2023 IEEE/ACIS 21st International Conference on Software Engineering Research, Management and Applications (SERA) | 979-8-3503-4588-9/23/$31.00 ©2023 IEEE | DOI: 10.1109/SERA57763.2023.10197719

A Systematic Literature Review on Test Case Prioritization and Regression Test Selection

1st Zhengxinchao Xiao
School of computer and Information Engineering
Xiamen University of Technolog
Xiamen, China
17268232312@163.com

2nd Lei Xiao
School of computer and Information Engineering
Xiamen University of Technology
Xiamen, China
lxiao@s.xmut.edu.cn

Abstract*—Regression testing is a crucial component of software testing and a crucial tool for ensuring the quality of software. An appropriate optimization method is essential for maximizing productivity and reducing expenses in regression testing. Test case prioritization (TCP) and regression test selection (RTS) are two popular methods in regression testing. This paper provides a qualitative analysis of 18 TCP and 17 RTS publications from the last five years. This paper presents four main issues. The first covers the most popular TCP techniques, the second covers the most popular RTS methods, the third covers the most popular metrics for measuring TCP and RTS, and the fourth covers data sources. Based on this study, we draw the following conclusions: (1) Defect prediction and machine learning-based TCP methods, machine learning, multi-objective, and model-based RTS methods will receive additional attention in future. (2) Defects4J is the most commonly used data set in TCP in the past five years. SIR and GitHub are the most commonly used datasets in RTS. (3) The most widely used measurement methods in TCP and RTS are APFD and cost, respectively. In future, researchers will use these two indicators to conduct a more comprehensive evaluation together with cost, fault detection capability, and test coverage.

Keywords*—Regression testing; Test case prioritization; Test case selection; Systematic literature review

I. Introduction

Regression testing is frequently used to guarantee the software's quality during the software development process. Regression testing accounted for more than half of the tests. Since regression testing is frequent and occupies a large proportion of total testing, the cost becomes higher and higher. Three main regression testing techniques are used to improve our efficiency. The three major technologies are test case prioritization (TCP), regression test selection (RTS), and test case minimization (TCM). Test case minimization is also known as test case reduction. Test case prioritization is the prioritization of test cases to improve the efficiency of regression testing through prioritization. TCP was first proposed by Wong et al. in 1997 [1]. Regression test selection selects a subset of test cases and performs only those parts that perform the change. Rothermel and Harrold defined regression test selection in 1996 [2]. Test case minimization is performed by identifying redundant test cases and then removing the identified redundant test cases. This technique is designed to reduce the size of the test suite. Harrold et al. proposed the first formal definition of test case minimization in 1993 [3]. This paper focuses on TCP and RTS.

Natural Science Foundation of Fujian Province of China (No.2022J011238).

The main contributions of this paper are: (1) This paper provides a literature review on test case prioritization and regression test selection, and reviews 18 TCP and 17 RTS papers from the last five years. (2) This review paper provides a detailed analysis of the methodology, evaluation metrics, and datasets for TCP and RTS. (3) This review paper is designed with five research questions that we hope will be useful for beginners and experts in the field of TCP and RTS. The following questions are discussed: RQ1: What have been the most common approaches to TCP over the last five years? RQ2: What have been the most common approaches to RTS over the last five years? RQ3: What are the common metrics used to evaluate TCP and RTS in the last five years? Which indicator is most commonly used? RQ4: What are the data sources for TCP and RTS papers in the last five years?

The rest of the paper is organized as follows: Section 2 describes our search strategy. Section 3 introduces the research contents. Section 4 describes the findings and discussions. Section 5 describes the conclusions and prospects.

II. Search Strategy

First, determine the search strings: (test case prioritization) or (test suite prioritization) or (regression test selection) or (test case selection) or (test suite selection). Next, use Google Scholar to search for literature. Google Scholar includes several authoritative databases, including the ACM Digital Library, Science Direct, IEEE Xplore, and Web of Science. The time horizon is from 2018 to 2022, and the language of the paper is English. After manual screening and grading, 35 literature works were selected. The number of selected documents from 2018 to 2022 is 8, 6, 10, 8, and 3. The number of papers in the past five years is shown in Figure 1. There were 21 journal papers and 14 conference papers among 35 articles, accounting for 60% and 40% of the total, respectively. TCP literature published at the conference is [4] and [8]. TCP literature published in journals includes [5-7] and [9-21]. RTS literature published at the meeting includes [23-26], [31-36], [38], and [40]. RTS literature published in journals include [22], [27-30], [37], and [39]. Tables 1 and 2, respectively, present details of the TCP literature and the RTS literature. The specific gravity of the literature is shown in Figure 2.

