Computers, Materials & Continua
DOI: 10.32604/cmc.2022.028386
Article

Tech Science Press

Managing Software Testing Technical Debt Using Evolutionary Algorithms

Muhammad Abid Jamil* **and Mohamed K. Nour**

Department of Computer Science, Umm Al-Qura University, Makkah, Saudi Arabia
*Corresponding Author: Muhammad Abid Jamil. Email: majamil@uqu.edu.sa
Received: 09 February 2022; Accepted: 23 March 2022

Abstract: Technical debt (TD) happens when project teams carry out technical decisions in favor of a short-term goal(s) in their projects, whether deliberately or unknowingly. TD must be properly managed to guarantee that its negative implications do not outweigh its advantages. A lot of research has been conducted to show that TD has evolved into a common problem with considerable financial burden. Test technical debt is the technical debt aspect of testing (or test debt). Test debt is a relatively new concept that has piqued the curiosity of the software industry in recent years. In this article, we assume that the organization selects the testing artifacts at the start of every sprint. Implementing the latest features in consideration of expected business value and repaying technical debt are among candidate tasks in terms of the testing process (test cases increments). To gain the maximum benefit for the organization in terms of software testing optimization, there is a need to select the artifacts (i.e., test cases) with maximum feature coverage within the available resources. The management of testing optimization for large projects is complicated and can also be treated as a multi-objective problem that entails a trade-off between the agile software's short-term and long-term value. In this article, we implement a multi-objective indicator-based evolutionary algorithm (IBEA) for fixing such optimization issues. The capability of the algorithm is evidenced by adding it to a real case study of a university registration process.

Keywords: Technical debt; software testing optimization; large scale agile projects; evolutionary algorithms; multiobjective optimization; indicator-based evolutionary algorithm (IBEA); pareto front

1 Introduction

In this research work, we presume a testing methodology that will assist IT companies to optimize the test technical debt in their large agile projects [1]. In continuous maintenance and testing, to implement new features, it is required to update the sprints continuously, fix errors, and make changes according to user requirements. Therefore, in large-scale agile projects, it is important to prioritize new features, technical debt, and bug testing simultaneously [1,2]. However, the technical debt items can become the cause of prolonging the testing process and there is no proper measurement approach to

calculate the financial cost of testing [3,4] making accountability difficult. Additionally, as mentioned in [5,6], the test case redundancy is a complicated process, and this process requires more time to resolve the testing problems.

This article proposes a novel technique to assist the testing optimization process when there is a need to implement new features or to pay back the test debt. The search space for new features, test cases, and debt items can extend into large numbers for large-size projects, In this case, a deterministic approach would be hard to implement [7]. Consequently, we proposed an indicator based evolutionary algorithm (IBEA) to search for the best solutions in the problem space [8]. In our work, there exist multiple conflicting objectives (like feature maximization and testing cost minimization) that need to be optimized. The Pareto optimality technique can help resolve or optimize the multiple objectives problems. This technique is a good approach in terms of the regression test case selection problem as the testers are willing to generate optimal results with different conflicting constraints [9]. It is also helpful in the trade-off between feature coverage (maximization) and minimization of the accumulated test debt cost [10,11].

Different researchers have proposed different techniques regarding the discovery of technical debt, its volume, and how to treat it [12–15]. The main disadvantage of these approaches is that they do not provide comprehensive details as to how future development can be influenced if we delay these new features or pay back test debt. In addition, our research will explore how to generate a certain set of solutions in terms of software testing optimization. We will also explore how to supply information concerning adding new feature requests and test debt. One of the main concerns about software testing is the large size of test cases. Hence, to perform the testing completely, there is a need to test all features and components. This is why complete testing becomes difficult when there is an increase in product variants or features [16]. Hence, tests in a large-scale system increase with the increase in the number of developed products [17]. The major concern at this level is how to minimize the redundant tests to reduce the testing effort. To reduce the test redundancy, the association between the developed versions and new desired features in large-scale projects should be considered [16].

This research article has been structured as follows. The background has been discussed in the Section II. Section III describes the implementation of the proposed approach as well as the experimentation setup and results. Lastly, we conclude with a reference to future work in Section IV.

2 Background

2.1 Technical and Tests Debt

The test debt is accumulated when we make incorrect technical decisions, either deliberately or unknowingly. These technical debts are generated due to poor software development practices [18]. For large-scale agile projects, this terminology has acquired considerable importance [19,20]. In particular, the fostering of agile approaches becomes a cause of technical debt, hence there is a need to handle these debts skillfully [14]. Due to these technical debts, there is a need to optimize the testing artifacts (test cases) to obtain optimal results for the software testing process in any organization. To persuade the long-term success of software testing, the interest of technical debt needs to be paid. However, the interest of technical debt can be unpredictable in contrast with financial debt (or testing cost debt). For instance, the developers care less about the technical debt which exists in components as compared to the technical debt present in a core system [20]. As we discussed earlier the test debt is unpredictable due to technical debt. Hence there is a need to minimize the testing cost to achieve the required optimized results [9,12]. The implementation and testing of new features add new costs to the large-scale systems where technical debt occurs.

