applied sciences

MDPI

Article

Learning to Prioritize Test Cases for Computer Aided Design Software via Quantifying Functional Units

Fenfang Zeng [1,2], Shaoting Liu [3], Feng Yang [3], Yisen Xu [3], Guofu Zhou [3] and Jifeng Xuan [3,*]

1 Wuhan KM Information Technology Co., Ltd., Wuhan 430070, China
2 Huazhong University of Science and Technology, Wuhan 430070, China
3 School of Computer Science, Wuhan University, Wuhan 430072, China
* Correspondence: jxuan@whu.edu.cn

Abstract: Computer Aided Design (CAD) is a family of techniques that support the automation of designing and drafting 2D and 3D models with computer programs. CAD software is a software platform that provides the process from designing to modeling, such as AutoCAD or FreeCAD. Due to complex functions, the quality of CAD software plays an important role in designing reliable 2D and 3D models. There are many dependencies between defects in CAD software. Software testing is a practical way to detect defects in CAD software development. However, it is expensive to frequently run all the test cases for all functions. In this paper, we design an approach to learning to prioritize test cases for the CAD software, called `PriorCadTest`. The key idea of this approach is to quantify functional units and to train a learnable model to prioritize test cases. The output of the approach is a sequence of existing test cases. We evaluate `PriorCadTest` on seven modules of an open-source real-world CAD project, `ArtOfIllusion`. The Average Percentage of Fault Detect (APFD) is used to measure the effectiveness. Experimental results show that the proposed approach outperforms the current industrial practice without test case prioritization.

Keywords: CAD software; computer aided design; test case prioritization; software testing; feature engineering; manufacturing software

check for updates

Citation: Zeng, F.; Liu, S.; Yang, F.; Xu, Y.; Zhou, G.; Xuan, J. Learning to Prioritize Test Cases for Computer Aided Design Software via Quantifying Functional Units. *Appl. Sci.* **2022**, *12*, 10414. https://doi.org/10.3390/app122010414

Academic Editor: Juan-Carlos Cano

Received: 27 August 2022
Accepted: 9 October 2022
Published: 15 October 2022

1. Introduction

Computer Aided Design (CAD) is a family of techniques that replaces manual designing and drafting models with an automated process [1]. CAD software is a software platform that provides the process from designing to modeling, such as AutoCAD [2] or FreeCAD [3]. CAD software, an indispensable tool for CAD technology applications, provides the capability and adaptability of graphic designing and drafting. Due to complex functions, many defects may be hidden in the CAD software. This badly hurts the output of CAD software, i.e., the generation of 2D and 3D models. Two major reasons result in the defects in CAD software. One is the complexity of functionality in CAD software; the other is the dependencies among the defects. It is important to find a way to trigger defects at an early stage. Thus, software testing is an indispensable step in the development process of CAD software [4].

The basic idea of testing CAD software is to run test cases to detect potential defects [5]. The updating of functionality of CAD software requires frequently running test cases. A direct way is to apply regression testing to run test cases to ensure a new code update does not violate existing functional behaviors [6]. However, due to the scale of functions in CAD software, it is expensive to frequently run all the test cases for all functions [7].

A solution to reduce the cost of running test cases is to prioritize test cases to find defects early. In this paper, we design an approach (called `PriorCadTest`) to learning to prioritize test cases for CAD software. The output of `PriorCadTest` is a sequence of existing test cases. The key idea of `PriorCadTest` is to quantify functional units (A functional unit can be a component, a package, or a function in different programming languages. In this evaluation of this paper, we evaluate the results on program functions). Then, each functional unit being tested is converted into a vector. `PriorCadTest` trains a learnable model from known test results and ranks tests for new functional units. To the best of our knowledge, this paper is the first work that prioritizes test cases for CAD software.

We evaluate the proposed approach on seven modules of an open-source CAD project, `ArtOfIllusion` (Project `ArtOfIllusion`, http://www.artofillusion.org/, accessed on 16 May 2022). In the evaluation, the `Average Percentage of Fault Detect` (APFD) that counts covered statements is used to measure the effectiveness of test case prioritization. We selected the model with the best ranking result from the models in comparison and used this model to validate against the test set. Experimental results show that the proposed approach outperforms the current industrial practice without test case prioritization.