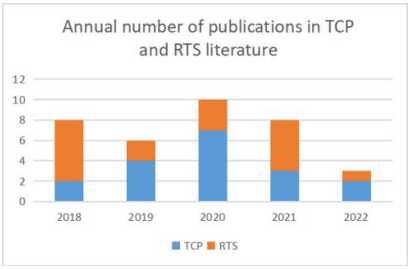

Fig.1 Annual number of publications in TCP and RTS literature

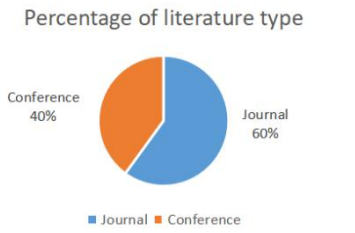

Fig.2 Percentage of literature type

TABLE I. TCP LITERATURE DETAILS TABLE

Reference	Year	TCP Approach	Evaluation Index	Dataset
[4]	2018	machine learning based	APFD	other programs
[5]	2019	machine learning based	APFD and cost	credit card approval system and registration verifier application
[6]	2021	machine learning based	APFD and NRPA	other programs
[7]	2022	machine learning based	fault detection ability	other programs
[8]	2019	defects prediction based	APFD	Defects4J
[9]	2022	defects prediction based	fault detection ability	Defects4J
[10]	2020	defects prediction based	APFD	SIR
[11]	2020	defects prediction based	APFD	Defects4J
[12]	2019	coverage based	APFD,APSC ,APBC,APF C,cost,fault detection ability, and test coverage	SIR
[13]	2020	coverage based	APFD and APFDc	other programs
[14]	2021	coverage based	APTC	other programs
[15]	2020	model based	APFD	other programs
[16]	2020	model based	APFD	other programs
[17]	2020	risk based	APFD and other	credit card approval system and registration verifier application
[18]	2021	multi-objective based	APFD, cost, fault detection ability and test coverage	other programs
[19]	2019	mutation based	APFD and APMK	Defects4J
[20]	2020	relation based	APFD	other programs
[21]	2018	object-oriented software	APFD and other	other programs

TABLE II. RTS LITERATURE DETAILS TABLE

Reference	Year	RTS Approach	Evaluation index	Dataset
[22]	2018	multi-objective based	cost	other programs
[23]	2021	multi-objective based	cost and test coverage	other programs
[24]	2021	multi-objective based	cost and test coverage	SIR
[25]	2020	machine learning based	test coverage	other programs
[26]	2021	machine learning based	test coverage	other programs
[27]	2019	model based	test coverage and fault detection ability	other programs
[28]	2022	model based	cost and fault detection ability	other programs
[31]	2020	java web	cost	other programs
[32]	2019	web service	cost	other programs
[33]	2018	hybrid	cost	GitHub
[34]	2018	mutation based	fault detection ability	GitHub
[35]	2021	requiremen t based	test coverage and fault detection ability	other programs
[36]	2021	genetic Improveme nt	cost	other programs
[37]	2018	fault localizatio n based	cost	SIR
[38]	2018	algorithm based	cost and test coverage	SIR
[39]	2020	algorithm based	fault detection ability	SIR
[40]	2018	continuous integration	cost	GitHub

III. RESEARCH QUESTIONS

Four research questions have been identified; they are as follows:

RQ1: What have been the most common approaches to TCP over the last five years?

RQ2: What have been the most common approaches to RTS over the last five years?

RQ3: What are the common metrics used to evaluate TCP and RTS in the last five years? Which indicator is most commonly used?

RQ4: What are the data sources for TCP and RTS papers in the last five years?

IV. Result and Discussion

A. Common approaches for TCP（RQ1）

Figure 3 displays common TCP approaches throughout the previous five years. The two most popular approaches are defects prediction-based TCP and machine learning (ML)-based TCP. At the same time, we anticipate that both approaches will gain popularity over time. This will develop into a fresh area of study. Applying machine learning techniques to regression testing has become popular in the last five years. There are four main ML techniques in TCP: supervised learning, unsupervised learning, reinforcement learning, and NLP-based learning. First, people label test cases according to different attributes and obtain a data set that can be used for training. Machine learning algorithms are then trained on the dataset to rank the test cases according to their priority. Test cases with higher priority are executed first. Ranked Support Vector Machines (SVM Rank), k-nearest-neighbor (KNN), Logistic Regression (Log Reg), and neural networks are four common algorithms in machine learning. Defect prediction (also called fault prediction) is used to estimate the possibility of file or function failure in a software system in future. Coverage-based TCP has been a popular TCP research method. Many other TCP approaches have received attention from other scholars in the last five years. These TCP methods are mutation, relation, risk, multi-objective, object-oriented software, and model-based TCP methods.

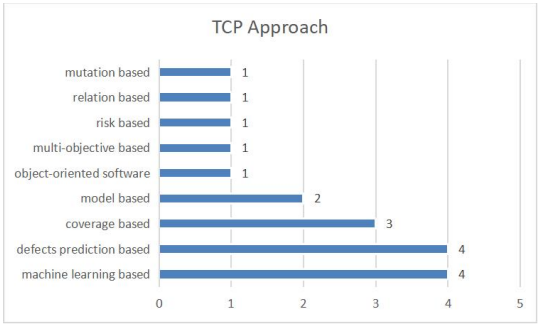

Fig.3 Common approaches to TCP over the last five years

1) Machine learning-based TCP

Lachmann [4] applied machine learning (ML) methods to black-box testing and proposed TCP technology based on supervised ML. These four well-known techniques were used to establish the priority of the test cases and the order in which they should be executed. Jahan et al. [5] proposed a ML-based supervised TCP approach. They used an artificial neural network model, trained and then classified, to predict the priority of test cases. They were limited by the modest size of the projects they used and the fact that the bugs were artificially implanted. Bagherzadeh et al. [6] applied reinforcement learning (RL) methods to Continuous Integration (CI) environments for the dynamic nature of test priorities in CI environments. They finished building the RL environment before training the agent with the most sophisticated RL algorithm. The RL agent was eventually integrated with the CI environment to score the test cases during each CI cycle. Waqar et al. [7] proposed a TCP method based on RL, called Q-learning. They use a RL model to train agents to interact with the environment. They used the RL model to predict the list of the highest future reward sequences based on the log dataset. They put bugs into the app. This is called fault seeding. They used fault seeding to validate the RL model.

