Computers, Materials & Continua
DOI:10.32604/cmc.2021.014686
Article

Tech Science Press

An Ontology Based Test Case Prioritization Approach in Regression Testing

Muhammad Hasnain[1], Seung Ryul Jeong[2,*], Muhammad Fermi Pasha[1] and Imran Ghani[3]

[1]Monash University, Petaling Jaya, 46150, Malaysia
[2]Kookmin University, Seoul, 136, Korea
[3]Indiana University of Pennsylvania, PA, 15705, USA
[*]Corresponding Author: Seung Ryul Jeong. Email: srjeong@kookmin.ac.kr
Received: 08 October 2020; Accepted: 26 November 2020

Abstract: Regression testing is a widely studied research area, with the aim of meeting the quality challenges of software systems. To achieve a software system of good quality, we face high consumption of resources during testing. To overcome this challenge, test case prioritization (TCP) as a sub-type of regression testing is continuously investigated to achieve the testing objectives. This study provides an insight into proposing the ontology-based TCP (OTCP) approach, aimed at reducing the consumption of resources for the quality improvement and maintenance of software systems. The proposed approach uses software metrics to examine the behavior of classes of software systems. It uses Binary Logistic Regression (BLR) and AdaBoostM1 classifiers to verify correct predictions of the faulty and non-faulty classes of software systems. Reference ontology is used to match the code metrics and class attributes. We investigated five Java programs for the evaluation of the proposed approach, which was used to achieve code metrics. This study has resulted in an average percentage of fault detected (APFD) value of 94.80%, which is higher when compared to other TCP approaches. In future works, large sized programs in different languages can be used to evaluate the scalability of the proposed OTCP approach.

Keywords: Software code metric; machine learning; faults detection; testing

1 Introduction

Regression testing involves retesting of software after adding changes in the software or its environment [1]. Retesting of entire software is quite a challenging task for testers. They usually take only the current test suit T with the maximum number of faults. Schwartz et al. in [2] examined cost-effective TCP approaches and suggested trade-offs among them. If test cases are significant in number, retesting of software becomes an expensive and time-consuming technique. Manual testing may take several weeks to months for retesting an entire system. Several regression testing approaches have been proposed in the literature to overcome the high consumption of resources, particularly TCP techniques [3,4].

TCP makes software evolution more rapid as well as cost-effective. Recent research has highlighted that TCP is widely used to manage the increased costs of execution of large test suits. Among the many TCP techniques is code coverage, which focuses on the code churn attribute to identify code parts that can be used to test and fix errors [5].

Ontology-based approaches can combine various software artifacts. They can also provide the description of design and testing of software and adhere to the full use of the semantic rules [6]. In [7], researchers proposed 'ontology-based knowledge map methodology' (PROM) to reduce the failure rate of 'business process re-engineering' (BPR) [8].

Ontologies present the formal conceptualization of different entities, their behavior, interfaces, and associations. They are widely used to meet automation and decision-making needs [9]. Prior studies have addressed the TCP aspects with regards to codes, requirements, risks and others. There is a lack of research exploring the identification of the defects of software systems using test case prioritization based on ontologies. The study in [10] used requirement ontology for test case generation during the software testing phase. The proposed approach in [11] is aimed to measure the objective risk management using web services analysis and ontology. Additionally, ontology-based literature has provided success in the classification of items in different research domains [12,13].

A study in [14] emphasized the ontology-driven software testing approach that covers functional and 'non-functional requirements' (NFRs). Ontology-based software testing can minimize ambiguity, heterogeneity, and incompleteness issues in properties and relationships. However, it is unknown how ontology can help in prioritizing the test properties of modified software systems. Prior to it, the study [15] proposed constructs in the ontology of NFRs, to provide a mechanism for tracing back their impacts on other constructs in an ontology. Moreover, ontologies can be updated or extended to support test activities, techniques, methods and software artifacts [16]. A recent work [17] sheds light on the role of ontologies in software systems' maintenance and testing. Ontologies are better than the existing approaches in detecting the critical scenarios in autonomous vehicle systems. Systems based on ontologies are modeled into inputs in combinatorial testing and used to generate test cases. A case study was carried out in [18] that demonstrated the applications of ontology-based 'test case generation' (TCG) in an industrial setting. This study proposes an ontology-based technique that generates different scenarios to reveal faults in a system. For continuous improvement of software systems, ontology, and provenance model [19] help capture and provide regression test data. The proposed ontology-driven studies [18,20] are relevant in identifying real-world system issues.

In this research, we integrate theoretical and practical aspects of ontologies with the testing of software systems.

The contributions of this paper are as follows:

- We propose an ontology-based TCP technique based on the defect prediction criterion.
- We propose to use the code metrics values to provide ground truth for prediction and classification of faulty and non-faulty classes.
- We propose usage of the classification models to classify the faulty and non-faulty classes and report the values of performance measuring metrics.
- We report empirical studies to evaluate the performance of the proposed TCP technique.

The rest of the paper is organized as follows. Section 2 covers a background of the main concepts used in the proposed study. Section 3 concerns the related works with the TCP approaches, software metrics, and reference ontology. The proposed ontology-based TCP technique is described

in Section 4. Empirical studies and experiments are presented in Section 5. Finally, Section 6 concludes the proposed research and presents implications for future works.

2 Background

This section covers the main concepts involved in this study, along with their definitions.

2.1 Regression Testing

Testing is a mandatory phase of software development. Software development companies face the challenges of testing for large scale and complex systems. Regression testing is one of these challenging testing activities, which consumes a lot of resources. It is performed to ensure that software modifications do not affect the other parts, and unchanged parts work as they did before any change or modification [21].

2.2 Test Case Prioritization (TCP)

Regression TCP aims to improve the fault detection rate, executing the important test cases as earlier as possible [22]. There are different proposed TCP techniques based on varying criteria. These are, code coverage and requirements coverage. TCP techniques are effective in reducing the overhead in regression testing. They are widely examined in the literature [23,24], with most studies focusing on code coverage criteria and prioritization algorithms. Institutively, code coverage criteria can be considered a characteristic of a test case, and studies use code coverage to guide the process of prioritizing test cases.

2.3 Ontology

Ontology is a rigorous and formal approach used for knowledge representation that provides precise and unambiguous semantics for terms [25]. In literature, ontologies are classified into top-domain, core, foundational, domain, or instance [14]. In software testing, ontologies provide the common shared knowledge understanding of concerned entities and their relationships, resulting in their easy organization, maintenance, and updates [26].

3 Related Works

In the following sections, we review the existing literature on TCP and related concepts, together with software metrics and reference ontology.