Application scenario and motivation. The source code of the CAD software can be frequently updated due to the code updates. A common way to ensure that a new code update does not violate existing program behaviors is regression testing [8]. In general, the process of regression testing is to run all the test cases if the source code is updated. The proposed approach in this paper is to re-rank test cases to trigger potential defects as the early stage. Applying this approach can reduce the time cost of running test cases and save the time of developing CAD software.

Contributions. This paper makes the following main contributions.

- A new approach for ranking test cases for CAD software. We proposed an automatic approach for converting each functional unit or each test case into a 103-dimensional numeric vector. Then, each functional unit and each test case is combined into a pair, which is converted into a 206-dimensional vector based on the coverage relationship between CAD software functions and test cases. A learnable ranking model of test cases is trained using the data of 206-dimension vectors (Section 3).
- An experimental setup on six ranking models of learning and ranking test cases for CAD software (Section 4).
- Evaluation results of the proposed approach `PriorCadTest` with six ranking models on a real open-source CAD software, `ArtOfIllusion`. We find that the random forest classifier is effective in ranking the test case for CAD software (Section 5).

The rest of this paper is organized as follows. Section 2 shows the basic background of testing CAD software. Section 3 presents the design of our approach, `PriorCadTest`. Experimental setup and results are presented in Sections 4 and 5. Section 6 discusses threats to validity. Finally, Section 7 concludes the paper and lists the future work.

2. Background

In this section, we discuss the background of CAD software, the background of software testing, and the related work.

2.1. CAD Software

CAD software is widely used in many fields, such as civil construction, machinery manufacturing, aerospace, landscape design, and urban planning. CAD software is a powerful tool for product innovation, with powerful graphic editing functions that enable accurate design of various 2D and 3D graphic models. The user interface of CAD software, which allows various operations to be performed through interactive menus or command lines. Meanwhile, CAD software supports secondary development. This highly increases the fields of applications. Figure 1 shows an example of using CAD software to draft a 3D model with three objects. Defects may be hidden in CAD software. This highly hurts the generation of CAD models.

In terms of model types, models in the CAD software can be generally divided into 2D CAD models and 3D CAD models. A 2D CAD model consists of several components in plane geometry. A 3D CAD model is a computer representation of the actual shape of the product into a three-dimensional model, which includes various information about the points, lines, surfaces, and shape bodies of the solid geometry. Depending on the requirements, CAD software can also be divided into architectural CAD software, mechanical CAD software, circuit CAD software, etc. [9].

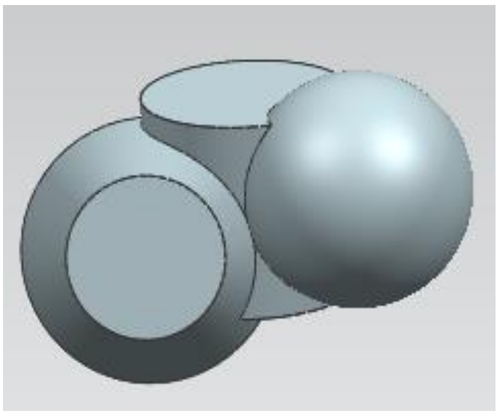

Figure 1. Sketch of a 3D model drafted by the CAD software.

2.2. Software Testing

With the development of software and IT industry, the number of software products grows rapidly. Software quality has attracted much attention from developers. In the 1980s, the basic theory and practical techniques of software testing were gradually formed [10]. The purpose of software testing is to check whether the software system meets the requirements. Software testing has been integrated into the entire development process.

Automation testing is a technique of software testing that uses an automated tool to validate various software testing requirements, including the management and implementation of testing activities and the development and execution of test scripts [11,12]. Software automation testing techniques are also classified into various types for different applications. Software functional testing aims to test the functionality of the software by giving appropriate input values, determining the output and verifying the actual output using the expected values. Functional testing techniques is widely applied. For example, Abbot Java GUI Test Framework [13] is mainly used for automated testing of Java GUI tests. Soapui [14] is mainly applied to web service testing. This tool is used to test web service through HTTP protocol.