2) Defect prediction-based TCP

File-level defect prediction is one of the most popular directions in defect prediction. Paterson et al. [8] introduced a method to determine TCP using defect prediction, called G-clef. To rank test cases, G-clef needed to find a secondary target first. Then they needed to use a constraint solver to cover all lines of code. Currently, there are no studies comparing the impact of predictors on the fault detection rate of test cases. Mahdieh et al. [9] introduced a neural network-based defect prediction TCP method. At present, numerous coverage-based TCP methods only use code coverage. Their methods differ in that they include fault tendency information. They designed a feature extractor to extract source code and historical features related to bug prediction. They introduced a defect prediction method to learn a neural network model with input and hidden layers. ShAo et al. [10] found that no software defect prediction model can predict the defect tendency of code statements. To address this issue, they proposed TCP-SCSDP, a TCP method based on code statement level defect prediction. They first extracted statement metrics from the source code, then preprocessed the data, and finally input a defect predictor. Together with the predictor predictions, they can obtain the defect propensity probability of valid code statements. Mahdieh et al. [11] introduced a method to incorporate fault propensity estimates for TCP by diversifying fault propensity and test cases. They added fault propensity to the cluster-based formula. They first used a clustering method to group similar test cases, and then the defect prediction method was applied to the error history and the estimation of all code units in the source code to estimate the fault tendency of code units.

3) Coverage-based TCP

Lu et al. [12] proposed a coverage-based ant colony system framework to test case ordering. They proposed a coverage-based ant colony system (CB-ACS). In an ant colony system, TCP problems are mapped to construct graph representations. They focus on the actual coverage information about statements that have not yet been covered. They used the ACB-Heuristic Function and this information in their subsequent search process. Numerous traditionally effective prioritization policies, such as greedy policies, tend to consider code units in isolation. Huang et al. [13] additionally considered a combination of code units to avoid the loss of coverage information. They applied code composition coverage to the regression test case prioritization (RTCP) and

proposed code combination coverage-based prioritization (CCCP). They referred to code units to represent code element statements, branches, or methods. Genetic algorithms (GA) are particularly popular among coverage-based techniques. The advantage of GA is that it can automatically generate test cases. Alrawashdeh et al. [14] introduced an automated TCP approach through the use of modified genetic algorithms.

4) Model-based TCP

Pospisil et al. [15] proposed a model-based enhanced adaptive stochastic TCP approach called Mc-ARP. They used semi-random heuristics to sort test cases. Model-based testing can automatically generate test cases from the model, and the generated test cases can be automatically executed by automated tools. Mohd-Shafie et al. [16] proposed an enhanced model-based TCP method (MB-TCP). The method uses a selective and evenly distributed counting method combined with a strict ranking criterion (SESOC). Regarding the system model, they chose the finite state machine (FSM) to prioritize.

5) Risk-based TCP

Manual calculation of the value of risk is widely valued, and experts who rely on historical data to determine the value of risk are frequently not accurate. Automated risk assessment programs have shown their superiority in identifying high-risk failures. Jahan et al. [17] proposed a semi-automatic risk-based test case prioritization method based on software modification information and method (function) call relationships. They used the changed requirements (CR), method complexity (MC), and method size (MS) risk factors to calculate the risk value of the TCP method.

6) Multi-objective-based TCP

Samad et al. [18] proposed an improved multi-objective particle swarm optimization (MOPSO) to prioritize by considering multiple objectives. Their algorithm considers three objective parameters: code coverage, execution time, and fault detection capability.

7) Mutation-based TCP

There is a growing body of research showing that distinguishing between mutants can lead to faster fault detection. Shin et al. [19] introduced a novel test case prioritization technique that combines variance-based and diversity-aware approaches. The diversity-aware technical approach distinguishes the behavior of each mutant from the behavior of all other mutants, including the original program. To be able to quickly distinguish between all mutants, diversity-aware mutations give higher priority to test cases.

8) Relation-based TCP

Chi et al. [20] proposed a relation-based TCP technique called Additional Greedy Call Sequence (AGC). They guided TCP processes through sequences of dynamic function calls and relationship-based information. Their approach has three steps: performing monitoring, aggregating graph construction, sampling, and prioritization.

9) Object-oriented-based TCP

At present, a growing number of regression testing methods are used in object-oriented software (OOS). However, the OOS test method produces a large number of test cases. Clustering techniques are widely used in regression testing. Test cases with similar attributes are grouped into the same cluster by clustering. Chen et al. [21] proposed an adaptive random sequence (ARS) method for OOS TCP. Their ARS method clusters test cases according to the number of objects and methods. Test cases in the same cluster have similar fault detection capabilities.