2.2 Multi-Objective Optimization (MOO)

In the optimization technique, the optimal solutions are chosen from a set of alternatives with some applicable constraints. The optimization for a single objective relies on a single criterion to find the best optimal solution, but MOO uses two or more measures to choose the most perfect solution in the case of multi objectives. These measurements are considered as objectives and there can exist a conflict between these objectives. As a result, no single good solution exists; therefore, a range of optional solutions occur with different intensities of the trade-off between the objectives [21,22]. Decision-makers are, therefore, faced with a challenge when deciding which solution to be considered as the best one. Many MOO approaches, fortunately, can help with the process of decision-making by using the Pareto optimal concept, which makes it easier to explore and evaluate the set of best possible alternatives [23]. The solutions which are not dominated by other solutions in the Pareto front are referred to as non-dominated Pareto optimal solutions. These solutions are reasonable because if an objective is selected with the best values then other objective values can be compromised. In the Pareto front, there is an optimal set of solutions which is the target of MOO methods [23,24]. Because for many objectives' optimization, the objective functions have conflicting issues while these objectives are linked with different metrics. So, these types of problems are well managed by the Pareto dominance approach. This approach has produced a set of acceptable good solutions. These solutions produce estimation to form a Pareto Front, which constructs various non-dominated solutions.

In these solutions, if X solution dominates another solution Y, it means that there is at least one objective in X having better value than correlative objective in Y, while other objectives in X may be equal to the values of Y. The search process is initiated with a population made up of some solutions that exist in the search space. The best trade-off solutions are generated by compromising the objectives. The technical and test debt management can also be considered as a multi-objective problem. It is not possible to get the optimal solution for one objective while we try to optimize more than one objective. As a result, the Pareto optimality idea is utilized in [9,10].

2.3 Search Based Software Testing

The requisition of optimization algorithms assists in solving issues concerned with software testing and software engineering and is known as Search-Based Software Testing (SBST) or Search-Based Software Engineering (SBSE). On benchmark challenges, new techniques, algorithms, and tools have been implemented and verified. Miller published first research work on Search-Based Software Testing (SBST) [25], and this research area has matured significantly in the recent decade, with multiple research papers and tools [26–29] aimed at enabling test case generation and test suite quality assessment [30–33]. The purpose of search-based software testing is to improve several criteria, such as automatically generating test cases for a software system. In different situations, evolutionary algorithms are used in developing the best optimal software test suites. The search result is frequently achieved by defining a fitness function that assists the searching process with the likelihood of success [7]. The main objective of software testing is to identify the flaws in the software. An exhaustive testing technique can be used to achieve sufficient assurance that the software system is reliable and free of flaws. However, as software features grow, the testing complexity increases in the form of a comprehensive testing approach, making exhaustive testing impractical. Instead, testers employ search-based strategies that make effective use of testing techniques even for large-scale agile projects [7].

Fig. 1 shows the different types of test artifacts which are used in the testing process. However, in our research the test case artifact has been selected. To optimize the software testing artifacts like test case generation as shown in Fig. 1, different search-based techniques like genetic algorithms, simulated annealing, and local search approaches have been utilized. Experiments are used in several techniques to achieve the goal of an optimization procedure that relies on a set of designated problems that need to be resolved [7]. In this article, we try to prove that the proposed SBST can help improve testing artifacts and automate the verification and validation (V&V) of large-scale agile software to minimize technical debt issues.