3.1 Test Case Prioritization Approaches

Test Case Prioritization is aimed at scheduling the execution of test cases following some criteria. It is performed to optimize some objective functions [23]. Yoo et al. [27] stated that TCP is also performed to maximize earlier fault detection in software systems. Srikanth et al. [28] stated that software issues mostly arise from modification or changes in software systems. Software maintenance, including software updates, modifications, and bug fixes make up more than 50% of the total cost of a project. Moreover, regression testing in any of its three types, 'test case selection,' 'test case reduction,' and 'test case prioritization' requires further processes to refine all these types. However, since the last several years, many researchers have focused on prioritizing test cases in a new fashion to detect the maximum faults. Parejo et al. [29] involved a highly configurable system known as 'Drupal framework' for their experiments using the multi-objective TCP approach. To test a more substantial and complex system is a challenging task, and therefore, multiple TCP criteria were proposed to identify the faults as early as possible.

Regarding the TCP approach proposed in [30], the faults' severity has been missed. Therefore, the researchers taking up future work need to offer criteria to prioritize test cases. They also need to allocate various scores according to the severity of each fault in software applications. To overcome the issue of TCP, Hao et al. [31] proposed the optimal coverage technique. Although several coverage-based TCP techniques have been proposed for fault detection and prioritization of test cases, researchers involve either structural coverage or detected faults to propose them.

3.2 Software Metrics in Predicting the Faulty Modules

The assessment of software quality attributes depends on the applications of appropriate metrics. However, the choice of software metrics becomes complicated as several of them are available in the literature. Arvanitou et al. [32] introduced a new property of software metrics called 'software metric fluctuation' (SMF), aimed at quantifying the variations in metrics score. Although the SMF has been aimed at measuring the variance in scores between successive software versions, two levels have been defined, which refer to the sensitivity (a high variation) and the stable (a low variation), respectively.

The role of SDP is to reduce the cost and ensure the quality of software. The success of SDP lies in using the software metrics that have been traditionally used; however, the class imbalance problem remains unresolved. To overcome this issue, Tong et al. [33] in a recently published research proposed the SDP named 'SDAEsTSE', which was focused on applying the software metrics to extract the defaults and address the imbalance issue of classes.

3.3 Reference Ontology

Souza et al. [34] used reference ontology for knowledge management with the objectives of structuring knowledge repositories and annotating the knowledge items to make the searching process much easier. The researchers proposed 'reference ontology on software testing' (ROoST) to address the issues related with testing process. They used a well-established, and well-known SABiO method. Several references were considered during the development of ROoST, as mentioned above. The proposed approach showed limitations in handling the software and human resources used in the previously mentioned study. This work [34] is related to our work, as ontologies cover test information. However, there is no proposal for the test case prioritization process in the earlier mentioned study.

In another research study, Serna et al. [35] emphasized establishing the association between testing and software metrics to software components. The value of metrics related to a software component was determined several times. However, the researchers in the lateral study used ontologies that were developed for software maintenance. Thus, the proposed technique in the latter study showed limitations regarding software maintenance and knowledge which can be managed, and which helps in maintaining the software products.

Saleh et al. [36] proposed a framework to classify the input pages depending on predefined domain classes. Web mining techniques and 'multi-layer domain ontology' were used to propose the framework. The researchers of the same research study claimed that their proposed framework outperformed the existing classification framework in terms of the recall, precision and accuracy metrics values. However, semantic relations to the ontology have been missed in this work, which can be carried out in future works.

In a recently published work, Lytvyn et al. [37] proposed the ontology-based method to classify text documents. They used metrics based on the classical structure of an ontology. Regarding the classical architecture of the ontology, two scalar values, such as the significance of concepts

and relationships, were added to the conventional three-elements-based ontology. An ontology with the three elements, such as set of concepts, relationships and interpretation of relationships, was already used. It showed inefficiency in calculating the distances.

4 Proposed Ontology-Based Test Case Prioritization (OTCP) Approach

This section describes the proposed OTCP approach, the process flow of which is depicted in Fig. 1.

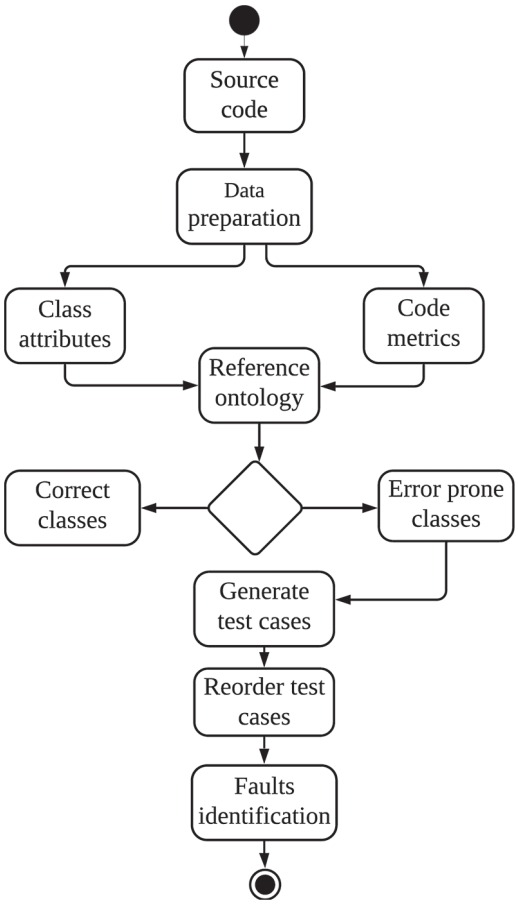

Figure 1: Process flow of proposed OTCP approach

Fig. 1 shows a high-level process flow of the proposed OTCP approach. The process starts by source code and data pre-processing for classification of Java classes. We have different phases of the proposed approach, as described in the following sections. Until the classification of the modules as faulty and non-faulty has reached, we continue the computation of metric values for modules of programs. We also process the attribute and metric values of classes in the proposed Reference Ontology. The description of the proposed Reference Ontology has been given in the following sections.

4.1 First Phase

The first phase involves the selection and data pre-processing of programs. In here, we perform a search for data repositories that host the desired programs used in this study. The datasets which have been frequently used in primary studies are selected from public data repositories. These repositories include PROMISE, GitHub, and ECLIPSE, and Maven. PROMISE is a source of datasets that are widely used in studies [33,38]. Other than the PROMISE data repository, Zhou et al. [39] explored the ECLIPSE repository to collect the datasets of three versions of Eclipse Java programs. In the current study, we access the programs from data repositories [40,41]. Datasets residing on these repositories are used for training and testing of the binary logistic regression and AdaBoostM1 classification models.

4.2 Second Phase

The next phase of the proposed approach as 'Metrics' is the core phase to select the potential code metrics. Appropriate code metrics can give us the valuable divergence of modules from their classification. We employ the code metrics, including lines of code, cyclomatic complexity, depth of inheritance, number of operands, and operators. Moreover, we present the reference ontology, which has been argued as a concept to classify the real-world issues [42]. We aim to use reference ontology by incorporating the code metrics at the class level of programs.

In the following Fig. 2, we present reference ontology at a class level to show how attributes of a class and software metrics match each other. Ontology can represent the consensus within a community by making the representation of fault identification in the projects [34]. It brings clarity in the description of classes and their code metrics, and how both have a relationship with each other.

