Natl. Acad. Sci. Lett.
https://doi.org/10.1007/s40009-025-01613-6

NEWS/VIEWS AND COMMENTS

Recent Trends in Regression Testing: Modeling and Analyzing the Critiques in Selection, Optimization, and Prioritization

Raja Marappan[1] · **Saraswatikaniga Raja**[2]

Abstract Nowadays, the software industry uses soft computing techniques to perform various tests and research to keep the software of better quality for customers or end users. *Regression testing* is needed to gain confidential insight into the software so that its performance is not affected. This testing is executed after updating the source code to ensure no new errors will be reported from the updates. As the software evolves, the *test suites* also increase in size, and hence, the computational cost is increased for the *test suites* execution. This research analyzes the recent trends in *regression testing* as different methods are developed to optimize the *test suite* using *selection*, *minimization*, and *prioritization*. The *test case selection* determines the suitable *test cases* for the necessary updates. *Test suite optimization* eliminates unnecessary or redundant *test cases* and reduces the total runs of *test cases*. Sequencing the *test cases* in an order is necessary to optimize the early detection of faults. This research analyzes the modeling and critiques in *selection*, *minimization*, and *prioritization* of recent techniques.

Keywords Software testing · Regression testing · Test suites · Test cases · Test cases prioritization · Fault detection

✉ Raja Marappan
professor.m.raja@gmail.com; m.raja@vit.ac.in

Saraswatikaniga Raja
saraswatikaniga.r2023@vitstudent.ac.in; raja.saraswatikaniga@gmail.com

1 School of Computer Science and Engineering (SCOPE), Vellore Institute of Technology, Chennai 600127, India

2 M.Tech Computer Science and Engineering (Business Analytics), School of Computer Science and Engineering, Vellore Institute of Technology, Chennai 600127, India

Regression testing (RT) is a complex and challenging activity when specific updates are required in the developed software such that the new updates do not disturb the existing software characteristics. Since the new logic with the updates may result in software defects, a series of test scenarios are to be executed every time, and the efficiency of the test should be optimized [1]. The proper troubleshooting should be performed when executing the RT. The scenario of RT is sketched in Fig. 1. RT tests can be performed based on the type of applications and attributes—mobile applications, web applications, functional, visual, security, and performance attributes [2, 3]. To conduct the usual checkups in the existing software, the RT requires the construction of automated *test suites*, identifying the specific test scenarios to execute on the affected segments in the updated releases [4, 5]. The RT is performed in the following situations that may result in unexpected characteristics: addition of new requirements in the existing components, addition of new functionalities, solving the defects, performance optimization of source code, adding patch fixes, releasing the new software version, making the updates in the user interfaces, configuration updates, integrating other systems with the working version [6–8]. The critical steps to be followed when constructing RT *test suites* is illustrated in Fig. 2.

Mathematically, RT is defined in terms of T, the sets of *test cases* (TCs) with R(T), and the corresponding results. Assume that the initial T is

$$T = \{t_1, t_2, t_3 \dots t_n\} \quad (1)$$

where each t_j, $1 \le j \le n$ is a TC, and n is the total number of TCs. Each t_j, $1 \le j \le n$ results in {fail, pass} such that

$$R(T) = \{r_1, r_2, r_3 \dots r_n\} \quad (2)$$

Published online: 04 February 2025

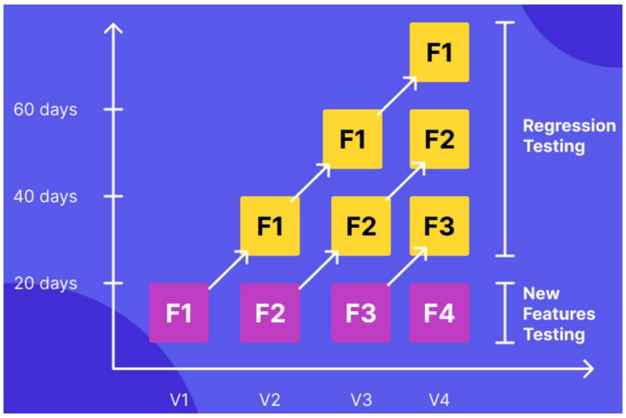

Fig. 1 Scenario of RT

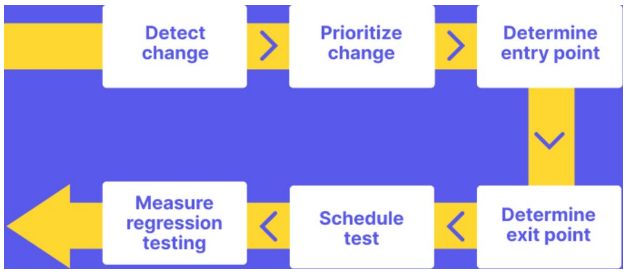

Fig. 2 Construction of RT *test suite*

where each $r_j \in \{fail, pass\}$, $1 \leq j \leq n$. The source code is modified with the re-execution of T to output new results,