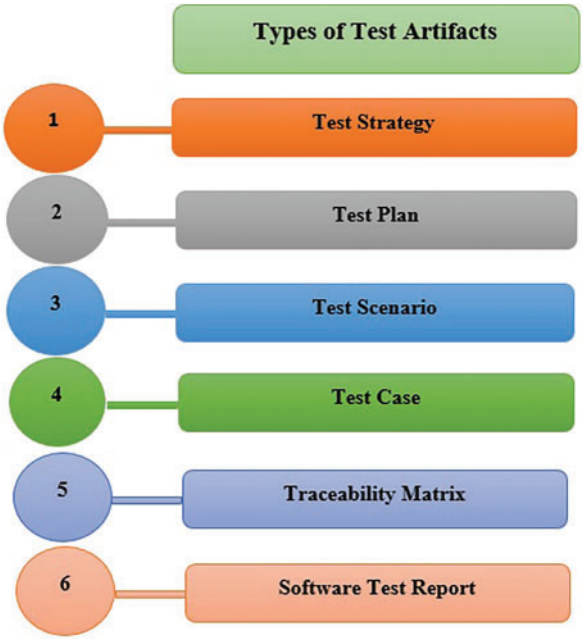

Figure 1: Types of test artifacts

2.4 Indicator-Based Evolutionary Algorithm (IBEA)

The indicator-based evolutionary algorithm (IBEA) employs a diverse set of indicators. It works with the user's preferences. Unlike other algorithms, there is no requirement for a diversity preservation technique like fitness. Because the IBEA uses the fitness ranking criterion technique, it can achieve good results [34]. The details and working flow of the IBEA algorithm can be explored in [35]. The working idea of IBEA is relying on preferences formalization for continuous generalizations in terms of the dominance relation. Due to the arbitrary size of the population, IBEA is considered more general and robust as well, because it uses a comparison technique for pairs of individuals [36]. There is an improvement in the quality of the generated Pareto set approximation in terms of defined optimization goals with the usage of the IBEA technique. The work of the IBEA has been shown in Fig. 2 and the algorithm procedure is described in detail as well.

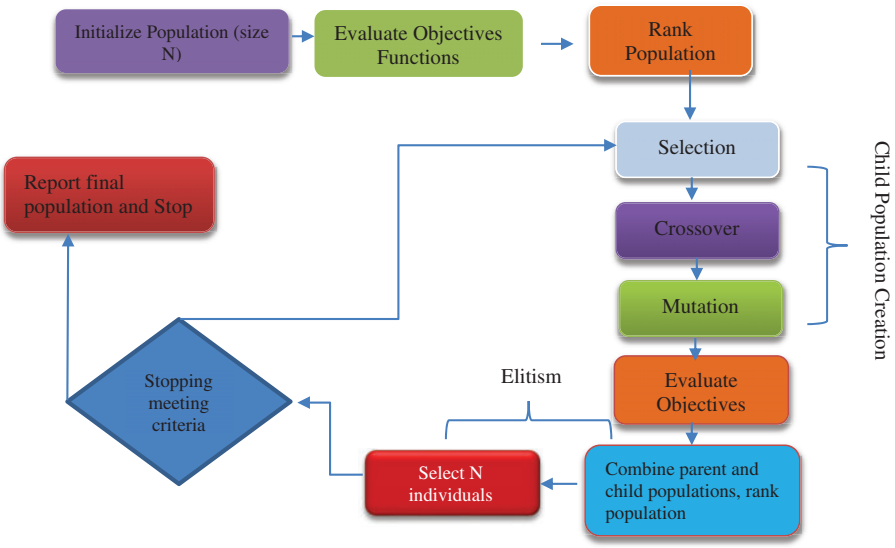

Figure 2: Workflow of IBEA

Algorithm 1: Stepwise Algorithm for IBEA

Input: α, N, k

Define initial population P of size α
Generation counter m to 0
while *population* P not greater than α do
    foreach *Population Individual* do
        Measure individual fitness values
    end for each
Select individual x ε P (smallest fitness value)
Eliminate x from the P
Revise the fitness values for other individuals
  if alternate stopping criterion satisfied **then**
    Selection of non-dominated individuals in P
  else
    Fill the mating pool P′ while running binary tournament selection with the replacement on P
    Use crossover and mutation operators on pool P′ and sum up the generated offspring to P
    Addition the generation counter m and start again from line 5
  end if
end while

3 Proposed Approach Implementation

3.1 Problem Definition

In our research approach, we believe the technical debt in terms of testing artifacts occurs in the form of a large number of test cases. As we discussed earlier, poor implementation can cause technical debt which results in test debt. Let $F = \{f_1, f_2, \ldots, f_n\}$, be the set of n proposed features

and against these n number of features we assume a certain number of test cases after implementation. Let $TC = \{tc_1, tc_2, \ldots, tc_n\}$, be the set of test cases to test these features. The testing cost to test the features is represented as $C = \{c_1, c_2, \ldots, c_n\}$ for all n number of test cases. We presume that the impact of technical debt items in terms of test cases on each of the new features needs to be evaluated in this work. For example, consider the tests debt item T (i.e., test case), which is associated with testing a component C. If test debt item T is not established, it is necessary to put in extra work to implement other features. The following two objectives need to be optimized in our proposed work.