B. Common approaches for RTS (RQ2)

Common approaches to RTS over the last five years are shown in Figure 4. In the last five years, the most popular RTS approaches have been based on multiple-objective, model, and machine learning. These three techniques are likely to garner more interest and develop into fresh study hotspots in the future. In the study of regression test selection, more and more consideration has been given to the effect of multiple objectives on the test. There is a tendency to optimize multiple objectives. Garousi et al. [22] considered effective requirements, costs, and benefits. Chen et al. [23] focused on code coverage, execution time, and fault detection capability. Olsthoorn and Panichella [24] targeted statement coverage, branch coverage, and execution costs. Machine learning approaches are also popular in RTS. The RTS method of Sutar et al. [25] applied NLP in ML. Martins et al. [26] used supervised learning in RTS. Model-based RTS has also received considerable attention. Al-Refai et al. [27] used a UML model consisting of class diagrams and activity diagrams. Cazzola et al. [28] used UML sequence diagrams and activity diagrams. Alternative RTS techniques include fault localization, genetic improvement, requirement, mutation, hybrid, web service, java web, and other techniques.

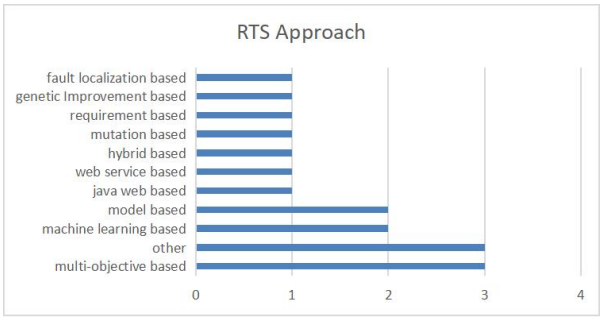

Fig.4 Common approaches to RTS over the last five years

1) Multi-objective-based RTS

Garousi et al. developed a GA-based multi-objective regression test optimization approach (MORTO). Their GA approach considers three objectives: full coverage of affected (changed) requirements, costs, and benefits. Chen et al. proposed two algorithms to select test case sets with significant fault detection capability within a constrained time. Their proposed multi-objective function takes fault detection ability and test execution time as inputs. They focus on the effect of changes in the program and its execution environment. The consideration of change effects from program changes and execution context changes, as well as the focus on change impact coverage, are unique to their technique. Olsthoorn and Panichella proposed a variant of the new NSGA protocol, L2-NSGA, for multi-objective Test Case Selection (TCS). They used multi-objective evolutionary algorithms (MOEAs) to

solve RTS problems. They focus on three goals: statement coverage (maximization), branch coverage (maximization), and execution cost (minimization). They used link learning methods from machine learning for multi-objective test case selection. They used an unsupervised clustering algorithm to infer promising patterns in the solution, a subset of the test suite. These patterns are then used in the next iteration of L2-NSGA to create a solution that preserves these inferred patterns.

2) Machine Learning-based RTS

Sutar et al. used natural language processing (NLP) techniques in RTS. They used natural language processing (NLP) to extract the meaning of each test case and defect in a domain-specific context. First, they input test suite data and defect data from Team Foundation Server (TFS). Second, they perform text analysis. The text information they collect comes from the first step and is preprocessed by the text data. They merge relevant columns of data extracted from the test suite and TFS and perform tokenization and similarity measurement. The final output is a column of selected test cases. Martins et al. used supervised learning in ML for RTS in CI. This approach makes use of features associated with changes to the test suite and code files. CI settings frequently have erratic tests and innocent submissions, which they resolve with classifiers. Their method for training ML classifiers comes from the method selected by the test suite. They started by defining datasets of information (during the test phase) about code changes made by developers and running tests on that code. Next, they found classifiers to train the dataset and provided some features of ML classifiers, such as test failure rate.

3) Model-based RTS

Compared to code-based RTS, model-based RTS scales better, regulates dependencies at a higher level of abstraction, and is more efficient than the source-code level. Al-Refai et al. extended their 2016 proposal for a regression test selection method based on MaRT's model [29]. MaRTS uses a UML model consisting of class diagrams and activity diagrams. In their original approach, they did not note the effect of changes in the inheritance hierarchy in the class graph. Their current approach extends MaRTS to support the effects of changes made to the inheritance hierarchy. Most UML model-based RTS methods use behavior diagrams or a combination of structure diagrams and behavior diagrams. Cazzola et al. modified the previously proposed RTS method [30] using UML sequence diagrams and activity diagrams. In the new study, they proposed the RTS method FLiRTS 2, which only focuses on UML class diagrams. FLiRTS 2 classifies test cases by fuzzy logic. FLiRTS 2 divides test cases into two classes: retestable test cases that must be re-executed and reusable test cases that can be discarded.

4) Java web-based RTS

Long et al. [31] proposed a dynamic regression test selection tool, WebRTS, for Java Web applications. They designed a WebRTS that can be easily integrated with web testing frameworks. However, the WebRTS tool has one drawback. The JavaScript file that runs in the default browser for WebRTS is a resource file. The JavaScript file change and all tests that use it will be selected.

5) Web service-based RTS

Zhong et al. [32] proposed TESTSAGE, the first RTS technique based on large-scale web service testing. TESTSAGE has three steps: analyze, run execution of selected tests, and gather information. TESTSAGE is separate from the test and the system under test. TESTSAGE uses XRAY to collect test dependencies.

6) Hybrid-based RTS

The RTS approach has the advantage of different granularity, such as basic blocks, methods, and file levels. Researchers have found that fine-grained RTS techniques are more accurate and coarse-grained techniques have less overhead. File-level RTS has a shorter end-to-end time. Zhang [33] proposed a more cost-effective hybrid RTS approach called HyRTS. This approach combines the advantages of both the method and the file granularity. They performed two levels of RTS analysis using HyRTS. File-level analysis obtains file dependencies and computes detailed method-level changes when corresponding files are modified. The method-level analysis captures the method dependencies and simply calculates file-level changes when files are deleted and added.