The existence of software defects greatly harms the quality of software and increases the cost of software maintenance. To detect software defects early, software testing becomes an essential stage in software development [15]. In software testing, developers design test cases in anticipation of early detection of potential software defects. Test cases are critical to reducing software defects in rapid code integration. A typical development process is test-driven development, designed to drive iterative code development with test cases generated ahead of code details [16]. To improve software quality, developers do everything they can to find, locate, analyze, and fix code defects. Typically, before a software is released, developers often write a set of supporting test cases to test the correctness and stability of the software based on the functions that need to be implemented and the possible execution paths of the software. However, due to the complexity of the software functions and structure, the set of supporting test cases written by the developers often cannot cover all the situations of the software operation, and all software defects still occur from time to time.

2.3. Related Work

We summarize the work related to CAD software testing and test case prioritization as follows.

Existing researchers have studied several research topics of testing CAD software. Frome [17] studied the perceptions of users or developers of a CAD system; results show that the users' perceptions of a CAD system are quite different from the developers'.

Grinthal [18] introduced an overview of software quality assurance and its relation to user interfaces for CAD systems. Hallenbeck et al. [19] discussed a CAD supporting tool for designing a digital system that meets predefined testability requirements. Gelsinger et al. [20] proposed CAD tools, which are created to accelerate the design of the i486 CPU.

Sprumont and Xirouchakis [21] proposed a model of the CAD process that allows an adaptive man-machine task sharing by allocating the user interaction to the knowledge model. The CAD processes can be used to recursively define the CAD activities of high levels of abstraction. Wang and Nnaji [22] addressed a soft constraint representation scheme based on nominal intervals. Su and Zeng [23] introduced a test methodology that detects both catastrophic and parametric faults. Issanchou and Gauchi [24] proposed a novel feed-forward neural network model that considers the framework of the nonlinear regression models to construct designs. Veisz et al. [25] presented a comparison between the role of CAD and sketching in engineering. Their results suggest that it is necessary to emphasize the importance of sketching and the deep understanding to the utility of CAD tools at each stage of the design process.

Banerjee et al. [26] introduced a computer-aided-test (CAT) tool for mixed signal designs. The CAT tool provides a hardware efficient integrated solution. Bahar [27] presented the leading-edge research and development solutions, and identified future road-maps for design automation research areas. Ramanathan et al. [28] presented a novel approach to the test case prioritization problem that addresses this limitation. Chi et al. [29] proposed a new approach AGC (Additional Greedy method Call) sequence, the approach leverages dynamic relation-based coverage as measurement to extend the original additional greedy coverage algorithm in test case prioritization techniques. An empirical study [30] was conducted to examine the relative importance of the size and coverage attributes in affecting the fault detection effectiveness of a randomly selected test suite.

Gupta [31] proposed a novel prioritization algorithm that can be applied over both original and reduced test suites depending upon the size of test suites. Gupta [31] introduced search-based regression testing that is applied to improve the quality of the test suite in order to select a minimum set of test cases. The performance of different meta-heuristics for the test suite minimization problem is compared with a hybrid approach of the ant colony optimization and the genetic algorithm. Chen et al. [32] presented an adaptive random sequence approach based on clustering techniques using black-box optimization. Liu [33] studied the topic of ranking great amounts of documents based on their relation to a given query, i.e., the examination of the inner mechanics of the search engines. Mirarab and Tahvildari [34] proposed an approach based on the probability theory to incorporate source code changes, software fault-proneness, and test coverage into a unified model. Lin et al. [35] used a pairwise learning-to-rank strategy XGBoost to combine several existing metrics to improve the effectiveness of test case prioritization.

To the best of our knowledge, our work is the first technique that prioritizes test cases for CAD software via qualifying functional units. The idea of our work is to propose a practical way to reduce the time cost of frequent test execution.

3. Learning to Prioritize Test Cases for CAD Software, `PriorCadTest`

Our proposed approach, called `PriorCadTest`, learns to prioritize test cases for CAD software. This approach consists of two major phases, the learning phase and the ranking phase. The learning phase trains the ranking model based on known testing results, while the ranking phase uses the learned model to rank test cases for new functional unit. The ranking model in the approach is based on feature extraction. In this section, we show the overview, the feature extraction, the learning phase, and the ranking phase as follows.