$$R'(T) = \{r'_1, r'_2, r'_3, \ldots r'_n\} \tag{3}$$

The RT checks the symmetric difference between $R'(T)$ and $R(T)$.

Algorithm 1: General RT

```
1: Define the initialization of T as given in Eq. (1).
2: Execute each $t_j$, $1 \leq j \leq n$ and find R(T) as defined in Eq. (2) by applying the Table I strategies.
3: Re-execute T after code change and obtain R'(T) as defined in Eq. (3) by applying the Table I strategies.
4: Perform the regression analysis to find the symmetric difference between R'(T) and R(T) using
    $\Delta R = R'(T) \Delta R(T)$
(4)
    If $\Delta R = 0$, then there are no changes in the system behavior.
    Otherwise, there is a change in R'(T), signifies the regression in the system.
```

The classification of techniques—*TC selection*, *test suite optimization*, and *TC prioritization* are defined as follows [1–3].

The *TC selection* is formulated as follows:

Inputs: Program code P, P′—the updated version of P, *test suite* TS.

Problem: Find the subset TS′ of TS to test P′.

The *test suite optimization* is formulated as follows:

Inputs: *Test suite* TS, test requirements set $TR = \{TR_1, TR_2, TR_3 \ldots TR_n\}$that should be fulfilled to define the required adequate testing of P, and subsets of T, TS_1, TS_2, $TS_3 \ldots TS_n$, linked with each of the subsets of TR such that any one of the TCs TC_j in TS_i satisfy the requirement TR_i.

Problem: Find the set TS′ of TCs from T that fulfills all subsets in TR.

The *test suite optimization* properties are formulated as follows:

- Test conditions should be satisfied when the requirements in the set TR are satisfied.
- Each of the test requirements TR_i is satisfied by the TCs $TC_{j,}$ which is in TS_i, a subset of TS.
- The collection of TCs targets the set of TS_js.
- To optimize the *test suite*, TS′ must be the minimal target of the TS_js.
- Finding the solution to minimal targeting is an NP-complete nature.

The *TC prioritization* is stated as follows:

Input: *Test suite* TS, per(TS)—permutations set of TSs, and a function from per(TS) to the real numbers, as

$$f : per(T) \rightarrow R$$

Problem: To find $TS' \in per(TS)$ such that the following must be fulfilled:

$$(\forall TS'')(TS'' \in per(TS))(TS' \neq TS'')$$

$$f(TS') \geq f((TS''))$$

The *TC prioritization* performs the sequencing of the TCs for the properties optimization, for example, to detect the fault rates. This optimization finds the optimal

Table 1 Analysis of recent methods—*Test Suite Optimization*, *TC Selection*, *TC Prioritization* [1–16]

No	Recent methods	Analysis	Classification
1	Greedy heuristics	The worst-case execution time is $O(\|TS\|maxTS_i)$. The redundant TCs are minimally removed.	*Test suite optimization*
2	Dual minimal hitting set	The method extends to set cover problems with no fixed ordering of TCs. The rate of reduction of TSs using mutation is 30%.	
3	Spanning set	The construction of decision graphs is lacking when mapping TR into entities in the decision graphs. The problem is reduced to an optimal spanning set.	
4	Modified greedy	The lattice is constructed, and the greedy method is applied in the modified TCs.	
5	Selected redundancy	The method defines additional TRs by checking for redundant TCs. The larger TSs are produced.	
6	Bi-criteria approach	Lack of history of fault detection and applied linear programming optimization.	
7	Multi-criteria approach	Several heuristics are compared and applied with the hybrid and prioritized optimization.	
8	Time aware & optimization	This method applied the Pareto-based multi-objective optimization.	
9	Call-stack coverage	The stacking concept is implemented to represent TSs with the extension to apply GUI-based testing.	
10	Operational abstraction	The dynamic characteristics are represented, and the model performance needs to be improved.	
11	Dynamic invariant detector	High fault detection rates with the program invariant detection.	
12	Minimization	There is no uncertainty in fault detection with the definition of failure TCs.	
13	Delta debugging	The size of failed TCs is reduced.	
14	Black box	The variables that affect the outcomes are identified, and the combinatorial TCs are generated and linked with the interaction testing.	
15	Dependence analysis	Extension of finite state machines. Automatic model changes are identified.	
16	Logic criterion	Boolean variables are used for the predicate testing.	

