. Software Testing Verification Reliab. 13, 65–83.
Elbaum, S., Malishevsky, A., Rothermel, G., 2002. Test case prioritization: a family of empirical studies. IEEE Trans. Software Eng. 28, 159–182.
Elbaum, S., Rothermel, G., Kanduri, S., 2004. Selecting a cost-effective test case prioritization. Software Quality J., 185–210
Elbaum, S., Rothermel, G., Penix, J., 2014. Techniques for improving regression testing in continuous integration development environments, Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering – FSE 2014, pp. 235–245.
Epitropakis, M.G., Yoo, S., Harman, M., Burke, E.K., 2015. Empirical evaluation of pareto efficient multi-objective regression test case prioritisation, Proceedings of the 2015 International Symposium on Software Testing and Analysis – ISSTA 2015, pp. 234–245.
Erdogmus, H., Morisio, M., Torchiano, M., 2005. On the effectiveness of the test-first approach to programming. IEEE Trans. Software Eng. 31, 226–237.
Fang, C., Chen, Z., Wu, K., Zhao, Z., 2014. Similarity-based test case prioritization using ordered sequences of program entities. Software Qual. J. 22, 335–361.
Fang, C.R., Chen, Z.Y., Xu, B.W., 2012. Comparing logic coverage criteria on test case prioritization. Sci. China Inf. Sci. 55, 2826–2840.
Fazlalizadeh, Y., Khalilian, A., Abdollahi Azgomi, M., Parsa, S., 2009. Prioritizing test cases for resource constraint environments using historical test case performance data, Proceedings – 2009 2nd IEEE International Conference on Computer Science and Information Technology, ICCSIT 2009, pp. 190–195.
Feldt, R., Poulding, S., Clark, D., Yoo, S., 2016. Test set diameter: quantifying the diversity of sets of test cases, Proceedings – 2016 IEEE International Conference on Software Testing, Verification and Validation, ICST 2016, pp. 223–233. 1506.03482.
Hao, D., Zhang, L., Mei, H., 2016a. Test-case prioritization: achievements and challenges 10, 769–777.
Hao, D., Zhang, L., Zang, L., Wang, Y., Wu, X., Xie, T., 2016b. To be optimal or not in test-case prioritization. IEEE Trans. Software Eng. 42, 490–504.
Hao, D., Zhang, L., Zhang, L., Rothermel, G., Mei, H., 2014. A unified test case prioritization approach. ACM Trans. Softw. Eng. Methodol. 24 (1), 10–31. article number 10.
Haymar, K., Hla, S., 2008. Applying Particle Swarm Optimization to Prioritizing Test Cases for Embedded Real Time Software Retesting YoungSik Choi 3 . Particle Swarm Optimization Technique in Test Case Prioritization This study applies the particle swarm optimization , 527–532.
Hemmati, H., Arcuri, A., Briand, L., 2013. Achieving scalable model-based testing through test case diversity. ACM Trans. Software Eng. Methodol. 22, 1–42.
Henard, C., Papadakis, M., Harman, M., Jia, Y., Traon, Y.L., 2016. Comparing white-box and black-box test prioritization, 2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE), pp. 523–534.
Henard, C., Papadakis, M., Perrouin, G., Klein, J., Heymans, P., Traon, Y.L., 2014. Bypassing the combinatorial explosion: Using similarity to generate and prioritize t-wise test configurations for software product lines. IEEE Trans. Software Eng. 40, 650–670. arXiv:1211.5451v1.
Huang, Y.C., Peng, K.L., Huang, C.Y., 2012. A history-based cost-cognizant test case prioritization technique in regression testing. J. Syst. Softw. 85, 626–637.
Jeffrey, D., Gupta, N., 2008. Experiments with test case prioritization using relevant slices. J. Syst. Softw. 81, 196–221.
Jiang, B., Zhang, Z., Chan, W.K., Tse, T.H., 2009. Adaptive random test case prioritization, ASE2009 – 24th IEEE/ACM International Conference on Automated Software Engineering, pp. 233–244.
Jones, J.A., Harrold, M.J., 2001. Test-suite reduction and prioritization for modified condition/decision coverage. IEEE Int. Conf. Software Maintenance, ICSM 29, 92–103.
Juristo, N., Moreno, A.M., Vegas, S., 2004. Reviewing 25 Years of Testing Technique Experiments. Empirical Software Engineering 9, 7–44. arXiv:1112.2903v1.