3.1. Overview

Our proposed approach, `PriorCadTest`, aims to generate a sequence of test cases that are prioritized. The goal of this approach can prioritize test cases for a new functional unit (e.g., a new component of source code). Then, the prioritization can reduce the time of running test cases via triggering potential defects at the early stage.

Figure 2 shows the overview of the proposed approach to learning to prioritize test cases for CAD software. This approach consists of two major phases, the learning phase and the ranking phase. First, in the learning phase, a functional unit under test and a test case are converted into a function vector and a test vector, respectively. Each function vector and each test vector are connected into a vector. A learning model is then built based on the above extracted vectors. In our work, four machine learning models are used and the random forest model performs the best. Second, in the ranking phase, for any new functional unit, such as a new component under test, the ranking model combines the functional unit with each test. Such combination is also converted into a vector. Then, the ranking model ranks these vectors based on the ranking scores. The test case with the highest ranking score is ranked as the first in the sequence of running test cases.

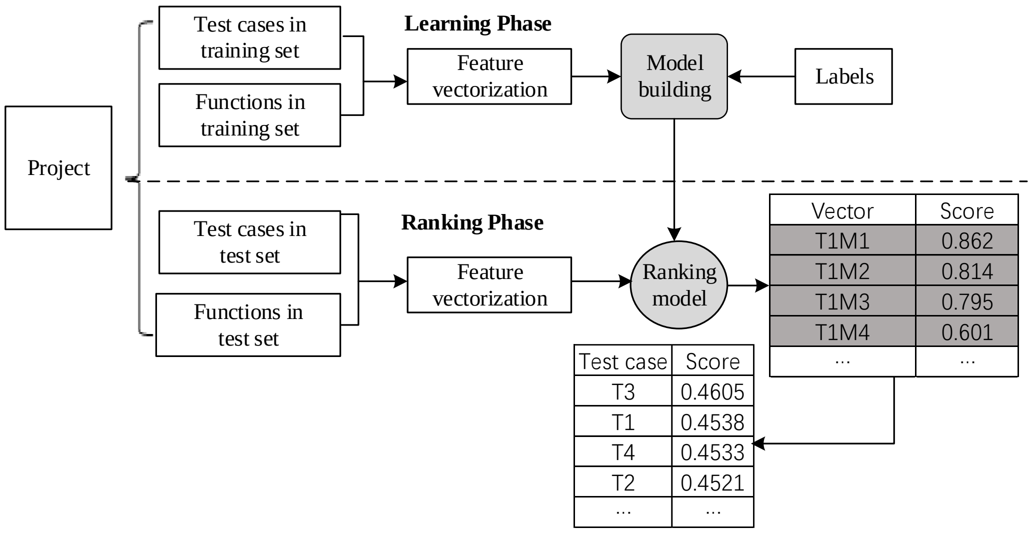

Figure 2. Overview of the approach to learning to prioritize test cases for CAD software.

3.2. Feature Extraction

To train a ranking model for test case prioritization, we convert each functional unit and each test case into a numeric vector. In many programming languages, a test case is a specific function that can be directly run in the testing framework, such as Java test cases in Java testing framework, JUnit [36]. This facilitates the conversion from source code to numeric vectors. In our work, both functional units and test cases are converted into a vector with the same dimensions.