7) Mutation-based RTS

Mutation testing is a fault-based testing method that has been used to evaluate the quality of test suites. But the downside is that it takes a long time. The RTS accelerated regression mutation test was first performed by Chen et al. [34]. They used the mutation testing tool PIT to perform regression mutation testing. They used an algorithm to aggregate the results of their mutation tests. They experimentally found that file-level static and dynamic RTS techniques outperformed each other in terms of accurate and efficient regression mutation testing.

8) Requirement-based RTS

With the popularity of DevOps, behavior-driven development (BDD), which is closely combined with agile practices, has attracted more and more attention. Xu et al. [35] introduced a requirement-based RTS technique in BDD. They used a framework to evaluate and demonstrate the effectiveness of the approach. This framework consists of inclusiveness, precision, efficiency, and generality.

9) Genetic improvement-based RTS

Guizzo et al. [36] combined the artificial intelligence technique of genetic improvement (GI) with the RTS method. They compared this method with RTS based on dynamic analysis and RTS based on static analysis. They investigated GI in combination with state-of-the-art RTS strategies and demonstrated the feasibility of the modified approach. They are the first to investigate the impact of any test selection strategy in a non-functional automated software improvement environment using GI and dynamic static RTS techniques.

10) Fault localization-based RTS

Wang et al. [37] introduced a multi-criteria RTS method based on fault localization. They used the fault location method to find the faulty statement from the correct statement. They used a greedy algorithm to rank all passed test cases according to priority criteria. The test cases are then selected according to the selection criteria.

11) Other RTS Approaches

Saber et al. [38] proposed a different hybrid algorithm, GREAP (for Greedy Evolutionary Algorithm Path-relinking). GREAP is based on three optimization algorithms. Greedy algorithms are used to find excellent solutions as quickly as possible. Genetic algorithms are used to expand the search space. The function of the local search algorithm is to refine the solution. In the end, they chose the reconnecting path. They also pushed the Pareto frontier. Agrawal et al. [39] introduced an RTS method based on the Hybrid Whale Optimization Algorithm (HWOA). The power of HWOA is to find the maximum fault coverage with the fewest test cases. They investigated the effect of the number of test suites on the fault detection efficiency of the RTS method. They also found the effect of the number of test cases on run time and cost. In a CI setting, test cases are frequently committed. Developers submit shorter, more frequent pieces of work in a CI environment. Therefore, a cost-effective RTS has been developed. Wang and Yu [40] studied the use of RTS in CI environments for different open-source projects. They studied the frequency of submissions, how project files vary between submissions, the extent to which testing times exceed the submission interval, and the impact of dependency analysis at different levels of granularity.

C. *Common metrics used to evaluate TCP and RTS in the last five years（RQ3）*

The evaluation metrics in the TCP literature are shown in Figure 5. The evaluation indicators of TCP include APFD, fault detection efficiency, time, code coverage, test suite reduction, APFDc, fault detection ability, number of defects detected, APFC, APBC, APSC, APTC, APMK, NRPA, and others. The APFD has been used 15 times in 18 TCP literature works, making it the most frequently used evaluation metric. The second most frequently used metric in TCP is fault detection efficiency. Seven studies [4], [8], [10], [11], [15], [16], and [20] only used APFD as the evaluation index. In some alternative literature, researchers have used APFD in conjunction with additional evaluation metrics. For example, when researchers used APFD in [5], [6], [13], and [19], they added cost, the normalized range percentile average (NRPA), APFDc, APMK, and additional evaluation indicators. Researchers used APFD, APSC, APBC, APFC, cost, fault detection ability, and test coverage in [12]. Researchers used APFD, cost, fault detection ability, and test coverage in [18]. In [12] and [18], researchers comprehensively evaluated TCP in terms of cost, fault detection capability, and test coverage. Costs include the test suite reduction rate and execution time. The fault detection capability consists of the fault detection rate and the number of detected faults. Test coverage includes code coverage. Therefore, APFD is the most widely used metric in TCP. To circumvent the limitations of using a single evaluation metric, a growing number of researchers have combined APFD with different metrics for comprehensive evaluation. In future, it will be a trend for researchers to use APFD in combination with cost, fault detection capability, and test coverage metrics.

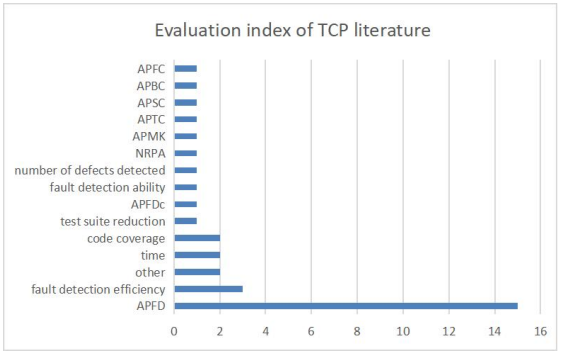

Fig.5 Evaluation index of TCP literature

The evaluation indicators in the RTS literature are shown in Figure 6. The evaluation metrics are as follows: cost, test coverage, and fault detection ability. The most commonly used metric in RTS is cost. Seven studies [22], [31-33], [36], [37], and [40] used cost alone as an evaluation indicator. In [23], [24], and [38], researchers used cost and test coverage. In [28], researchers used cost in combination with fault detection ability. [25] and [26] separately used test coverage as an evaluation indicator. [34] and [39] only used fault detection abilities. [27] and [35] used two evaluation indicators: test coverage and fault detection ability. Costs include selected tests, test time, expenses, and reduced test case sets. Test coverage includes code coverage, statement coverage, branch coverage, line coverage, and MC/DC coverage. Fault detection capability includes recall, micro recall, effectiveness, consistency, fault detection effectiveness, inclusiveness, precision, and generality. Overall, cost is the most widely used metric in RTS. In future, more and more people will choose cost, test coverage, and fault detection ability as evaluation indicators for a more comprehensive evaluation.