3.1.1 Tests Debt Cost (Minimization)

The features testing cost can be minimized with the following objective function.

$$J_1(t) = cost\ (t), \tag{1}$$

The above objective function guarantees that the feature testing cost can be minimized. The different resources and costs are required by each feature to be tested. In this regard, each feature is allocated a value indicative of an assessment of features testing cost. Therefore, the total cost is presumably equivalent to the cost of testing for one module. In other words, if the cost of the feature *fi* is symbolized by cost (*fi*), then a module $C = \{\pm f1, \ldots, \pm fn\}$ possesses a cost equal to [9]:

$$CS_K = \sum_{i}^{n} \sum_{j}^{m} \lambda(PF(i,j)$$

$$\cos t\,(c) = \sum_{i=1}^{n} P\,(i) \cos t\,(i)\,, \text{ where } P\,(i) = \begin{cases} 1 & if + f\ (\mathrm{i}) \\ 0 & if - f\ (\mathrm{i}) \end{cases} \tag{2}$$

The $+fi$ feature means that this feature is selected for a particular sprint while the $-fi$ feature means that this feature is not selected for development for the same sprint. The cost of the tests is composed of summing up the cost of a test suite. Hence, the equation given below provides the cost of $CS = \{C1, \ldots, Cm\}$:

$$\cos t\,(CS) = \sum_{i=1}^{m} \text{cost}\,(C_i) \tag{3}$$

3.1.2 Feature Coverage (Maximization)

The population is test Suite $\{C_1, C_2, C_3 \ldots\ldots C_k\}$, use count number of positive features in a vector of Configuration $C_i = \{+f_1, \ldots., +f_n, -f_1 \ldots\ldots, -f_n\}$. The goal of this function is to count features included using the MOEAs like IBEA. This objective can be defined mathematically as:

$$J_2 = \sum_{i}^{n} (PF(i,j) > 1 \tag{4}$$

where $\lambda(PF\,(i,j) = 1$, if, else $= 0$

The above mathematically define two objectives that need to be optimized in terms of testing process optimization.

3.2 Impact of Indicator Based Evolutionary Algorithm (IBEA) on Proposed Approach

To be able to use IBEA, the chromosome must be encoded in terms of candidate solution as shown in Fig. 3. For chromosome representation, a bit vector is defined with a string of 0 s and 1 s values, where every bit expresses a gene on the chromosome. Each test debt and feature item in the list has an entry in the string that is prepared for a sprint. In the case of our problem, a single-point crossover operator would be applicable [7]. The mutation process is carried out on chromosomes by inserting or randomly mutating a bit. These two operators might generate a result that surpasses the feature testing cost. As we mentioned above, with the increase in the number of features, the size of test cases also increases. To ensure that the solution does not exceed the sprint testing cost, a feature is not chosen, or a test case is chosen in a random manner (i.e., setting a gene to zero). As given in Eq. (3), a solution testing cost would be the general cost allotted for a sprint.

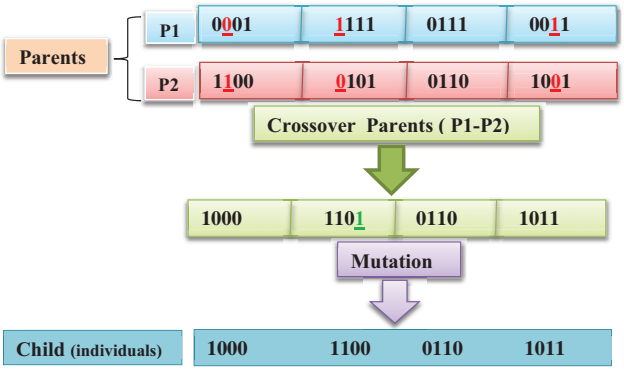

Figure 3: Encoding of chromosome in binary representation

For an evolutionary algorithm, the crossover and mutations probabilities are provided as parameters to an algorithm. Each solution is evaluated by the fitness function in terms of features (maximization) and testing cost (minimization). Both use a separate objective function *J1* (testing cost minimization) and *J2* (features maximization) which are implementations of Eqs. (3) and (4), respectively.

After the realization of the IBEA algorithm, the individuals would endure for the next generation to be adopted in the following way. Choose an individual with the smallest fitness value, remove it from the population and consider only those individuals (solutions) which are not dominated in the population. Otherwise, IBEA would perform a binary tournament selection process as mentioned in line 13 of IBEA algorithm in Section 2.4.1. Again, crossover and mutation processes would be repeated to generate the final individuals in the population.

For the Pareto front, there would be usually a random different number of selection of individuals (solutions) from the population. Selecting a single Pareto front isn't adequate to generate an adequate population as shown in Fig. 4. As mentioned above, the selection procedure is repeated until we obtain the necessary minimum of optimal individuals (solutions). So, the general concept will be to generate the best trade-off optimized solutions for the objectives *J1* and *J2* by the Pareto Front. Hence, the optimized results will be considered in such a way that if we have to maximize the number of features (*J2*) then we have to compromise the testing cost (*J1*) and vice versa. The ranks are based on the principle of non-dominated sorting (Pareto dominance).