Kapfhammer, G.M., Soffa, M.L., 2007. Using coverage effectiveness to evaluate test suite prioritizations, 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 Conference on Automated Software Engineering (ASE) 2007 – WEASELTech '07, pp. 19–20.
Kim, J.M., Porter, A., 2002. A history-based test prioritization technique for regression testing in resource constrained environments, Proceedings of the 24th international conference on Software engineering – ICSE '02, p. 119.
Kim, S., Baik, J., 2010. An effective fault aware test case prioritization by incorporating a fault localization technique, Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement – ESEM '10, p. 1.
Korel, B., Koutsogiannakis, G., Tahat, L.H., 2007. Model-based test prioritization heuristic methods and their evaluation, Proceedings of the 3rd international workshop on Advances in model-based testing – A-MOST '07, pp. 34–43.
Korel, B., Koutsogiannakis, G., Tahat, L.H., 2008. Application of system models in regression test suite prioritization. IEEE International Conference on Software Maintenance 2008, 247–256.
Korel, B., Tahat, L.H., Harman, M., 2005. Test prioritization using system models. IEEE International Conference on Software Maintenance, ICSM 2005, 559–568.
Krishnamoorthi, R., Arul, Sahaaya, Mary, S.A., 2009. Factor oriented requirement coverage based system test case prioritization of new and regression test cases. Inf. Softw. Technol. 51, 799–808.
Ledru, Y., Petrenko, A., Boroday, S., Mandran, N., 2012. Prioritizing test cases with string distances. Automat. Software Eng. 19, 65–95.
Leon, D., Podgurski, A., 2003. A comparison of coverage-based and distribution-based techniques for filtering and prioritizing test cases, Proceedings – International Symposium on Software Reliability Engineering, ISSRE 2003-Janua, pp. 442–453.
Li, S., Bian, N., Chen, Z., You, D., He, Y., 2010. A simulation study on some search algorithms for regression test case prioritization, 2010 10th International Conference on Quality Software, pp. 72–81.
Li, Z., Harman, M., Hierons, R.M., 2007. Search algorithms for regression test case prioritization. IEEE Trans. Software Eng. 33, 225–237.
Lu, Y., Lou, Y., Cheng, S., Zhang, L., Hao, D., Zhou, Y., Zhang, L., 2016. How does regression test prioritization perform in real-world software evolution?, Proceedings of the 38th International Conference on Software Engineering – ICSE '16, pp 535–546.
Luo, Q., Moran, K., Poshyvanyk, D., 2016. A large-scale empirical comparison of static and dynamic test case prioritization techniques, Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, pp. 559–570.
Malishevsky, A.G., Rothermel, G., Elbaum, S., 2002. Modeling the cost-benefits tradeoffs for regression testing techniques, 2002. Proceedings. International Conference on Software Maintenance, pp. 204–213.
Malishevsky, A.G., Ruthruff, J.R., Rothermel, G., Elbaum, S., 2006. Cost-cognizant Test Case Prioritization. Department of Computer Science and Engineering University of Nebraska Lincoln Techical Report, pp. 1–41.
Marchetto, A., Islam, M.M., Asghar, W., Susi, A., Scanniello, G., 2016. A multi-objective technique to prioritize test cases. IEEE Trans. Software Eng. 42, 918–940.
Mei, H., Hao, D., Zhang, L., Zhang, L., Zhou, J., Rothermel, G., 2012. A static approach to prioritizing JUnit test cases. IEEE Trans. Software Eng. 38, 1258–1275.
Mei, L., Chan, W.K., Tse, T.H., Merkel, R.G., 2009. Tag-based techniques for black-box test case prioritization for service testing. Proceedings – International Conference on Quality Software, 21–30.
Memon, A., Gao, Z., Nguyen, B., Dhanda, S., Nickell, E., Siemborski, R., Micco, J., 2017. Taming google-scale continuous testing, Proceedings – 2017 IEEE/ACM 39th International Conference on Software Engineering: Software Engineering in Practice Track, ICSE-SEIP 2017, pp. 233–242.
Memon, A.M., Xie, Q., 2005. Studying the fault-detection effectiveness of GUI test cases for rapidly evolving software. IEEE Trans. Software Eng. 31, 884–896.