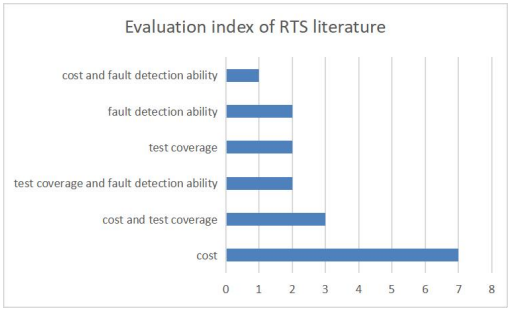

Fig.6 Evaluation index of RTS literature

From Table 1, we can see that among the TCP topics, nine studies (accounting for 50% of the total literature) only focus on a single evaluation indicator, four studies use two evaluation indicators, and five studies consider more than three indicators for evaluation. From the research on RTS in Table 2, we can see that eleven studies (accounting for about 65% of the total literature) only focus on a single evaluation index, and six studies use two evaluation indexes for evaluation. If only one evaluation metric is used, the evaluation results are easily one-sided. Increasingly, scholars are using two or more evaluation metrics.

D. *Data sources for TCP and RTS papers (RQ4)*

The data sources in the TCP literature are shown in Figure 7. Among the different programs, the most used dataset in TCP is Defects4J. The second is SIR, the credit card approval

system, and the registration verifier application. [8], [9], [11], and [19] used Defects4J's real failures to enhance the validity of the studies. Defects4J is a collection of 395 reproducible real-world software errors from six large open-source Java programs. Defects4J was used in three of the four TCP articles based on defect prediction: [8-9], [11], and mutation-based literature [19]. Literature [8-9] and [11] chose Defects4J for several reasons. First, Defects4J has a large test suite. Secondly, Defects4J contains various test cases and recorded bug histories. Third, the source of the project, error location, and failed test cases must be identifiable. Literature [8] requires obtaining previous versions of source code. [10] and [12] used Software-Artifact Infrastructure Repository (SIR) datasets. The coverage-based study [12] selected the Siemens and Space programs in SIR, which have been widely used as benchmarks for coverage based on TCP problems. [5] and [17] used a credit card approval system and registration verifier application. Machine learning-based research [5] select two medium-sized software applications with multiple versions. These two procedures are used to evaluate test predictions based on artificial neural networks. There are additional programs used by various researchers. For example, [4] used the automotive domain system. [7] used an Android program. [13] used Java programs and Unix programs for empirical research. [14] used the vending machine system (VMS) and ATM system. [15] used a dataset of six industrial project data sets. [16] used a web application. [18] used TreeDataStructure, JodaTime, and Triangle datasets. [6] and [20] used Java programs. [21] used programs written in C++ or C# languages.

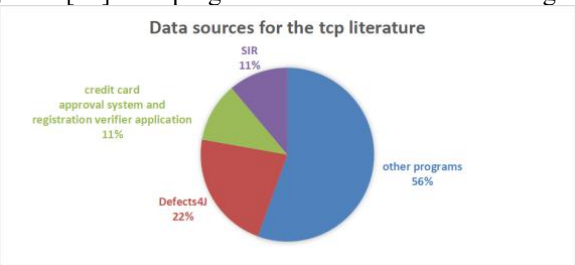

Fig.7 Data sources for the TCP literature

Figure 8 displays the RTS literature's data sources. Among several programs, SIR and GitHub have the highest usage rates for datasets in RTS. Studies [24] and [37-39] used SIR data sets. SIR provides several subsequent versions of the program and their test suites. The SIR also includes incorrect versions of these programs with seed (human) errors. Research [24] considered nontrivial seed failures, which can only be detected by a few tests. Based on the study of fault location [37], Siemens and space programs in SIR were selected. These two procedures are widely used in the study of fault locations. Errors in the Siemens project were manually displayed, while those in the space project occurred naturally. [33-34] and [40] used GitHub datasets. Mutation-based research [34] uses 20 real-world Java projects on GitHub that have been widely used for mutation testing research. Another 59% of studies chose alternative data sets. For example, [22] used the controller software system in the field of national defense. [28] used 13 open-source software systems. The study [23] used the educational software idea thread mapper and a web application implemented by Java. [25] used the Test Management System (TMS) tool. [26] used data from service studio in the OutSystems IDE. [27] used Java programs. [31] made use of Java Web applications as well as web applications. [32] used a web service project. [35] used subsystems of real industry projects. [36] used seven large real-world software from Apache Commons Project 4.