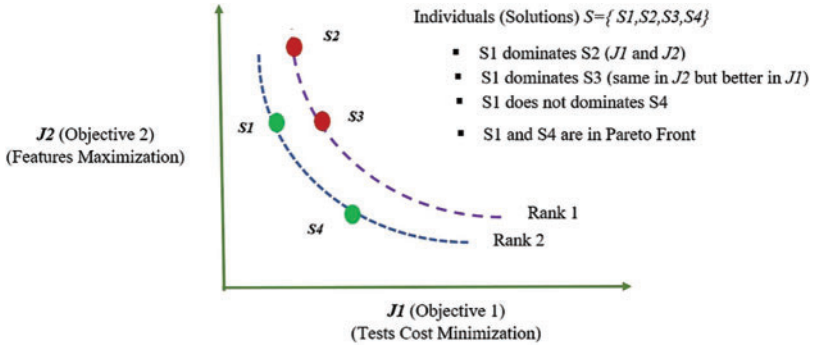

Figure 4: General concept of pareto front

3.3 Experiments Setup and Results Analysis

We verified the applicability of our proposed approach with the student course registration project conducted at the Computer Science Faculty of Umm Al Qura, Makkah, KSA. This project needs to improve while adding the new features in the final product. For development, the Scrum methodology has been adopted [37] and it was needed to be completed in different sprints. To test every feature there was the need to assess the testing cost described earlier before starting the implementation of the project. This information has been clearly described in the product backlog with other artifacts information. The testing overheads that arise after a new feature addition request become the testing debt and these debts. In terms of testing, they do not deliver any business value, Rather, they tend to degrade the quality of the project. Hence, the request to add new features can cause overhead for testing process because new test cases need to be designed and executed. In the perspective of quality assurance, for better project management in such agile projects, we need to update the product backlog by adding new test cases. Hence, we have a large number of test cases that need to be optimized or minimized while covering the maximum number of features during the testing process. We initially examine what features or perhaps test debt in terms of test cases need to be picked up for the last sprint, taking into account for the best trade-off between feature coverage and test case minimization.

In the experimental setups adopted in [38,39], the initial population size is set at 200, and the number of generations is set at 500 for the IBEA algorithm. The crossover and mutation rate values are considered 60% and 30% respectively. The initial values (at generation 1) and the final values (at generation 500) of both the three sub-objectives are measured against every 30 runs. The parameter settings are specified before the execution of the experiment. The fixed parameter settings are used to set the fact that the outcomes are consistent instead of accidental. As a result of the arbitrary dynamics of IBEA, the algorithm is run thirty times [9,40]. As shown in Fig. 5, it is observed that the IBEA algorithm has generated non-dominated solutions in the Pareto Front. From Fig. 5, it is clear that a solution is considered a good one if it has optimal value concerning objective feature coverage (maximization) and objective tests cost (minimization). For example, solution *S1* has a high value for the feature coverage. However, the tests cost value that may be acquired by selecting this particular solution would be extremely small. IBEA is the most effective since it makes the most of the user preferences [41].

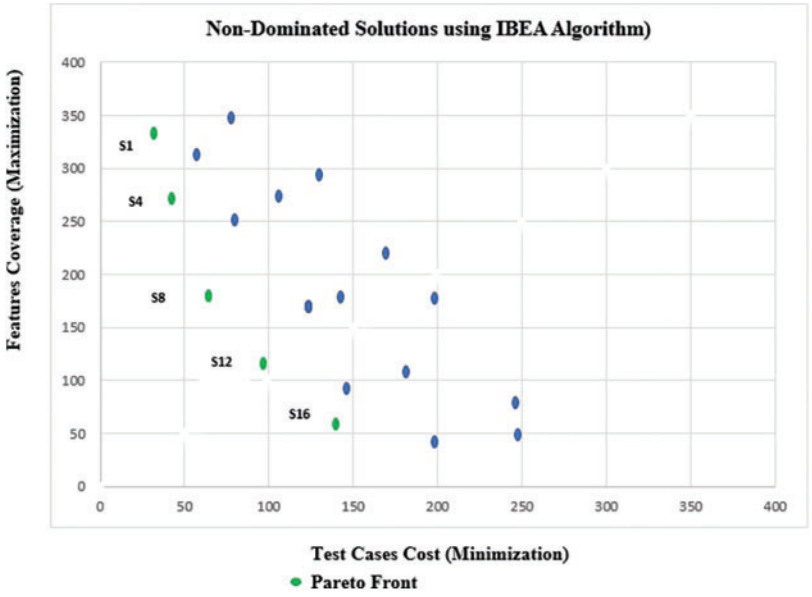

Figure 5: Best pareto front solutions

The reason for adopting this specific solution is to select only features that need to be implemented in the sprint. Similarly, the *S4* solution has the same reason for selection. For solutions *S8* and *S12* both feature coverage (maximization) and test debt (minimization) obtained the medium level of values. This means that in the sprint there is a need to tackle the selected features and test debt. But it is observed from Fig. 5, it is not easy to obtain a solution that has both values optimal. In this particular circumstance, rather than generating one solution, it is good to generate a set of solutions that are excellent concerning both objectives. If we consider the same behavior of results then we can know what nature of Pareto fronts would be generated for the future sprints. The results demonstrated that the feature coverage value for solution *S16* would be higher than *S1* and *S4* as the debt tests need to be fixed for a future sprint. In this way, testers can stay away from supplementary exertion required by the features. Hence, additional features can be supplied within the assigned cost.