Mirarab, S., Tahvildari, L., 2007. A prioritization approach for software. Test, 276–290.
Mogyorodi, G., 2001. requirements-based testing: an overview caliberrbt/ mercury interactive integration overview. Integrat. VLSI J., 286–295
Nardo, D.D., Alshahwan, N., Briand, L., Labiche, Y., 2015. Coverage-based regression test case selection, minimization and prioritization: a case study on an industrial system. Software Test., Verif. Reliab. 25, 371–396.
Panigrahi, C.R., Mall, R., 2010. Model-based regression test case prioritization. ACM SIGSOFT Software Eng. Notes 35, 1.
Panigrahi, C.R., Mall, R., 2014. A heuristic-based regression test case prioritization approach for object-oriented programs. Innovations Syst. Softw. Eng. 10, 155–163.
Parejo, J.A., Sánchez, A.B., Segura, S., Ruiz-Cortés, A., Lopez-Herrejon, R.E., Egyed, A., 2016. Multi-objective test case prioritization in highly configurable systems: a case study. J. Syst. Softw. 122, 287–310.
Park, H., Ryu, H., Baik, J., 2008. Historical value-based approach for cost-cognizant test case prioritization to improve the effectiveness of regression testing, Proceedings – The 2nd IEEE International Conference on Secure System Integration and Reliability Improvement, SSIRI 2008, pp. 39–46.
Petke, J., Cohen, M.B., Harman, M., Yoo, S., 2015. Practical combinatorial interaction testing: empirical findings on efficiency and early fault detection. IEEE Trans. Software Eng. 41, 901–924.
Qu, X., Cohen, M.B., Rothermel, G., 2008. Configuration-aware regression testing: An Empirical Study of Sampling and Prioritization, Proceedings of the 2008 international symposium on Software testing and analysis – ISSTA '08, p. 75.
Qu, X., Cohen, M.B., Woolf, K.M., 2007. Combinatorial interaction regression testing: a study of test case generation and prioritization, IEEE International Conference on Software Maintenance, ICSM, pp. 255–264.
Rothermel, G., Harrold, M., 1996. Analyzing regression test selection techniques. IEEE Trans. Software Eng. 22, 529–551.
Rothermel, G., Harrold, M., Ostrin, J., Hong, C., 1998. An empirical study of the effects of minimization on the fault detection capabilities of test suites,

Proceedings. International Conference on Software Maintenance (Cat. No. 98CB36272), pp. 34–43.
Rothermel, G., Untcn, R.H., Chu, C., Harrold, M.J., 2001. Prioritizing test cases for regression testing. IEEE Trans. Software Eng. 27, 929–948.
Rummel, M.J., Kapfhammer, G.M., Thall, A., 2005. Towards the prioritization of regression test suites with data flow information, Proceedings of the 2005 ACM symposium on Applied computing – SAC '05, p. 1499.
Saff, D., Ernst, M.D., 2003. Reducing wasted development time via continuous testing, Proceedings – International Symposium on Software Reliability Engineering, ISSRE 2003-Janua, pp. 281–292.
Saha, R.K., Zhang, L., Khurshid, S., Perry, D.E., 2015. An information retrieval approach for regression test prioritization based on program changes. Proceedings – International Conference on Software Engineering 1, 268–279.
Salem, Y.I., Hassan, R., 2011. Requirement-based test case generation and prioritization, ICENCO'2010 – 2010 International Computer Engineering Conference: Expanding Information Society Frontiers, pp. 152–157.
Sampath, S., Bryce, R.C., Viswanath, G., Kandimalla, V., Koru, A.G., 2008. Prioritizing user-session-based test cases for web applications testing, Proceedings of the 1st International Conference on Software Testing, Verification and Validation, ICST, pp. 141–150.
Singh, Y., Kaur, A., Suri, B., 2010. Test case prioritization using ant colony optimization. ACM SIGSOFT Software Eng. Notes 35, 1.
Smith, A., Geiger, J., Kapfhammer, G.M., Soffa, M.L., 2007. Test suite reduction and prioritization with call trees, Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering – ASE '07, p. 539.
Smith, A.M., Kapfhammer, G.M., 2009. An empirical study of incorporating cost into test suite reduction and prioritization, Proceedings of the 2009 ACM symposium on Applied Computing – SAC '09 1, p. 461.