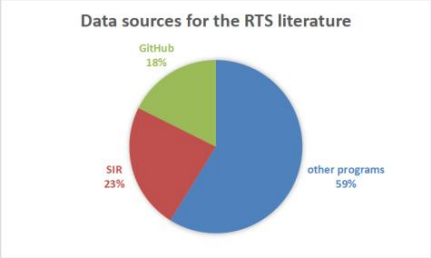

Fig.8 Data sources for the RTS literature

V. Conclusion and Prospect

This paper provides a systematic review of 18 TCP and 17 RTS papers from the past 5 years. We analyze the current status and future trends of TCP and RTS methods, data sources, and evaluation indicators. The study's conclusions are as follows: (1) Current status and future trends of TCP and RTS methods. In the past five years, defect prediction and machine learning methods have been the most used methods in TCP. As a result, we predict that these two methods will attract more and more attention in future. This will become a new research trend. In the past five years, the most used methods in RTS have been based on multi-objective, machine learning, and model. These methods of RTS will become the new research trend. Scholars can increase investment and research in this direction in future. (2) Data sources. Among other programs, the most used dataset in TCP is Defects4J. The second is SIR, the credit card approval system, and the registration verifier application. In future, we predict that more people will choose Defects4J for experimentation. Among other programs, the most used datasets in RTS are SIR and GitHub. Future usage of SIR and GitHub is something we anticipate will grow. (3) Evaluation indicators. The most used measurement techniques in TCP and RTS, respectively, are APFD and Cost. Future studies will combine cost, fault detection capability, and test coverage with these two indicators to complete a more thorough evaluation.

Acknowledgment

This work is partially supported by the Natural Science Foundation of Fujian Province of China (No. 2022J011238).

References

[1] Wong, J. R. Horgan, S. London, and H. Agrawal, “A study of effective regression testing in practice,” in PROCEEDINGS The Eighth International Symposium On Software Reliability Engi- neering. IEEE, 1997, pp. 264–274

[2] G. Rothermel and M. J. Harrold, “Analyzing regression test selection techniques,” IEEE Transac- tions on software engineering, vol. 22, no. 8, pp. 529–551, 1996.

[3] M. J. Harrold, R. Gupta, and M. L. Soffa, “A methodology for controlling the size of a test suite,” ACM Transactions on Software

Engineering and Methodology (TOSEM) , vol. 2, no. 3, pp. 270–285, 1993.

[4] R. Lachmann, "Machine learning-driven test case prioritization approaches for black-box software testing," in The European test and telemetry conference, Nuremberg, Germany, 2018.

[5] H. Jahan, Z. Feng, S. Mahmud, and P. Dong, "Version specific test case prioritization approach based on artificial neural network," Journal of Intelligent & Fuzzy Systems, vol. 36, no. 6, pp. 6181–6194, 2019.

[6] M. Bagherzadeh, N. Kahani, and L. Briand, "Reinforcement learning for test case prioritization," IEEE Transactions on Software Engineering, 2021.

[7] M. Waqar, M. A. Zaman, M. Muzammal, and J. Kim, "Test suite prioritization based on optimization approach using reinforcement learning," Applied Sciences, vol. 12, no. 13, p. 6772, 2022.

[8] D. Paterson, J. Campos, R. Abreu, G. M. Kapfhammer, G. Fraser, and P. McMinn, "An empirical study on the use of defect prediction for test case prioritization," in 2019 12th IEEE Conference on Software Testing, Validation and Verification (ICST) . IEEE, 2019, pp. 346–357.

[9] M. Mahdieh, S.-H. Mirian-Hosseinabadi, K. Etemadi, A. Nosrati, and S. Jalali, "Incorporating fault- proneness estimations into coverage-based test case prioritization methods," Information and Soft- ware Technology, vol. 121, p. 106269, 2020.

[10] Y. ShAo, B. Liu, S. WAng, and P. XiAo, "A novel test case prioritization method based on problems of numerical software code statement defect prediction," Eksploatacja i Niezawodno´s´c, vol. 22, no. 3, 2020.

[11] M. Mahdieh, S.-H. Mirian-Hosseinabadi, and M. Mahdieh, "Test case prioritization using test case diversification and fault-proneness estimations," Automated Software Engineering, vol. 29, no. 2, pp. 1–43, 2022.

[12] C. Lu, J. Zhong, Y. Xue, L. Feng, and J. Zhang, "Ant colony system with sorting-based local search for coverage-based test case prioritization," IEEE Transactions on Reliability, vol. 69, no. 3, pp. 1004–1020, 2019.

[13] R. Huang, Q. Zhang, D. Towey, W. Sun, and J. Chen, "Regression test case prioritization by code combinations coverage," Journal of Systems and Software, vol. 169, p. 110712, 2020.

[14] T. Alrawashdeh, F. ElQirem, A. Althunibat, and R. Alsoub, "A prioritization approach for regression test cases based on a revised genetic algorithm," Information Technology and Control, vol. 50, no. 3, pp. 443–457, 2021.

[15] T. Pospisil, J. Sobotka, and J. Novak, "Enhanced adaptive random test case prioritization for model- based test suites," Acta Polytechnica Hungarica, vol. 17, no. 7, 2020.

[16] M. L. Mohd-Shafie, W. M. N. Wan-Kadir, M. Khatibsyarbini, and M. A. Isa, "Model-based test case prioritization using selective and even-spread count-based methods with scrutinized ordering criterion," PloS one, vol. 15, no. 2, p. e0229312, 2020.

[17] H. Jahan, Z. Feng, and S. Mahmud, "Risk-based test case prioritization by correlating system methods and their associated risks," Arabian Journal for Science and Engineering, vol. 45, no. 8, pp. 6125–6138, 2020.