Table 1 (continued)

No	Recent methods	Analysis	Classification
17	Modified revealing	The fault-revealing TCs are identified, and the selection is based on weaker criteria. The testing of controlled regression is assumed, and controlling all environments is not always practical.	*TC selection*
18	Safe regression	Unsafe for all possible faults.	
19	Integer programming	The relation of matrices between TCs and the segments of the code is defined by treating the functions as segments. The changes in the control flow of P′ have not been dealt with. The execution of all TCs is required when the control flow structure is changed.	
20	Data flow analysis	The updated or removed pairs of definitions are obtained and used in P′. TCs are selected to execute the pairs in P′.	
21	Slicing	The method results in high cost without dataflow analysis and does not detect unrelated changes.	
22	Data flow regression	They are applied in spreadsheet applications with the construction of cell relation graphs.	
23	Symbolic execution	The values of variables are defined as symbols. The code analysis is performed to identify the input partitions. The method is costly, and the complexity of symbolic execution is to be reduced. Still, there are many challenging issues in the pointer arithmetic operations.	
24	Dynamic slicing	The subset of slices is executed, and the relevant and approximate slices are used. The control flow graph should not be changed during the execution.	
25	Graph walk	There is a lack of data dependency relationships when applying the depth-first traversal. The method is unsuited for inter-procedural *TC selection* and applied in web service testing.	
26	Textual difference	Unix tool-based method identified the modified segments. Preprocessing is required to convert the source code into canonical forms.	
27	Path analysis	Algebraic expression is needed to construct exemplar paths in P and P′. The test paths are classified as canceled or new. The new or canceled test paths TCs are not selected. This is not a safe method.	
28	Modification	The method partitions the system under test into program entities and P′ into program entities. It extends graph walks. There are issues in pointer handling, languages without type coercion, and pointers.	
29	Firewall	The firewalls are constructed around the system modules. The method is applied in object orientation and banking systems.	
30	Cluster identification	Identification of clusters is the primary task. Selecting TCs from the unmodified portions results in a lack of precision.	
31	Design based	Traceability is required between the TCs and design. The design should be implemented using a unified modeling language.	
32	Multi-objective decision [14]	The decision has multiple objectives, such as evaluating and quantifying technical debts. The analytic hierarchy is combined with the similarity-based order preferences. Further enhancements are required using quality metrics integration to the different industrial TCs.	*TC selection*
33	Semantic representation[15]	The ontology-based mapping is defined to select the TCs dynamically. The higher chance TCs are executed first for the industrial automation.	*TC selection*

Table 1 (continued)