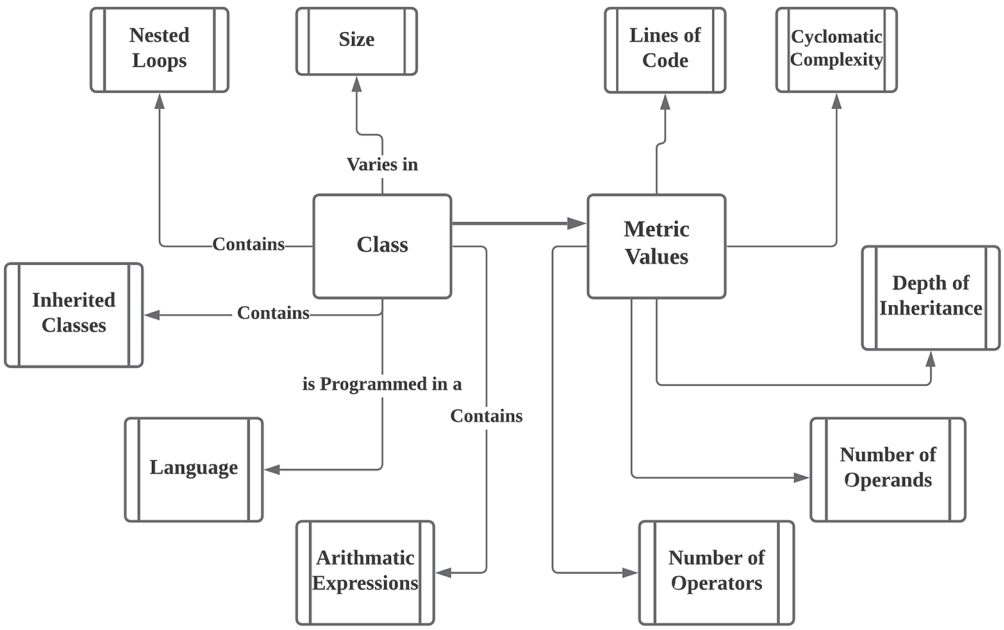

Figure 2: A class level reference ontology

Fig. 2 shows the high-level diagram of reference ontology as proposed in this study. Reference ontology provides an effective way to extract information. It also plays a vital role in bridging the gap of connecting metrics' values and their relevant classes. Before creating an ontology, experts' knowledge and literature support were used to define classes and their relationship with code attributes. There are various concepts across the metrics values. However, we keep the number of code metrics limited to achieve simplicity in results. We know that object-oriented programming is mostly based on the classes, and each class programmed in any language has many attributes, as shown in Fig. 2.

Before this study, references [43,44] used the threshold values of metrics such as LOCs, CC, number of operators, and number of operands. Therefore, we add metrics such as 'depth of inheritance hierarchy' (DIH) in the proposed OTCP approach. The lateral added metric represents the CK metric that was used in [45]. We use Plugin Metrics 1.3.8 to measure the values of each metric in this study.

Class complexity is generalized by using the DIH metric. A deeply positioned class makes sense, and is relatively complex maintain. The nominal range of the DIH metric's value lies between 0 and 4. The findings of Erdogmus et al. [46] provide evidence that a Java program with (DIH = 5) takes a longer time than one with an inheritance depth of 3. A flat program with 0 DIH does not require much effort to maintain. On the other hand, a Java program with a DIH of more than 4 becomes more complex and fault proneness is likely increased. Therefore, we have set the threshold value of DIH as 4, as given in this section. Below we have defined a few rules to reveal the threshold values of software metrics.

(1) If lines of code metric are >65, the class is faulty
(2) If Cyclomatic Complexity score is >10, the class is faulty
(3) If Depth of Inheritance score is >4, the class is faulty
(4) If number of operators is >125, the class is faulty
(5) If number of operands is >70, the class is faulty

We have adopted the threshold frequencies of code metrics, including lines of code, cyclomatic complexity, operators, and operands from the study of Abaei et al. [43], where they have provided a detailed description of software metrics. To show the binary relationship between class attributes and metric values, we have defined a rule as follows:

A class attribute (A) is in a binary relation to a metric value (V) in a one to one relationship. This means, A-to-V shows us a general overview of a relationship between two entities.

For example, we have five class attributes and five metrics values, so there is a one to one relationship between attributes and metrics values. We use an x-value and a y-value to show this relationship in a paired form. The set of x-values is called domain, while that of y-values is known as the range. To display the one-to-one relationship, reference ontology expresses this relationship between class attributes and metric values through the respective annotations (Fig. 3).

Fig. 3 shows the Visual Notation for OWL Ontologies (VOWL) view of the developed reference. It is clearly showing us the focused classes and sub-classes in the reference ontology. As mentioned earlier, the main focus has been on 'class' in the context of a programming language. As such, a class represents the classes of the application programmed in languages (Java, C++, C#, and VB.Net), and each class has its sub-classes, as shown in Fig. 3. Main classes, along with their relevant sub-classes, represent the knowledge domain that we want to model. Therefore, the resulting ontology may be used to make a statement concerning the metrics, classes and

decisions. Ontology serves as a means of knowledge representation and reasoning [47]. Generally, an ontology is represented by using OWL syntax, which makes it superior in representing complex relationships, information sharing, and reuse.