This sort of study can assist project managers to obtain additional facts in various situations, such as, what would happen if we concentrate more on new feature coverage, or it would be a good approach to optimize the test debt cost or postpone the tests debt for the future sprint. More significantly, the research's key endowment is a real, two-dimensional multi-objective search, which uses the Indicator-Based Evolutionary Algorithm (IBEA) to bring every researcher's preferences in the concentration without aggregating, and then uses Pareto dominance to include user preferences. We present novel results discovered with IBEA and apply up to two objectives optimization, namely: maximization of features and minimization of test case cost. IBEA's distinctiveness is its approach to compute dominance as a result, which takes into account the user's choices (i.e., objectives optimization), finally producing the results in the form of dominance. Other algorithms prioritize absolute dominance with a diversity of solutions when ranking solutions in the objective space.

This obvious advantage of IBEA at higher dimensions of objective optimization sends an optimistic message to project managers and software, test engineers. Zitzler developed the IBEA algorithm with preferences criteria and incorporated it into a multi-objective search. IBEA, on the other hand, uses preference criteria more frequently, taking into account more of the user's optimization goals.

IBEA's author, Zitzler, created the algorithm so that "decision-maker preference knowledge" can be incorporated into multi-objective search [35]. Although evolutionary search algorithms like IBEA are commonly employed for optimization, they also help in searching optimal solutions from a large space of possible solutions, as in our case study.

Smart operators (like crossover and mutation) can be employed in the optimization process to promote optimal results. IBEA keeps track of all the valid solutions that the algorithm generates. The proposed approach computes the Pareto front of the optimal solutions. Results of this nature are obtained because of the parameter values, population size and the number of generations.

Our experimentation applied to a real case generates acceptable optimal results using IBEA. The proposed implementation of the objective function(s) played an important role to obtain valid solutions. We found it considerably easier to get appropriate solutions with IBEA when we defined the objective functions accurately. The results generated by IBEA prove the sufficiency of the proposed approach but there are chances to obtain different results if we modify the probability values of crossover and mutation operators.

4 Conclusion and Future Work

Choosing to invest resources in implementing required features or in paying off test debt is a hard job. This research offers an "indicator-based" strategy for assisting project managers and quality testers in resolving such a difficult task as dealing with multi-objective optimization. From the results obtained, it will be obvious for the quality testers to select the best compromise solutions based on project preferences. The initial experiments described the capability to explore different scenarios or situations, for instance, situations that may arise in future or coming sprints if test debt is not fixed. There would either be a need to pay these debts or to delay them. The current research takes into account the comprehensive evaluation of features and test debt items (test cases).

The most important finding of the proposed study is the distinct benefit of the IBEA search algorithm due to the way it utilizes user preferences or objectives. IBEA is being employed in software engineering search-based problems. It is proving successful results as we have shown. Consequently, the findings of this work should spur the discipline to investigate IBEA's performance in comparison to prior results in a variety of issues; tackle harder, more complex problems, and conduct further comparisons across MEOAs applied to software engineering problems [38,42]. The authors in [41,43] have also adopted the IBEA algorithm to generate optimal pareto front in compliance with the feature model configurations but our approach tries to generate pareto front to optimize the two objectives in case of technical debt optimization. The purpose of using an evolutionary algorithm (IBEA) to produce Pareto optimal solutions for feature coverage with test case minimization lies in its ability to employ the best tradeoff to achieve the optimization across the two objectives. The one threat of validity of our study is the selection of parameter values as mentioned in experiments setup Section 3.3.

Other possible future research directions include:

1-Exploring the scalability of IBEA's novel findings with more than two objectives optimization for large-scale agile projects.
2-Using IBEA and other MEOAs to explore the influence of various parameters adjustment.
3-Investigating the behavior of different quality indicators to generate more accurate Pareto fronts between MEOAs.

Acknowledgement: The authors are thankful to Umm Al Qura University, Makkah, Saudi Arabia, for supporting this research work.

Funding Statement: The authors would like to thank the Deanship of Scientific Research at Umm Al-Qura University for supporting this work by Grant Code: (22UQUyouracademicnumberDSRxx).