Sprenkle, S., Gibson, E., Pollock, L., Souter, A., 2005. An empirical comparison of test suite reduction techniques for user-session-based testing of Web applications, 21st IEEE International Conference on Software Maintenance (ICSM'05), pp. 587–596.
Srikanth, H., Banerjee, S., Williams, L., Osborne, J., 2013. Towards the prioritization of system test cases. Software Test, Verif. Reliab. 24, 320–337.
Srikanth, H., Cohen, M.B., Qu, X., 2009. Reducing field failures in system configurable software: Cost-based prioritization, Proceedings – International Symposium on Software Reliability Engineering, ISSRE, pp. 61–70.
Srikanth, H., Williams, L., 2005. On the economics of requirements-based test case prioritization. ACM SIGSOFT Software Eng. Notes 30, 1.
Srikanth, H., Williams, L., Osborne, J., 2005. System test case prioritization of new and regression test cases. Int. Symp. Empirical Software Eng. ISESE 2005 (00), 64–73.
Srivastava, A., Thiagarajan, J., 2002. Effectively prioritizing tests in development environment. ACM SIGSOFT Software Eng. Notes 27, 97.
Srivastava, P.R., 2008. Model for optimizing software testing period using non homogenous Poisson process based on cumulative test case prioritization, IEEE Region 10 Annual International Conference, Proceedings/TENCON, pp. 1–6.
Srivastva, P.R., Kumar, K., Raghurama, G., 2008. Test case prioritization based on requirements and risk factors. ACM SIGSOFT Software Eng. Notes 33, 1.
Stallbaum, H., Metzger, A., Pohl, K., 2008. An automated technique for risk-based test case generation and prioritization. Automat. Software Test 67.
Thomas, S.W., Hemmati, H., Hassan, A.E., Blostein, D., 2014. Static test case prioritization using topic models. Empirical Software Eng. 19.
Tonella, P., Avesani, P., Susi, A., 2006. Using the case-based ranking methodology for test case prioritization, IEEE International Conference on Software Maintenance, ICSM, pp. 123–132.
Uusitalo, E.J., Komssi, M., Kauppinen, M., Davis, A.M., 2008. Linking requirements and testing in practice, Proceedings of the 16th IEEE International Requirements Engineering Conference, RE'08, pp. 265–270.
Walcott, K.R., Soffa, M.L., Kapfhammer, G.M., Roos, R.S., 2006. Time aware test suite prioritization, Proceedings of the 2006 international symposium on Software testing and analysis – ISSTA'06, p. 1.
Williams, Z.D., Kapfhammer, G.M., 2010. Using synthetic test suites to empirically compare search-based and greedy prioritizers, Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation (GECCO '10), pp. 2119–2120.
Wong, W.E., Morgan, J.R., London, S., Mathur, A.P., 1998. Effect of test set minimization on fault detection effectiveness. Software – Practice and Experience 28, 347–369.
Yoo, S., Harman, M., 2007. Regression testing minimisation, selection and prioritisation: a survey. Test. Verif. Reliab 00, 1–7.
Yoo, S., Harman, M., Tonella, P., Susi, A., 2009. Clustering test cases to achieve effective and scalable prioritisation incorporating expert knowledge. Proc. ISSTA, 201–212.
Yoon, M., 2012. A test case prioritization through correlation of requirement and risk. J. Software Eng. Appl. 05, 823–836.
You, D., Chen, Z., Xu, B., Luo, B., Zhang, C., 2011. An empirical study on the effectiveness of time-aware test case prioritization techniques, Proceedings of the 2011 ACM Symposium on Applied Computing, pp. 1451–1456.
Zhang, L., Hou, S.S., Guo, C., Xie, T., Mei, H., 2009a. Time-aware test-case prioritization using integer linear programming, Proceedings of the eighteenth international symposium on Software testing and analysis – ISSTA '09, pp. 401–419.
Zhang, L., Zhou, J., Hao, D., Zhang, L., Mei, H., 2009b. Jtop: managing Junit test cases in absence of coverage information, ASE2009 – 24th IEEE/ACM International Conference on Automated Software Engineering, pp. 677–679. 7.
Zhang, X., Nie, C., Xu, B., Qu, B., 2007. Test case prioritization based on varying testing requirement priorities and test case costs. Proceedings – International Conference on Quality Software, 15–24.