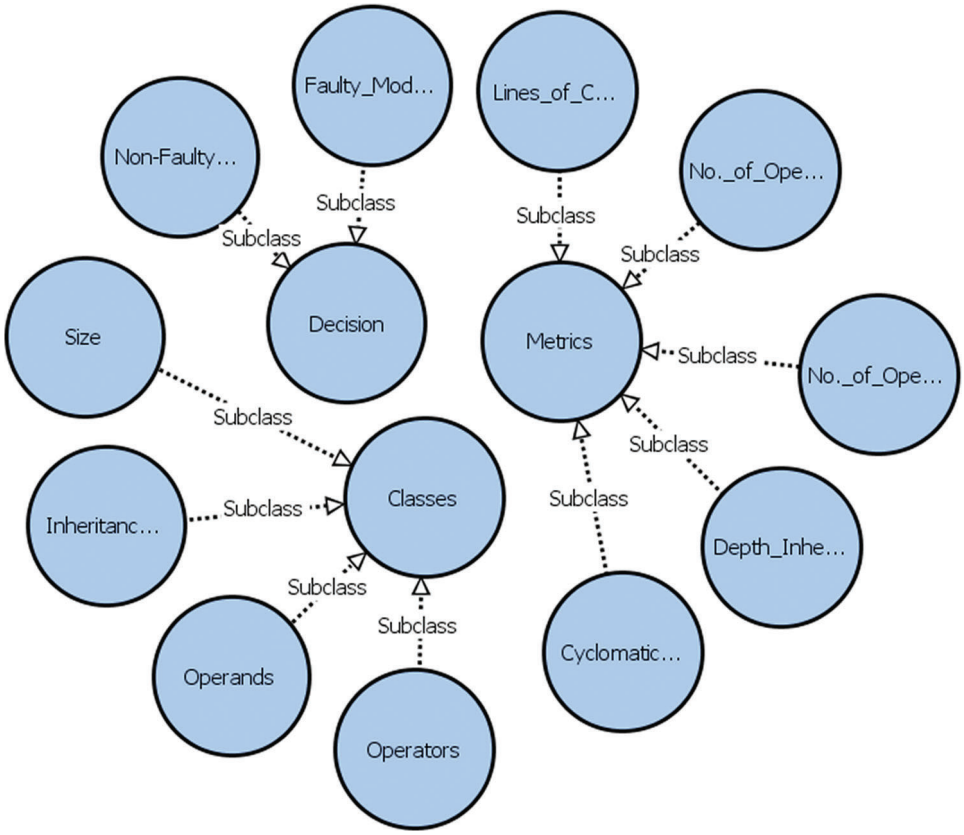

Figure 3: VOWL view of the reference ontology

Fig. 4 gives us an overview of an OntoGraph of the Ontologies which are used in this study. The OntoGraph supports the interactive navigation and relationship between OWL ontologies. We have imported the OWL for ontology parsing, which can be visualized by Protégé. Therefore, we have constructed a software program ontology in the OntoGraph in Protégé, as illustrated in Fig. 4. The class hierarchy is starting from the 'Thing' class to the main classes and sub-classes. The main classes include the metrics, classes, and decision. Each of these classes further incorporate a set of properties and conditions.

4.3 Third Phase

To validate the performance of the Metrics phase of the proposed OTCP approach, we aim to use the Binary Logistic Regression, and AdaBoostM1 classification models. Generally, logistic models have very close overall classification accuracy as compared to neural network models [48]. The main advantage of using logistic regression is its effectiveness in handling the multiple predictors to the mixed data types [49]. We aim to analyze the metrics data and validate the Metrics phase in step 1 to estimate the accuracy of the proposed OTCP approach in the same

phase. Fault prediction urges the researchers to assure that detection of faulty and non-faulty modules is supported by the statistical analysis.

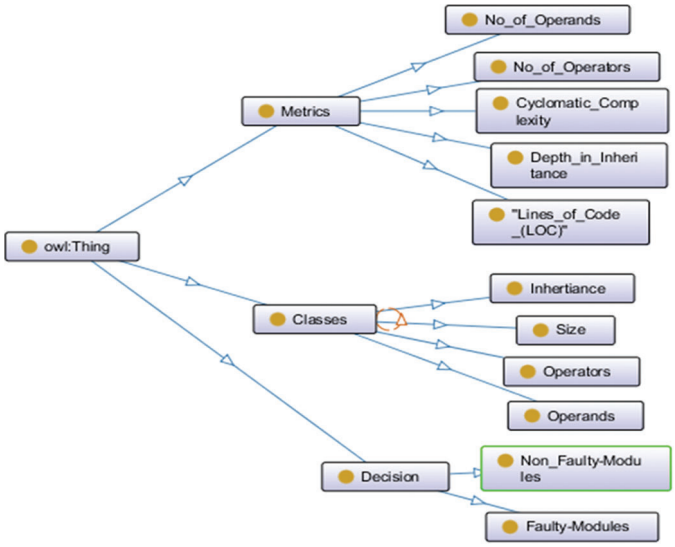

Figure 4: OntoGraph view of the ontology

We reorder the test case based on the fault score covered by each class in a program. However, a class may have multiple test cases, which need to be reordered based on higher number of faults. Previous studies emphasize the defect prediction methods which are leveraged for automated tasks, i.e., TCP. Most of the existing TCP techniques do not involve the latter mentioned aspect of software testing.

Based on the assumption, we extract the prior knowledge of fault probability from code metrics values. The probability of existing faults can be derived using the following formula given in Eq. (1), provided that the probability of the existing faults in a unit is as uj $1 \leq j \leq m$ and is Prob. of F_j; where F_j indicates the event in which j as a coding unit is faulty, and Prob. (F_j) represents the probability of this event [50]. We can determine the coverage for the test case Ti from the modified coverage formula, as shown in Eq. (1).

$$FaultsBasedCover\,(i) = \sum_{1 \leq i \leq m}^{\infty} (Cover\,(i,j)\,xProb.(F_j) \tag{1}$$

Based on the above formula, a unit test case with a higher weight has more fault probability. As such, more priority is given to that unit in a class. A class may have multiple test cases, and each test with a higher fault probability is prioritized for the execution.

5 Empirical Study and Experiments

This section presents the empirical studies conducted to answer the research. Experiments were carried out on a PC with an Intel Core processor and 8 GB of RAM. Before we discuss experiments, we propose the research question as follows:

RQ: Is the proposed OTCP approach superior to the existing TCP techniques?

The RQ is aimed to measure the efficiency of the proposed TCP approach against other TCP approaches. To compare the performance of our proposed approach, we have five different criteria based on TCP techniques. In addition to these TCP approaches, we also aim to compare the OTCP approach's performance with other approaches if any of the five Java programs have been analyzed in other TCP approaches. Based on the APFD value, we present each technique's fault detection competency, because the APFD metric is widely explored in the research area of regression testing. Some researchers [51,52] have doubts about the effectiveness of mutant based TCP approaches. Therefore, we collect datasets with the number of test cases and detected faults. Experiments on the actual Java programs are convincing for answering the proposed RQ.

5.1 Implementation and Subject Programs

Eclipse SDK 3.7.2. Framework [53] is used to implement experiments. This framework supports several plugins, including CodePro Analytix, which is mainly used by Java developers. Eclipse supports the dynamic uploading of Java projects, which are further explored in Java packages and classes. Next, we perform the categorization of classes of Java programs. Since we collect information at the class level, we perform experiments at the class level test cases. The section below describes the five chosen Java programs (datasets).

'Measure of Software Similarity' (Moss) is an online service freely available for academic use. It detects the similarity between a pair of documents. Several studies highlight its use for experiments and empirical evaluation of code-based approaches [54,55]. This dataset resides on GitHub repository [56]. The second dataset, PureMVC, is a lightweight framework used to create applications based on model-view-controller design and meta-patterns. The source code of the PureMVC program resides on the repositories [41,57]. This program helps in setting the best practices and architecture for software development [58]. Ant program and several other programs have been used to evaluate the proposed code of the coverage-based TCP approach [59]. Jsystem and Tomcat programs, with their different versions, can be accessed from the Maven repository [41]. Tomcat and Ant datasets have been used for the evaluation of ensemble learning classifiers [60]. All these programs were used to validate the proposed ontology.

5.2 Empirical Study Design

We exploit the information regarding code metric values of Java programs' classes, test cases, faults, prediction and classification of faulty classes, and performance accuracies of classification models. Execution results of the earlier mentioned factors are obtained by using CodePro Analytix and Weka 3.8.4. We use JUnit testing [61] to design test cases, which also gives us information regarding the faults identified for each class's test cases. As we assumed that the earliest Java programs did not have any errors, and faulty versions were produced by modifications, and introduced commits. Thus, the faulty versions contained the metrics values, as explained in the second phase of the proposed OTCP approach.