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References

[1] G. Samarthyam, M. Muralidharan and R. K. Anna, "Understanding test debt," in *Trends in Software Testing*, Singapore, Springer, pp. 1–17, 2017.
[2] T. Mikkonen and S. Kari. "Maximizing product value: Continuous maintenance," in *Int. Conf. on Product-Focused Software Process Improvement*, Springer, Cham, pp. 298–301, 2014.
[3] M. Leppanen, S. Makinen, S. Lahtinen, O. Sievi-Korte, A. -P. Tuovinen *et al.,* "Refactoring-a shot in the dark?," *IEEE Software*, vol. 32, no. 6, pp. 62–70, 2015.
[4] K. Wang, C. Zhu, A. Celik, J. Kim, D. Batory *et al.,* "Towards refactoring-aware regression test selection," in *2018 IEEE/ACM 40th Int. Conf. on Software Engineering (ICSE)*, Sweden, IEEE, pp. 233–244, 2018.
[5] K. Petersen and C. Wohlin, "A comparison of issues and advantages in agile and incremental development between state of the art and an industrial case," *Journal of Systems and Software*, vol. 82, no. 9, pp. 1479–1490, 2009.
[6] M. A. Jamil, M. Arif, N. S. A. Abubakar and A. Ahmad, "Software testing techniques: A literature review," in *2016 6th Int. Conf. on Information and Communication Technology for the Muslim World (ICT4 M)*, Indonesia, IEEE, pp. 177–182, 2016.
[7] M. A. Jamil, A. Alhindi, M. Arif, M. K. Nour, N. S. A. Abubakar *et al.,* "Towards software product lines optimization using evolutionary algorithms," *Procedia Computer Science*, vol. 163, pp. 527–537, 2019.
[8] M. Zbigniew, "Genetic algorithms + data structures = evolution programs," *Computational Statistics and Data Analysis, Elsevier*, vol. 24, no. 3, pp. 372–373, 1996.
[9] M. A. Jamil, "Maintenance of software product line using software testing optimization techniques," Ph.D. dissertation, International Islamic University Malaysia, 2020.
[10] K. Deb, "Evolutionary algorithms for multi-criterion optimization in engineering design," in *Evolutionary Algorithms in Engineering and Computer Science*, 2nd ed, UK, Springer, pp. 135–161, 1999.
[11] M. A. Jamil, A. Alhindi, M. Arif, M. K. Nour, N. S. A. Abubakar *et al.,* "Multiobjective evolutionary algorithms NSGA-II and NSGA-III for software product lines testing optimization," in *2019 IEEE 6th Int. Conf. on Engineering Technologies and Applied Sciences (ICETAS)*, Malaysia, IEEE, pp. 1–5, 2019.
[12] Y. Guo, R. Spnola and C. Seaman, "Exploring the costs of technical debt management a case study," *Empirical Software Engineering*, vol. 21, no. 1, pp. 1–24, 2014.
[13] N. Zazworka, C. Seaman and F. Shull, "Prioritizing design debt investment opportunities," in *Proc. of the 2nd Workshop on Managing Technical Debt*, USA, ACM, pp. 39–42, 2011.
[14] N. Brown, Y. Cai, Y. Guo, R. Kazman, M. Kim *et al.,* "Managing technical debt in software-reliant systems," in *Proc. of the FSE/SDP Workshop on Future of Software Engineering Research*, USA, ACM, pp. 47–52, 2010.
[15] T. T. Ho and G. Ruhe, "When-to-release decisions in consideration of technical debt," in *Proc. of the 6th Int. Workshop on Managing Technical Debt*, Canada, IEEE, pp. 31–34, 2014.
[16] E. Engström and P. Runeson, "Software product line testing–a systematic mapping study," *Information and Software Technology*, vol. 53, no. 1, pp. 2–13, 2011.
[17] M. K. Mehlawat, P. Gupta and D. Mahajan, "A Multi-period multi-objective optimization framework for software enhancement and component evaluation, selection and integration," *Information Sciences*, vol. 523, pp. 91–110, 2020.