Vector conversion. In our approach `PriorCadTest`, an off-the-shelf tool, CodeZhi, is used to conduct the conversion. CodeZhi is a tool of qualifying Java source code (Project CodeZhi, http://cstar.whu.edu.cn/pr/codezhi/, accessed on 20 May 2022).This tool can extract features at the method level for Java programs. CodeZhi tries to find representative features, which can distinguish methods well. It picked up 103 features as the metrics to profile Java methods. For example, the CodeZhi tool profiles the function *outset*(*double dist*) to get a feature vector with 103 dimensions as shown in Figure 3. A piece of compilable source code is converted into a vector with 103 dimensions. Each dimension indicates a manual-defined feature. The categories of features consist of features about statements, variables, objects, methods, operators and operands, complexity, code blocks, nested blocks, jumping statement, finals and statics, assignments, and distance. For example, the number of if-conditions and the cyclomatic complexity of the source code can be viewed as two features in feature extraction [37]. We denote a vector of the functional unit as M_i ($1 \leq i \leq m$), and m is the number of functional units. We denote a vector of the test case as T_j ($1 \leq j \leq n$), and n is the number of test cases. Then, each of $M_1, M_2, ..., M_m$ and $T_1, T_2, ..., T_n$ is a feature vector with 103 dimensions.

Vector of the function `outset(double dist)`

```
outset(double dist),0,0,6,6,1,12,0,0,6,6,6,12,6,6,0,1,6,0,
0,0,0,0,1,0,1,6,6,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0.00000,0.00000,1.00000,0.00000,0.00000,0.00000,0.00000,1,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,
0,0,0,6,0,0,0,2,2,2.00000,0,0,1,0,0,0,1,0,0
```

Figure 3. Example of the feature vector of a function `outset(double dist)`.

Test coverage matrix. In the CAD software, a functional unit may be covered by multiple test cases; and one test case may cover multiple functional units. We construct a test coverage matrix to show the coverage relationship between test cases and functional units. We run each test case and analyze test results to obtain a matrix C and $C_{i,j} \in C$ in the train set, where $C_{i,j}$ indicates that the ith test case covers the jth functional unit. The value of $C_{i,j}$ is 1 if a test case T_i has covered a functional unit M_j, and 0 otherwise. Table 1 is an example of a test coverage matrix.

Table 1. Example of a test coverage matrix. T_i indicate the i-th test case and M_j indicates the j-th functional unit.

	M_1	M_2	**...**	M_m
T_1	1	0	...	1
T_2	1	1	...	0
...	...	...	...	...
T_n	0	1	...	1

Vectors with labels. In our work, each vector of a functional unit and each vector of a test case are connected into a vector. This vector is a pair of the functional unit and the test case, called a *test pair* in this paper. The vector contains 206 dimensions and indicates the qualification of the coverage $C_{i,j} \in C$, where $C_{i,j}$ indicates that the ith test case covers the jth functional unit. To train a model that ranks test cases, we use a classifier to assign scores to the test cases. In our work, we label a vector with 206 dimension as 1 if $C_{i,j} = 1$ and the functional unit M_j contains a defect; otherwise, we label the vector as 0. Such labeling shows that the value of the label indicates the test coverage of a functional unit with a defect. Base on the labeling, each vector of the test coverage (indicating the coverage of a functional unit by a test case) and its label are conducted. In the learning phase of our approach, the labels are known as 1 or 0; in the ranking phase, the label is unknown, and the ranking model will assign a score to each 206-dimension vector. Then test cases can be ranked according to the assigned scores of test cases.

3.3. Learning Phase

Based on the test pairs (i.e., combined vectors) and their labels, we employ a machine learning model to assign a score to each new test pair. In general, any classifier in machine learning can be workable. In our experiment, we use six classifiers to conduct the evaluation. We briefly describe the classifiers in our work.

The Classification and Regression Trees (CART) [38] represents a data-driven, model-based, nonparametric estimation method that implements the define-your-own-model approach. In other words, CART is a method that provides mechanisms for building a custom-specific, nonparametric estimation model based solely on the analysis of measurement project data, called training data. The random forest [39] is a combination of tree predictors such that each tree depends on the values of a random vector sampled independently and with the same distribution for all trees in the forest. The Support Vector Machine (SVM) [40] is a binary classification model whose basic model is a linear classifier defined by maximizing the interval on the feature space, which distinguishes it from a perceptron. SVM also includes kernel tricks, which make them essentially nonlinear classifiers. The learning strategy of SVM is interval maximization and can be formalized as a problem of solving convex quadratic programming. The Bayesian network is ideal for taking an event that occurred and for predicting the likelihood of a possible cause being the contributing factor. For example, a Bayesian network could represent the probabilistic relationships between diseases and symptoms. Convolutional neural network (CNN) [41] is a type of feed-forward neural network that includes convolutional computation. CNN is one of the representative algorithms of deep learning. Recurrent Neural Network (RNN) [42] is a type of recursive neural network that takes sequential data as input. The evolutionary direction of the sequence determines the next node of the recursion, and all nodes are connected in a chain-like manner. Given a classifier model, the test pairs and their labels can be used to build a concrete model that can be used in the follow-up ranking phase.