Next, we have analyzed code metric results using the defined rules to classify the faulty and non-faulty classes of five Java programs. Tab. 1 presents a summary of actual and predicted faulty and non-faulty classes in focused programs, and the accuracy score of both the classifiers. For the Moss program, we have determined that out of 10 classes, 5 java classes were identified as faulty classes. However, we validate our findings by performing binary logistic regression analysis. We also use the AdaBoostM1 classifier to verify the accuracy in the classification and prediction of faulty classes and non-faulty classes. Below we show the correct classification and prediction results of five Java programs.

Table 1: Prediction results of faulty and non-faulty classes in five Java programs

Program name	Classifier	Actual faulty classes	Actual non-faulty classes	Predicted faulty classes	Predicted non-faulty classes	Accuracy (% age)
Moss	BLR	7	3	6	1	70.00
	AdaBoostM1	7	3	6	3	90.00
PureMVC	BLR	6	19	4	19	92.00
	AdaBoostM1	6	19	5	19	96.00
Jsystem	BLR	10	7	7	7	82.35
	AdaBoostM1	10	7	7	7	82.35
Tomcat	BLR	8	2	8	2	99.80
	AdaBoostM1	8	2	8	0	80.00
Ant	BLR	7	8	6	8	93.33
	AdaBoostM1	6	9	4	8	80.00

Results illustrated in Tab. 1 verify that six Java classes are correctly observed as non-faulty classes from the BLR classifier. In contrast, the AdaBoostM1 classifier has correctly predicted the same number of classes for the Moss program. This also shows us that the prediction made by using BLR classifier is according to the identified Java classes. We also performed a binary classification of the PureMVC Java program using the chosen classifiers. Out of 25 Java classes in the PureMVC program, four classes were identified as faulty classes using the BLR classifier, while five classes were identified as non-faulty by the AdaBoostM1 classifier. For the remaining three Java programs, Jsystem, Tomcat, and Ant, we have better classification accuracies from two classifiers. Thus, the binary classification with the support of two classifiers verified what we observed the correct prediction of faulty and non-faulty classes in five Java programs. Prediction accuracy of both types of classes varied between 70% and 99.80%, leading to better testing in further steps. We have generated test cases of the faulty Java classes of datasets of five programs.

Table 2: APFD values of five programs

Program name	Number of classes	Number of test cases	Number of faults	APFD value
Moss	5	16	20	0.8877
PureMVC	25	138	75	0.9684
Jsystem	17	310	103	0.9494
Tomcat	18	165	215	0.9533
Ant	15	91	76	0.9814

Tab. 2 is showing us the programs, number of classes, number of test cases, faults, and APFD values of the five focused datasets. These results illustrate that PureMVC is an extensive size program with a more significant number of classes, while Moss is with the minimum number of classes. The other three programs such as Jsystem, Tomcat, and Ant are with medium number of classes. Our proposed OTCP approach is best in revealing faults in small and medium-sized software programs. Programs such as Moss, PureMVC, and Tomcat have 2466, 2481, and 1460 LOCs, respectively. We assume these programs as being small sized programs, since the

LOCs are less than 5000. On the other hand, Ant with 7232 and Jsystem with 8983 LOCs are considered medium-sized, because their size in LOCs is more than 5000 and less than 10000 LOCs. Faults were not revealed according to the size of a program. Future works may investigate how a program size affects predicting the number of faults. Maybe the versions used in these programs were not stable enough in revealing the faults with their program size. In the following figure, we show the efficiency of the proposed OTCP approach with respect to APFD values.

Based on the APFD values as listed in Tab. 2 and shown in Fig. 5a, the fault detection capability of the OTCP approach is excellent for most of the programs. Among the Java programs, the chosen Ant program is also used for the evaluation of the "Additional Greedy-method Call" (AGC) sequence [62] approach that received 82.59% as APFD value in the latter mentioned study. In contrast, our study received an APED value of 98.14% in the same subject program. It clearly shows that the OTCP approach is better in fault detection than the other TCP approaches. This finding directly answers the RQ proposed in this study, as Ant program is widely used in other studies.

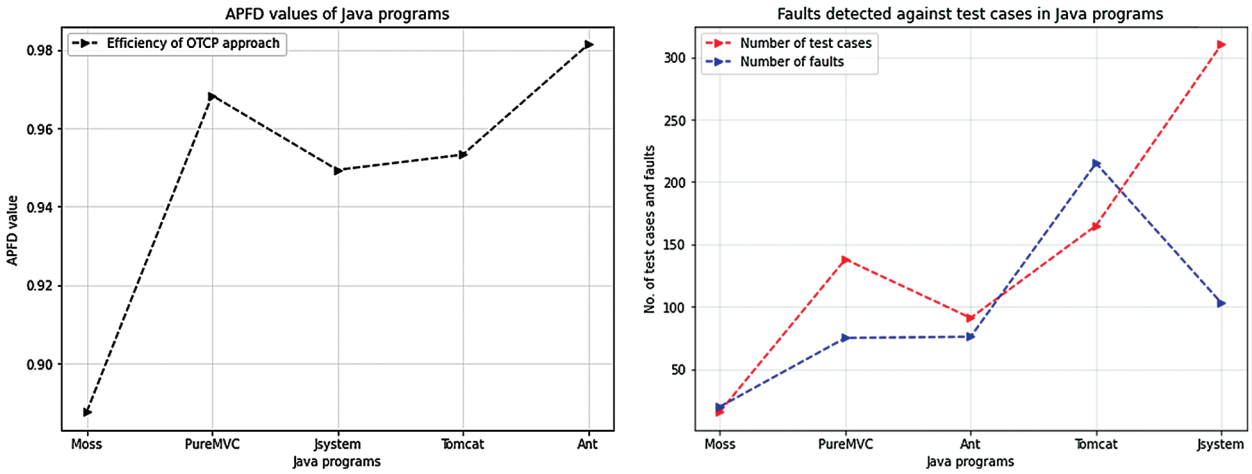

Figure 5: APFD values from five Java programs based on the number of test cases and detected faults in them. The line in the plot (a) represents the efficiency of the OTCP approach from the obtained APFD values. In the plot (b), red and blue lines represent the number of test cases and detected faults in the five Java programs

Binary logistic regression (BLR) is a model used for the prediction of faulty classes in Java programs. We apply the AdaBoostM1 classifier to compare the performance of the BLR model. AdaBoostM1 is a typical model of the boosting family, and it is trained on the datasets to sequentially generate base classifiers [63]. During the iterations, incorrect identifications in the previous iteration are emphasized to introduce a new classifier. In this research, the dependent variable is 0 and 1, where 0 represents non-faulty classes, and 1, the faulty classes [64].

5.3 Performance of the OTCP Approach

This section contains the experiment data used to answer the RQ, and justifies how the proposed OTCP approach is efficient in regression testing.