No	Recent methods	Analysis	Classification
34	Coverage based	This method applied various fault detection surrogates.	*TC prioritization*
35	Interaction testing	The combinations of components-based testing are integrated. The covering array is constructed to define the interactions of components. The TC permutation improves the coverage of interaction.	
36	Distribution based	Some dissimilarity metrics are defined for TC profiles. The clustering of TCs is done based on similarity. Early fault is detected using the TCs with a high priority. The hybrid strategy is repeatedly used for essential coverage prioritization.	
37	Human-based	The boosting technique is developed to combine the learners, and the rank boost algorithm performs rank learning. The cyclomatic complexity and the statement coverage metric are evaluated. Still, there are issues in scalability.	
38	Probabilistic	The method is developed by defining the probabilities of the TCs in Bayesian networks.	
39	History based	Matrix analysis is performed with the association clusters.	
40	Requirement based	The method maps TCs to the software requirements. The method has a weakness in setting subjective values of requirement properties.	
41	Model-based	Selective and random prioritization are applied with dependence-based heuristics. The performance depends on the model specification quality.	
42	Cost aware	Each fault is equally assumed to be severe, and the cost of each TC is the same.	
43	Ant colony optimization	Multi objectives are designed to achieve the highest coverage of faults. The correctness versus cost-effectiveness analysis is made. The model results in intelligent exploitation, enhanced exploration, quick fault coverage from the *test suite*, and *test site minimization*. The experiments are evaluated on the smaller piece of code.	
44	Quantum-based swarm optimization	Perturbation is performed on the combinatorial *TC prioritization*. The method results in detection loss with high fault and is to be extended to large-scale real-world applications.	
45	Java-based regression test	The test compares the techniques HyRTS, OpenClover, Ekstazi, and STARTS. The expected fault detection capability is 8.75% lower than the actual TS version.	
46	Non-iterative identification	Using the sensitivities-based test selection, the fault compensation technique incorporates the uncertainty and tolerance influence. The soft fault is identified and not the hard fault. The method is applicable for laboratory and model testing.	
47	Multi-criteria optimization	This method merges the Pareto approach with the heuristic criterion slower for open-source solvers and is applied in smaller systems. There are issues with robustness for larger systems. The functions are added and removed during the investigational search. More experimental evidence is required for complexity analysis. The scalability versus performance should be evaluated. The model's applicability and empirical support are to be extended. The user's experience in selecting the Pareto front in the selection process will be investigated. For automated selection, the selection of metrics to be analyzed.	
48	Continuous integration	The data model is constructed to capture the relations and data sources by defining comprehensive features. Continuous integration develops tools and methods to gather open-source software systems' various keys and tools. The time needed to collect data versus the effectiveness of machine learning techniques is still to be analyzed.	
49	Integrated optimization	Integrated optimization reduces the testing time versus failure detection effectiveness. The integrated model is constructed to optimize the sequencing and selection using precedence constraints in the TCs. The TC sequencing is done as a job-based and time-index construction. When sparsity is small, the decreasing time ratio increases the computational time. The model complexity is affected by sparsity and decreasing time ratio.	

Table 1 (continued)

No	Recent methods	Analysis	Classification
50	Multi-objective searching [9]	The software microbenchmarks are tested using several search-based objectives. The objectives are based on simple greedy strategies. The maximum runtime overhead obtained is 1%.	*TC prioritization*
51	Multi-goal PSO [10]	The multi-goal-based PSO with different ordering strategies is defined to maximize fault coverage by up to 85%.	
52	Law of minimum (LoM) [11]	The probabilistic LoM is applied to order and slicing. The granularity should be replaced with new strategies and various analysis tools.	
53	Cluster-based adaptive [12]	Cluster-based adaptive method is designed for fault detection with the limitations—needed to enhance system quality, fault detection errors, define fault similarity factors with correlation, diverse clustering and threshold analysis for further enhancement.	
54	Contextual approach with ML [13]	The contextual ML approaches are designed to execute in the continuous integration cycle. The enhancements are required to investigate Android environments and real scenarios with contextual approaches.	
55	Semantic representation [15]	The functional size determines the prioritization of ontology-based selected TCs to help with industrial automated RT.	
56	Ensemble and time average [16]	Extended principles are further required to implement different computing libraries.	

permutation TC sequence, assuming the TCs are executed in some permutation order. The testing activity may be terminated arbitrarily since TC selection is not involved during prioritization.

The TCs are classified into reusable, retestable, obsolete, new structural, and new specification.

- Reusable: Execute the program segments without changing the original and updated versions.
- Retestable: Execute the program segments of P, which are updated in P′.
- Obsolete: Input–output relations are incorrect because of the specification's modification.
- New structural: Execute the structured and modified segments in P′.
- New specification: Execute the updated specification segments in P′.

The critiques in the *selection*, *optimization*, and *prioritization* problems are analyzed in Table 1.

The complexity and costing measures of *test case selection*, *optimization*, and *prioritization* of RT are analyzed. The *TC selection* takes O(n) or O($n$log$n$) complexity. The TC execution timer is defined as T(j) for the selected TCs, $1 \leq j \leq n$. The *TS optimization* ranges from O(n^2) to exponential complexities. The coverage metrics include the fault detection probability and code coverage. The costing measures include the development and execution costs, the total TC cost, and the total testing & optimization costs, which include the time needed to run the optimized algorithm with the computational resources.