3.4. Ranking Phase

In the learning model, a classifier is built to show the coverage of functional units that contain defects. In the ranking phase, a new functional unit is connected with each test case to conduct a vector of 206 dimensions. This new vector (i.e., a test pair of the functional unit M_j and the test case T_i) has no label, and the ranking model can assign a score to the vector. This score indicates the probability that the test case can cover potential defects. Then all the test cases can be ranked as a sequence based on the assigned scores.

In the ranking phase, the score of one test case T_i is defined as the average learned probability of finding defects in all functional units by the test case. The definition is shown as follows:

$$score(T_i) = \frac{\sum_{j=1}^{m} Pr(M_j, T_i)}{m} \tag{1}$$

where M_j is the j-th functional unit, m is the number of all functional units under test, and $Pr(M_j, T_i)$ is the learned probability for a test pair of the functional unit M_j and the test case T_i by the ranking model. A high value of $score(T_i)$ means the i-th test case is likely to find defects.

4. Experimental Setup

In this section, we describe the research questions, the data preparation, and the evaluation metrics.

4.1. Research Questions

To evaluate the proposed approach, we design two research questions (RQs) to conduct the evaluation. In RQ1, we compare the effectiveness of machine learning models (including deep learning models) that could be embedded in our approach; In RQ2, we evaluate the ability of test case prioritization.

- **RQ1. Can we find a better ranking model to prioritize test cases for CAD software?** To save the time cost of defect detection, developers want to find out defects within a limited time. That is, developers need to automatically prioritize test cases before executing them. In this paper, we design RQ1 to explore the effectiveness of machine learning models in the model of test case prioritization.

- **RQ2. How effective is the proposed approach in test case prioritization in testing CAD software?** In CAD software, a single test case may cover multiple functions, and a single function may be covered by multiple test cases. Therefore, it is difficult for developers to determine which test case should be executed first. We design RQ2 to evaluate the ability of test case prioritization in testing CAD software.

4.2. Data Preparation and Implementation

Project under evaluation. To investigate the ranking of test cases in CAD software, we select the most widely-used open-source CAD software, Project `ArtOfIllusion`, as an experimental dataset. We select Project `ArtOfIllusion` based on the following reasons. First, to conduct the experiment, we need an open-source project; second, our approach relies on the conversion from source code to vector and the tool for such conversion, i.e., CodeZhi, is designed for Java programs; third, we tend to select a well-known CAD software in the evaluation. Based on the above three reasons, we check the CAD software written in Java in GitHub (Github, http://github.com/, accessed on 16 May 2022) and then select the one candidate, Project `ArtOfIllusion`. (Source code of Project `ArtOfIllusion`, http://github.com/ArtOfIllusion/ArtOfIllusion/, accessed on 16 May 2022). Project `ArtOfIllusion` is a free, multi-platform modeling, animation, and rendering suite written in Java programming language. `ArtOfIllusion` features a simple and streamlined interface to a broad array of powerful features, including key-frame-based and pose-based animations as well as a built-in raytracer. There are 15 modules in `ArtOfIllusion`, including animation, icons, image, keystroke, material, math, object, procedural, script, texture, titleImages, ui, unwrap, util, and view. There are seven modules with pre-defined test cases, i.e., module animation, math, object, procedural, raytracer, texture, and util. We use the source code and test cases in these seven modules. To increase the diversity of the datasets, we created a new dataset based on 5-fold cross validation [43]. We randomly divided all the files in the seven modules into five folds. Table 2 shows the summary of two datasets in Project `ArtOfIllusion`, including Dataset 1 with seven modules and Dataset 2 with five folds. Due to the limited development budget allocated for open-source projects, some of functions are not directly covered by manually-written test cases.