[18] A. Samad, H. B. Mahdin, R. Kazmi, R. Ibrahim, and Z. Baharum, "Multiobjective test case pri- oritization using test case effectiveness: Multicriteria scoring method," Scientific Programming, vol. 2021, 2021.

[19] D. Shin, S. Yoo, M. Papadakis, and D.-H. Bae, "Empirical evaluation of mutation-based test case prioritization techniques," Software Testing, Verification and Reliability, vol. 29, no. 1-2, p. e1695, 2019.

[20] J. Chi, Y. Qu, Q. Zheng, Z. Yang, W. Jin, D. Cui, and T. Liu, "Relation-based test case prioritization for regression testing," Journal of Systems and Software, vol. 163, p. 110539, 2020.

[21] J. Chen, L. Zhu, T. Y. Chen, D. Towey, F.-C. Kuo, R. Huang, and Y. Guo, "Test case prioritization for object-oriented software: An adaptive random sequence approach based on clustering," Journal of Systems and Software, vol. 135, pp. 107–125, 2018.

[22] V. Garousi, R. Özkan, and A. Betin-Can, "Multi-objective regression test selection in practice: An empirical study in the defense software industry," Information and Software Technology, vol. 103, pp. 40–54, 2018.

[23] Y. Chen and M.-H. Chen, "Multi-objective regression test selection," EPiC Series in Computing, vol. 76, pp. 105–116, 2021.

[24] M. Olsthoorn and A. Panichella, "Multi-objective test case selection through linkage learning-based crossover," in International Symposium on Search Based Software Engineering. Springer, 2021, pp. 87–102.

[25] S. Sutar, R. Kumar, S. Pai, and B. Shwetha, "Regression test cases selection using natural language processing," in 2020 International Conference on Intelligent Engineering and Management (ICIEM) . IEEE, 2020, pp. 301–305.

[26] R. Martins, R. Abreu, M. Lopes, and J. Nadkarni, "Supervised learning for test suit selection in continuous integration," in 2021 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW) . IEEE, 2021, pp. 239–246.

[27] M. Al-Refai, S. Ghosh, and W. Cazzola, "Model-based regression test selection for validating run- time adaptation of software systems," in 2016 IEEE International Conference on Software Testing, Verification and Validation (ICST) . IEEE, 2016, pp. 288–298.

[28] M. Al-Refai, S. Ghosh, and W. Cazzola, "Supporting inheritance hierarchy changes in model-based regression test selection," Software & Systems Modeling, vol. 18, no. 2, p p. 937–958, 2019.

[29] M. Al-Refai, W. Cazzola, and S. Ghosh, "A fuzzy logic based approach for model-based regression test selection," in 2017 ACM/IEEE 20th International Conference on Model Driven Engineering Languages and Systems (MODELS) . IEEE, 2017, pp. 55–62.

[30] W. Cazzola, S. Ghosh, M. Al-Refai, and G. Maurina, "Bridging the model-to-code abstraction gap with fuzzy logic in model-based regression test selection," Software and Systems Modeling, vol. 21, no. 1, pp. 207–224, 2022.

[31] Z. Long, Z. Ao, G. Wu, W. Chen, and J. Wei, "Webrts: A dynamic regression test selection tool for java web applications," in 2020 IEEE International Conference on Software Maintenance and Evolution (ICSME) . IEEE, 2020, pp. 822–825.

[32] H. Zhong, L. Zhang, and S. Khurshid, "Testsage: Regression test selection for large-scale web service testing," in 2019 12th IEEE Conference on Software Testing, Validation and Verification (ICST) . IEEE, 2019, pp. 430–440.

[33] L. Zhang, "Hybrid regression test selection," in 2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE) . IEEE, 2018, pp. 199–209.

[34] L. Chen and L. Zhang, "Speeding up mutation testing via regression test selection: An extensive study," in 2018 IEEE 11th International Conference on Software Testing, Verification and Validation (ICST). IEEE, 2018, pp. 58–69.

[35] J. Xu, Q. Du, and X. Li, "A requirement-based regression test selection technique in behavior- driven development," in 2021 IEEE 45th Annual Computers, Software, and Applications Conference (COMPSAC). IEEE, 2021, pp. 1303–1308.

[36] G. Guizzo, J. Petke, F. Sarro, and M. Harman, "Enhancing genetic improvement of software with re- gression test selection," in 2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE). IEEE, 2021, pp. 1323–1333.

[37] K.-c. Wang, T.-t. Wang, and X.-h. Su, "Test case selection using multi-criteria optimization for effective fault localization," Computing, vol. 100, no. 8, pp. 787–808, 2018.

[38] T. Saber, F. Delavernhe, M. Papadakis, M. O'Neill, and A. Ventresque, "A hybrid algorithm for multi-objective test case selection," in 2018 IEEE Congress on Evolutionary Computation (CEC) . IEEE, 2018, pp. 1–8.

[39] A. P. Agrawal, A. Choudhary, and A. Kaur, "An effective regression test case selection using hy- brid whale optimization algorithm," International Journal of Distributed Systems and Technologies (IJDST), vol. 11, no. 1, pp. 53–67, 2020.

[40] T. Yu and T. Wang, "A study of regression test selection in continuous integration environments," in 2018 IEEE 29th International Symposium on Software Reliability Engineering (ISSRE) . IEEE, 2018, pp. 135–143.