For the performance comparison of TCP approaches, the APFD metric is widely used to measure how quickly the faults are identified in each test suite [65]. The APFD metric value is always between 0 and 1 or between 0% and 100%. A TCP technique with a value closer to 100% is called a very efficient approach [66]. Therefore, we used the APFD metric, which is defined in the following equation:

$$APFD = 1 - \frac{(TF1 + TF2 + \ldots)}{nm} + \frac{1}{2n} \tag{2}$$

where n signifies the total number of test cases, and m is the number of faults.

We have compared the efficiency of the proposed OTCP approach with several other TCP approaches, including 'prioritization of requirements for testing)' (PORT), risk-based, optimal-coverage, scope aided, and conventional code-based techniques, as shown in Tab. 3. The average APFD value in a coverage-based study [67] remained 0.74, which is less as compared to 0.9480 or (94.80%), which is an average APFD value of five programs studied in this research. PORT and Risk-based techniques are requirement-based TCP techniques [66]. A higher APFD value indicates that a maximum number of faults have been detected. An increased APFD value has been observed in our study compared to the rest of the studies. For example, Miranda et al. [68] reported the overall APFD values of 93.29% and 93.46% for scope-aided and traditional TCP techniques, respectively. Therefore, the OTCP approach has an edge over the reported APFD values in the lateral discussed studies.

Table 3: APFD values of TCP techniques

Technique	OTCP (this study)	PORT	Risk based	Optimal coverage	Scope aided	Traditional technique
APFD value	94.80	80.00	77.00	74.00	93.29	93.46

5.4 10-Fold Cross-Validation Method

We trained BLR and AdaBoostM1 classification models using code metrics values of five Java programs. Furthermore, we performed a 10-Fold cross-validation method to evaluate the chosen classification models. Data was split into training and testing sets [67]. Data is divided into ten equal subsets. Each classification model has been trained using 90% of the data, while the remaining 10% of the data is used for testing. We record the average accuracy for ten iterations as a final measurement. The performance measures, including Precision, Recall, and F-Measure (see Tab. 4) to evaluate the performance of classification models, have been reported. Cross-validation is aimed to minimize the risks of the internal validity of results.

As shown in Tab. 4, Precision, Recall, and F-Measures are taken as an average of the measures for faulty (F) and non-faulty (NF) classes of programs. The minimum Precision, Recall, and F-measure values are 0.675, 0.700, and 0.680, respectively. These values were measured for the evaluation of the BLR model using the Moss program. For the remaining programs,

Precision, Recall, and F-measure values are higher, which indicates that the predicted classes of five programs are correctly classified.

The chosen classifiers perform very well on the five datasets of Java programs. Performance indicators are based on the confusion matrix of each dataset. A confusion matrix gives information on how a certain behavior is correctly detected, and how it is classified as another behavior [69,70]. Evaluation of the classifiers' performance implicitly describes the efficiency of OTCP approach, because correct classification of classes and their execution for TCP is essentially focused on in this study.

Table 4: Performance evaluation results

Program	Model	Precision	Recall	F-Measure
Moss	BLR	0.675	0.700	0.680
	AdaBoostM1	0.925	0.900	0.903
PureMVC	BLR	0.928	0.920	0.914
	AdaBoostM1	0.962	0.960	0.959
Ant	BLR	0.941	0.933	0.933
	AdaBoostM1	0.800	0.800	0.796
Tomcat	BLR	0.998	0.995	0.990
	AdaBoostM1	0.800	0.795	0.770
Jsystem	BLR	0.876	0.824	0.820
	AdaBoostM1	0.870	0.810	0.800

Additionally, the performance measuring metric results presented show how the proposed OTCP approach supports the classification of faulty and non-faulty classes. As listed in Tab. 4 (Precision, Recall and F-measure), accuracy measures evidence the correct prediction of faulty classes. AdaBoostM1 classifier shows better classification accuracy for Moss and PureMVC programs than the BLR model. For the other three programs (Jsystem, Tomcat, and Ant), the BLR model shows better accuracy results than the AdaBoostM1 model. Overall, both classifiers show higher accuracies in the categorization of classes. However, in medium-sized programs such as PureMVC and Ant with more than 5000 LOCs, the chosen classifiers' performance is very well. This answers the RQ that the chosen classifiers showed a better classification, and prediction of classes in Java programs.

5.5 Findings

The APFD value of our proposed OTCP approach was higher in comparison to code coverage and requirement based TCP approaches. The significant variance between code-based and requirement-based approaches shows the importance of code metrics used in this study. We revealed useful information about code metrics such as LOCs, CC, DIH, number of operators, and the number of operands metrics, which helped us classify the fault-prone classes in Java programs. Our obtained results suggest that the proposed OTCP approach achieved consistently excellent results if class attributes were mapped to software metrics using the standard ontology. Considering the class attributes and assigning them appropriate software metrics provides us support for a better classification of faulty and non-faulty classes of software systems. It was observed that code complexity and DIH were useful in gauging the capability or strength of a Java class in terms of analyzing the code. A class that is more complicated in its code structure

may result in more logical errors in programs. Therefore, we used CC and DIH metrics to reveal the earlier faults in Java programs.

On the other hand, operators and operands metrics were used to study the black box behavior of Java programs that helped us in revealing the possible errors in Java programs. The APFD value is better than the APFD values of other TCP approaches. Our proposed OTCP approach's increased performance is due to the ontology facts used in this study. Reference ontology is used indirectly, mapping the class features and their pertaining software metrics values.

6 Conclusions and Future Works

This paper provides a comprehensive insight into an ontology-based TCP approach for prioritizing test cases using Java programs. The proposed approach involves reference ontology in mapping the class attributes to software metrics. Classification of Java classes into faulty and non-faulty modules is validated using the BLR and AdaBoostM1 classification models. Faulty Java classes are further analyzed for JUnit testing. The JUnit test results reveal a high number of faults in the prioritized test cases. Our proposed OTCP approach shows better results than the state-of-the-art TCP approaches based on APFD metric values.

In future work, we plan to extend the proposed OTCP approach for faults identification of large-sized programs using web services, which have become a hot spot in the research area of regression testing. It will be very interesting to know how performance of a large web service can be affected when new web services are integrated, or existing services are updated. Our proposed approach will apply the code metrics criteria to identify the faulty services, and prioritize the test cases of faulty services to fix the errors.

Funding Statement: The authors received no specific funding for this study.

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

References

[1] W. Zheng, R. M. Hierons, M. Li, X. Liu and V. Vinciotti, "Multi-objective optimization for regression testing," *Information Sciences*, vol. 334, pp. 1–16, 2016.

[2] A. Schwartz and H. Do, "Cost-effective regression testing through adaptive test prioritization strategies," *Journal of Systems and Software*, vol. 115, pp. 61–81, 2016.

[3] J. A. P. Lima and S. R. Vergilio, "Test case prioritization in continuous integration environments: A systematic mapping study," *Information and Software Technology*, vol. 121, pp. 1–16, 2020.

[4] A. Arrieta, S. Wang, G. Sagardui and L. Etxeberria, "Search-based test case prioritization for simulation-based testing of cyber-physical system product lines," *Journal of Systems and Software*, vol. 149, pp. 1–34, 2019.