[18] W. Cunningham, "The wycash portfolio management system," *ACM SIGPLAN OOPS Messenger*, vol. 4, no. 2, pp. 29–30, 1992.
[19] Z. Li, P. Avgeriou and P. Liang, "A systematic mapping study on technical debt and its management," *Journal of Systems and Software*, vol. 101, pp. 193–220, 2015.
[20] E. Tom, A. Aurum and R. Vidgen, "An exploration of technical debt," *Journal of Systems and Software*, vol. 86, no. 6, pp. 1498–1516, 2013.
[21] K. Miettinen, "Nonlinear multiobjective optimization," in *Operation Research and Management Science*, 1st ed, New York, Springer Science & Business Media, vol. 12, 1998.
[22] X. R. Zhang, W. F. Zhang, W. Sun, X. M. Sun and S. K. Jha, "A robust 3-D medical watermarking based on wavelet transform for data protection," *Computer Systems Science & Engineering*, vol. 41, no. 3, pp. 1043–1056, 2022.
[23] C. A. C. Coello, G. B. Lamont and D. A. Van Veldhuizen, "Evolutionary Algorithms for Solving Multi-Objective Problems," New York, USA: Springer, vol. 5, pp. 79–104, 2007.
[24] X. R. Zhang, X. Sun, X. M. Sun, W. Sun and S. K. Jha, "Robust reversible audio watermarking scheme for telemedicine and privacy protection," *Computers, Materials & Continua*, vol. 71, no. 2, pp. 3035–3050, 2022.
[25] W. Miller and D. L. Spooner, "Automatic generation of floating-point test data," *IEEE Transactions on Software Engineering*, vol. 2, no. 3, pp. 223–226, 1976.
[26] G. Fraser and A. Arcuri, "Evosuite: Automatic test suite generation for object-oriented software," in *Proc. of the 19th ACM SIGSOFT Symp. and the 13th European Conf. on Foundations of Software Engineering*, Hungary, pp. 416–419, 2019.
[27] A. Panichella, F. M. Kifetew and P. Tonella, "Reformulating branch coverage as a many-objective optimization problem," in *Int. Conf. on Software Testing, Verification and Validation (ICST)*, Austria, IEEE, pp. 1–10, 2015.
[28] A. Panichella, F. M. Kifetew and P. Tonella, "Automated test case generation as a many-objective optimisation problem with dynamic selection of the targets," *IEEE Transactions on Software Engineering*, vol. 44, no. 2, pp. 122–158, 2017.
[29] M. Khari and P. Kumar, "An extensive evaluation of searchbased software testing: A review," *Soft Computing*, vol. 23, no. 6, pp. 1933–1946, 2019.
[30] P. McMinn, "Search-based software testing: Past, present and future," in *2011 IEEE Fourth Int. Conf. on Software Testing, Verification and Validation Workshops*, Germany, IEEE, pp. 153–163, 2011.
[31] M. Harman, S. A. Mansouri and Y. Zhang, "Search-based software engineering: Trends, techniques and applications," *ACM Computing Surveys (CSUR)*, vol. 45, no. 1, pp. 1–61, 2012.
[32] S. Panichella, A. Panichella, M. Beller, A. Zaidman and H. C. Gall, "The impact of test case summaries on bug fixing performance: An empirical investigation," in *Proc. of the 38th Int. Conf. on Software Engineering, ICSE*, Texas, USA, pp. 547–558, 2016.
[33] S. Panichella, "Summarization techniques for code, change, testing, and user feedback (invited paper)," in *Workshop on Validation, Analysis and Evolution of Software Tests (VST)*, Italy, IEEE, pp. 1–5, 2018.
[34] C. M. Fonseca and P. J. Fleming, "Multiobjective optimization and multiple constraint handling with evolutionary algorithms. I. A unified formulation," *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, vol. 28, no. 1, pp. 26–37, 1998.
[35] E. Zitzler and S. Künzli, "Indicator-based selection in multiobjective search," in *Int. Conf. on Parallel Problem Solving from Nature*, Springer, Berlin, Heidelberg, pp. 832–842, 2004.
[36] J. D. Knowles, "Local-search and hybrid evolutionary algorithms for pareto optimization," *Ph.D. dissertation*, University of Reading, UK, 2002.
[37] J. Sutherland and K. Schwaber, "The scrum guide the definitive guide to scrum: The rules of the game," Available: Scrum.org, 2017.
[38] R. Alfayez and B. Boehm, "Technical debt prioritization: A search-based approach," in *2019 IEEE 19th Int. Conf. on Software Quality, Reliability and Security (QRS)*, Bulgaria, IEEE, pp. 434–445, 2019.

[39] A. Arcuri, D. R. White, J. Clark and X. Yao, "Multi-objective improvement of software using co-evolution and smart seeding," in *Asia-Pacific Conf. on Simulated Evolution and Learning*, Springer, pp. 61–70, 2011.
[40] A. Arcuri and L. Briand, "A practical guide for using statistical tests to assess randomized algorithms in software engineering," in *2011 33rd Int. Conf. on Software Engineering (ICSE)*, Honolulu, IEEE, pp. 1–10, 2011.
[41] A. S. Sayyad, T. Menzies and H. Ammar, "On the value of user preferences in search-based software engineering: A case study in software product lines," in *35Th Int. Conf. on Software Engineering (ICSE)*, USA, IEEE, pp. 492–501, 2013.
[42] S. H. Vathsavayi and K. Systä, "Technical debt management with genetic algorithms," in *2016 42th Euromicro Conf. on Software Engineering and Advanced Applications (SEAA)*, Cyprus, IEEE, pp. 50–53, 2016.
[43] P. Kouros, T. Chaikalis, E. M. Arvanitou, A. Chatzigeorgiou, A. Ampatzoglou and T. Amanatidis, "Jcaliper: Search-based technical debt management," in *Proc. of the 34th ACM/SIGAPP Symp. on Applied Computing*, Cyprus, pp. 1721–1730, 2019.