This research article provides a modeling and a complete analysis of the recent trends in RT with *TC selection*, *optimization*, and *prioritization*. It also sketches the relationship between the three areas of RT and shows the application areas with the corresponding critiques. Some interesting properties are observed from the comparative critiques of the different methods with the importance of *TC prioritization*. The expected performance measures are analyzed for the recent methods defined in Table 1. Percentage of code covered by the TCs: 60%–85%, fault detection: 60%–90%, the average percentage of faults detected: 60%–80%, *test suite* size reduction: 50%–70% and risk exposure reduction: 40%–70%. Hence, the research is expected to focus on the balances and tradeoffs among different criteria, considering values versus costs when evaluating RT techniques. The design of optimal algorithms is expected further to work on the three classifications of RT techniques for large-scale real-time projects.

Funding No funding was received.

Declarations

Conflict of interest The authors declare that no conflicts of interest in publishing this research article.

References

1. Yoo S, Harman M (2007) Regression testing minimisation, selection and prioritisation: a survey. Softw Test Verif Reliab 22:67–120
2. Singhal S, Jatana N, Sheoran K, Dhand G, Malik S, Gupta R, Suri B, Niranjanamurthy M, Mohanty SN, Ranjan Pradhan N (2023) Multi-objective fault-coverage based regression test selection and prioritization using enhanced ACO_TCSP. Mathematics 11:2983. https://doi.org/10.3390/math11132983
3. Bajaj A, Abraham A, Ratnoo S, Gabralla LA (2022) Test case prioritization, selection, and reduction using improved quantum-behaved particle swarm optimization. Sensors 22:4374
4. Shin MK, Ghosh S, Vijayasarathy Leo R (2022) An empirical comparison of four Java-based regression test selection techniques. J Syst Softw 186:111174
5. Hałgas S (2023) Soft fault diagnosis in linear circuits: test selection and non-iterative identification procedure. Measurement 217:113061. https://doi.org/10.1016/j.measurement.2023.113061
6. Dobslaw F, Wan R, Hao Y (2023) Generic and industrial scale many-criteria regression test selection. J Syst Softw 205:111802
7. Yaraghi AS, Bagherzadeh M, Kahani N, Briand LC (2023) Scalable and accurate test case prioritization in continuous integration contexts. IEEE Trans Softw Eng 49(4):1615–1639. https://doi.org/10.1109/TSE.2022.3184842
8. Zhang H, Li Y-F (2022) Integrated optimization of test case selection and sequencing for reliability testing of the mainboard of Internet backbone routers. Eur J Oper Res 299(1):183–194. https://doi.org/10.1016/j.ejor.2021.06.028
9. Laaber C, Yue T, Ali S (2024) Evaluating search-based software microbenchmark prioritization. IEEE Trans Software Eng 50(7):1687–1703. https://doi.org/10.1109/TSE.2024.3380836
10. Nazir M, Mehmood A, Aslam W, Park Y, Choi GS, Ashraf I (2023) A multi-goal particle swarm optimizer for test case prioritization. IEEE Access 11:90683–90697. https://doi.org/10.1109/ACCESS.2023.3305973
11. Ufuktepe E, Tuglular T (2023) Application of the law of minimum and dissimilarity analysis to regression test case prioritization. IEEE Access 11:57137–57157. https://doi.org/10.1109/ACCESS.2023.3283212
12. Wang X, Zhang S (2024) Cluster-based adaptive test case prioritization. Inf Softw Technol 165:107339. https://doi.org/10.1016/j.infsof.2023.107339
13. Da Roza EA, Do Jackson A, Lima P, Vergilio SR (2024) On the use of contextual information for machine learning based test case prioritization in continuous integration development. Inf Softw Technol 171:107444. https://doi.org/10.1016/j.infsof.2024.107444
14. Zarrad A, Bahsoon R, Manimaran P (2024) Optimizing regression testing with AHP-TOPSIS metric system for effective technical debt evaluation. Autom Softw Eng 31:58. https://doi.org/10.1007/s10515-024-00458-5
15. Sakhrawi Z, Labidi T (2024) Test case selection and prioritization approach for automated regression testing using ontology and COSMIC measurement. Autom Softw Eng 31:51. https://doi.org/10.1007/s10515-024-00447-8
16. Zhang B, Zhang C, Hu X (2024) Automated regression test method for scientific computing libraries: illustration with SPHinXsys. J Hydrodyn. https://doi.org/10.1007/s42241-024-0042-6