[5] J. Anderson, M. Azizi, S. Salem and H. Do, "On the use of usage patterns from telemetry data for test case prioritization," *Information and Software Technology*, vol. 113, pp. 110–130, 2019.

[6] M. M. Ali, M. B. Doumbouya, T. Louge, R. Rai and M. H. Karray, "Ontology-based approach to extract product's design features from online customers' reviews," *Computers in Industry*, vol. 116, pp. 1–20, 2020.

[7] M. Abdellatif, M. S. Farhan and N. S. Shehata, "Overcoming business process reengineering obstacles using ontology-based knowledge map methodology," *Future Computing and Informatics Journal*, vol. 3, no. 1, pp. 7–28, 2018.

[8] M. McDaniel, V. C. Storey and V. Sugumaran, "Assessing the quality of domain ontologies: Metrics and an automated ranking system," *Data & Knowledge Engineering*, vol. 115, pp. 32–47, 2018.

[9] Y. Li, J. Tao and F. Wotawa, "Ontology-based test generation for automated and autonomous driving functions," *Information and Software Technology*, vol. 117, pp. 1–16, 2020.

[10] S. Ul Haq and U. Qamar, "Ontology based test case generation for black box testing," in *Proc. of the 2019 8th Int. Conf. on Educational and Information Technology*, Cambridge, UK, pp. 236–241, 2019.

[11] X. Bai, R. S. Kenett and W. Yu, "Risk assessment and adaptive group testing of semantic web services," *International Journal of Software Engineering and Knowledge Engineering*, vol. 22, no. 5, pp. 595–620, 2012.

[12] X. Xue and J. Lu, "A compact brain storm algorithm for matching ontologies," *IEEE Access*, vol. 8, pp. 43898–43907, 2020.

[13] S. El-Sappagh, F. Franda, F. Ali and K. S. Kwak, "SNOMED CT standard ontology based on the ontology for general medical science," *BMC Medical Informatics and Decision Making*, vol. 18, no. 76, pp. 1–19, 2018.

[14] G. Tebes, D. Peppino, P. Becker, G. Matturro, M. Solari *et al.,* "Analyzing and documenting the systematic review results of software testing ontologies," *Information and Software Technology*, vol. 123, pp. 1–23, 2020.

[15] M. Kassab, O. Ormandjieva and M. Daneva, "Relational-model based change management for non-functional requirements: Approach and experiment," in *Fifth Int. Conf. on Research Challenges in Information Science*, Gosier, France, pp. 1–9, 2011.

[16] H. Zhu and Y. Zhang, "A test automation framework for collaborative testing of web service dynamic compositions," in *Advanced Web Services*. New York, USA: Springer, pp.171–197, 2014.

[17] S. Popereshnyak and A. Vecherkovskaya, "Modeling ontologies in software testing," in *IEEE 14th Int. Conf. on Computer Sciences and Information Technologies*, Lviv, Ukraine, pp. 236–239, 2019.

[18] C. F. Fan and W. S. Wang, "Validation test case generation based on safety analysis ontology," *Annals of Nuclear Energy*, vol. 45, pp. 46–58, 2012.

[19] H. de Souza Campos Jr, C. A. de-Paiva, R. Braga, M. A. P. Araújo, J. M. N. David *et al.,* "Regression tests provenance data in the continuous software engineering context," in *Proc. of the 2nd Brazilian Sym. on Systematic and Automated Software Testing*, Fortaleza, Brazil, pp. 1–6, 2017.

[20] É. F. de-Souza, R. A. Falbo and N. L. Vijaykumar, "Knowledge management initiatives in software testing: A mapping study," *Information and Software Technology*, vol. 57, pp. 378–391, 2015.

[21] N. M. Minhas, K. Petersen, J. Börstler and K. Wnuk, "Regression testing for large-scale embedded software development-exploring the state of practice," *Information and Software Technology*, vol. 120, pp. 1–15, 2020.

[22] 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, pp. 1–21, 2020.

[23] G. Rothermel, R. H. Untch, C. Chu and M. J. Harrold, "Test case prioritization: An empirical study," in *Proc. IEEE Int. Conf. on Software Maintenance*, Oxford, UK, pp. 179–188, 1999.

[24] B. Jiang, Z. Zhang, W. K. Chan and T. Tse, "Adaptive random test case prioritization," in *2009 IEEE/ACM Int. Conf. on Automated Software Engineering*, Washington, USA, pp. 233–244, 2009.

[25] J. Lin, M. S. Fox and T. Bilgic, "A requirement ontology for engineering design," *Concurrent Engineering*, vol. 4, no. 3, pp. 279–291, 1996.

[26] H. Naseer and A. Rauf, "Validation of ontology based test case generation for graphical user interface," in *2012 15th Int. Multitopic Conf.*, Islamabad, Pakistan, pp. 465–469, 2012.

[27] S. Yoo and M. Harman, "Regression testing minimization, selection and prioritization: A survey," *Software Testing, Verification and Reliability*, vol. 22, no. 2, pp. 67–120, 2012.

[28] H. Srikanth, M. Cashman and M. B. Cohen, "Test case prioritization of build acceptance tests for an enterprise cloud application: An industrial case study," *Journal of Systems and Software*, vol. 119, pp. 122–135, 2016.

[29] J. A. Parejo, A. B. Sánchez, S. Segura, A. Ruiz-Cortés, R. E. Lopez-Herrejon *et al.,* "Multi-objective test case prioritization in highly configurable systems: A case study," *Journal of Systems and Software*, vol. 122, pp. 287–310, 2016.

[30] H. Srikanth, C. Hettiarachchi and H. Do, "Requirements based test prioritization using risk factors: An industrial study," *Information and Software Technology*, vol. 69, pp. 71–83, 2016.

[31] D. Hao, L. Zhang, L. Zang, Y. Wang, X. Wu *et al.,* "To be optimal or not in test-case prioritization," *IEEE Transactions on Software Engineering*, vol. 42, no. 5, pp. 490–505, 2016.

[32] E. M. Arvanitou, A. Ampatzoglou, A. Chatzigeorgiou and P. Avgeriou, "Software metrics fluctuation: A property for assisting the metric selection process," *Information and Software Technology*, vol. 72, pp. 110–124, 2016.

[33] H. Tong, B. Liu and S. Wang, "Software defect prediction using stacked de-noising auto-encoders and two-stage ensemble learning," *Information and Software Technology*, vol. 96, pp. 94–111, 2018.

[34] E. Souza, R. Falbo and N. L. Vijaykumar, "Using ontology patterns for building a reference software testing ontology," in *17th IEEE Int. Enterprise Distributed Object Computing Conf. Workshops*, Vancouver, BC, Canada, pp. 21–30, 2013.

[35] E. Serna and A. Serna, "Ontology for knowledge management in software maintenance," *International Journal of Information Management*, vol. 34, no. 5, pp. 704–710, 2014.

[36] A. I. Saleh, M. F. Al Rahmawy and A. E. Abulwafa, "A semantic based web page classification strategy using multi-layered domain ontology," *World Wide Web*, vol. 20, no. 5, pp. 939–993, 2017.

[37] V. Lytvyn, V. Vysotska, O. Veres, I. Rishnyak and H. Rishnyak, "Classification methods of text documents using ontology based approach," in *Advances in Intelligent Systems and Computing*, Warsaw, Poland: Polish Academy of Sciences, pp. 229–240, 2017.

[38] S. S. Rathore and S. Kumar, "An empirical study of some software fault prediction approaches for the number of faults prediction," *Soft Computing*, vol. 21, no. 24, pp. 7417–7434, 2017.

[39] Y. Zhou, B. Xu and H. Leung, "On the ability of complexity metrics to predict fault-prone classes in object-oriented systems," *Journal of Systems and Software*, vol. 83, no. 4, pp. 660–674, 2010.

[40] M. M. Öztürk, "A bat-inspired algorithm for prioritizing test cases," *Vietnam Journal of Computer Science*, vol. 5, no. 1, pp. 45–57, 2018.

[41] Maven Repository, What's new in Maven. [Online]. Available: https://mvnrepository.com/.

[42] C. Donalds and K. M. Osei-Bryson, "Toward a cybercrime classification ontology: A knowledge-based approach," *Computers in Human Behavior*, vol. 92, pp. 403–418, 2019.

[43] G. Abaei, A. Selamat and H. Fujita, "An empirical study based on semi-supervised hybrid self-organizing map for software fault prediction," *Knowledge-Based Systems*, vol. 74, pp. 28–39, 2015.

[44] E. Erturk and E. A. Sezer, "Iterative software fault prediction with a hybrid approach," *Applied Soft Computing*, vol. 49, pp. 1020–1033, 2016.

[45] D. P. P. Mesquita, L. S. Rocha, J. P. P. Gomes and A. R. Neto, "Classification with reject option for software defect prediction," *Applied Soft Computing*, vol. 49, pp. 1085–1093, 2016.

[46] H. Erdogmus and O. Tanir, *Advances in Software Engineering: Comprehension, Evaluation, and Evolution*, 1st ed. Germany: Springer Science & Business Media, 2013.

[47] M. M. Mkhinini, O. Labbani-Narsis and C. Nicolle, "Combining uml and ontology: An exploratory survey," *Computer Science Review*, vol. 35, pp. 1–14, 2020.

[48] J. A. Vallejos and S. D. McKinnon, "Logistic regression and neural network classification of seismic records," *International Journal of Rock Mechanics and Mining Sciences*, vol. 62, pp. 86–95, 2013.

[49] A. R. Gilal, J. Jaafar, L. F. Capretz, M. Omar, S. Basri *et al.,* "Finding an effective classification approach to develop a software team composition model," *Journal of Software: Evolution and Process*, vol. 30, no. 1, e1920, 2018.

[50] 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 Software Technology*, vol. 121, pp. 1–12, 2020.

[51] J. H. Andrews, L. C. Briand and Y. Labiche, "Is mutation an appropriate tool for testing experiments?," in *Proc. of the 27th Int. Conf. on Software Engineering*, St. Louis, USA, pp. 402–411, 2005.

[52] 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, pp. 1–23, 2019.

[53] Eclipse SDK 3.7.2, 2012. [Online]. Available: http://archive.eclipse.org/eclipse/downloads/drops/R-3.7.2-201202080800/.

[54] S. Schleimer, D. S. Wilkerson and A. Aiken, "Winnowing: Local algorithms for document fingerprinting," in *Proc. of the 2003 ACM SIGMOD Int. Conf. on Management of Data*, San Diego California, USA, pp. 76–85, 2003.

[55] W. T. Cheung, S. Ryu and S. Kim, "Development nature matters: An empirical study of code clones in javascript applications," *Empirical Software Engineering*, vol. 21, no. 2, pp. 517–564, 2016.

[56] PureMVC Java MultiCore Framework. [Online]. Available: https://github.com/PureMVC/puremvc-java-multicore-framework/wiki.

[57] Moss Program. [Online]. Available: https://github.com/genchang1234/How-to-cheat-in-computer-science-101/blob/master/README.md.

[58] M. David, *Building Games with Flash for the Mobile Market*, NY, USA: Taylor & Francis, pp. 115–153, 2011.

[59] A. Boucher and M. Badri, "Software metrics thresholds calculation approaches to predict fault-proneness: An empirical comparison," *Information and Software Technology*, vol. 96, pp. 38–67, 2018.

[60] S. Moustafa, M. Y. ElNainay, N. El Makky and M. S. Abougabal, "Software bug prediction using weighted majority voting techniques," *Alexandria Engineering Journal*, vol. 57, no. 4, pp. 2763–2774, 2018.

[61] M. Bruntink and A. van Deursen, "An empirical study into class testability," *Journal of Systems and Software*, vol. 79, no. 9, pp. 1219–1232, 2006.

[62] J. Chi, Y. Qu, Q. Zheng, Z. Yang, W. Jin *et al.,* "Relation-based test case prioritization for regression testing," *Journal of Systems and Software*, vol. 163, pp. 1–18, 2020.

[63] E. R. Q. Fernandes and A. C. P. L. F. de Carvalho, "Evolutionary inversion of class distribution in overlapping areas for multi-class imbalanced learning," *Information Sciences*, vol. 494, pp. 141–154, 2019.

[64] S. Saha, M. Saha, K. Mukherjee, A. Arabameri, P. T. T. Ngo *et al.,* "Predicting the deforestation probability using the binary logistic regression, random forest, ensemble rotational forest and REPTree: A case study at the Gumani river basin," *India Science of the Total Environment*, vol. 730, pp. 1–20, 2020.

[65] M. Hasnain, I. Ghani, M. F. Pasha, C. H. Lim and S. R. Jeong, "A comprehensive review on regression test case prioritization techniques for web services," *KSII Transactions on Internet and Information Systems*, vol. 14, no. 5, pp. 1861–1885, 2020.

[66] C. Hettiarachchi, H. Do and B. Choi, "Risk-based test case prioritization using a fuzzy expert system," *Information and Software Technology*, vol. 69, pp. 1–15, 2016.

[67] G. Rothermel, R. H. Untch, C. Chu and M. J. Harrold, "Prioritizing test cases for regression testing," *IEEE Transactions on Software Engineering*, vol. 27, no. 10, pp. 929–948, 2001.

[68] B. Miranda and A. Bertolino, "Scope-aided test prioritization, selection and minimization for software reuse," *Journal of Systems and Software*, vol. 131, pp. 528–549, 2017.

[69] D. Hao, L. Zhang, L. Zang, Y. Wang, X. Wu *et al.,* "To be optimal or not in test-case prioritization," *IEEE Transactions on Software Engineering*, vol. 42, no. 5, pp. 490–505, 2015.

[70] F. Zhang, J. Bai, X. Li, C. Pei and V. Havyarimana, "An ensemble cascading extremely randomized trees framework for short-term traffic flow prediction," *KSII Transactions on Internet and Information Systems*, vol. 13, no. 4, pp. 1975–1988, 2019.