Table 2. Dataset information with functions and test cases in Project `ArtOfIllusion`. # *Functions* indicates the number of functional units in the module. # *Test Cases* indicates the number of test cases that are manually-written for testing the module.

Module	#Functions	#Test Cases
math	192	11
object	897	43
procedural	577	12
animation	994	19
raytracer	0 †	2
texture	459	2
util	65	2
Sum	**3184**	**91**

† Module raytracer only provides test cases that examine the functional behaviors of multiple modules. Thus, there are no functions in the module.

Defect seeding. We employ Project `ArtOfIllusion` in the evaluation. To conduct the scenario of triggering defects, we follow the existing work [44] and use program mutation to automatically seed defects in the source code of Project `ArtOfIllusion`. Program mutation is a technique of modifying the source code in a target program to create a defective program. The purpose of program mutation is to obtain a program with seeded defects [45]. To seed defects in the source code, we use a simple way to mutate the original source code. We select all functions without return values as the mutation candidates and apply mutation rules that remove 30% of the statements in functions. Such kinds of functions are considered as defective, and the other functions are considered as non-defective. Such mutation is easy to be implemented and can keep the programs compilable without any manual effort. The program mutation is implemented based on the Spoon tool (Spoon, http://spoon.gforge.inria.fr/, accessed on 17 May 2022). to remove the source code from the original code base. Figure 4 shows an example of seeding a defect via program mutation. The two code snippets are a function `outset()` before and after program mutation. The *Mutation Code* removes `maxy += dist` (line 7) and `maxz += dist` (line 8) from the *Original Code*.

Original Code

```
1 public final void outset(double dist)
2 {
3     minx -= dist;
4     miny -= dist;
5     minz -= dist;
6     maxx += dist;
7     maxy += dist;
8     maxz += dist;
9 }
```

Mutation Code

```
1 public final void outset(double dist)
2 {
3     minx -= dist;
4     miny -= dist;
5     minz -= dist;
6     maxx += dist;
7 }
```

Figure 4. Example of seeding a defect in a function `outset()`.

Dataset. We conducted cross-validation experiments to evaluate the proposed approach. In the seven modules of Project `ArtOfIllusion`, we selected four modules with over 10 test cases as the test set of each round of evaluation. Then, module animation, math, object, and procedural are, respectively, selected as the test set; the remaining six modules are formed the train set. However, in module animation, no defective functions are covered by manually written test cases. Then, in the follow-up evaluation, we do not consider module animation as a test set. Finally, three rounds of cross-validation experiments are conducted. In the five folds of Project `ArtOfIllusion`, we used each fold in turn as the test set and the remaining four folds as the train set, and conducted five rounds of experiments. So we conducted eight rounds of experiments from different datasets in the whole Project `ArtOfIllusion`. Table 3 shows the data description of the train sets and the test sets of eight rounds of evaluation.

Implementation. We implemented our approach in Spoon, Codezhi, and JaCoCo. Spoon is used in program mutation; CodeZhi is used in feature extraction; JaCoCo is used to collect test coverage and to filter out invalid mutants (JaCoCo, http://github.com/jacoco/jacoco/, accessed on 20 May 2022).

Table 3. Data information of training sets and test sets. We show the division of training sets and test sets of eight rounds of evaluation. # *Defective functions covered by tests* is the number of defective functions that are covered by test cases.

Dataset	Module or Fold as Test Set	Functions and Test Cases	Training Set	Test Set
Dataset 1 based on modules	math	# Functions	2992	192
		# Defective functions	1488	66
		# Test Cases	80	11
		# Defective functions covered by tests	69	8
	object	# Functions	2287	897
		# Functions with seeding defects	1160	394
		# Test Cases	48	43
		# Defective functions covered by tests	36	33
	procedural	#Functions	2607	577
		# Functions with seeding defects	1217	337
		# Test Cases	79	12
		# Defective functions covered by tests	45	24
	animation (not used in evaluation)	# Functions	2190	994
		# Functions with seeding defects	1051	503
		# Test Cases	72	19
		# Defective functions covered by tests	69	0
Dataset 2 based on folds	Fold 1	#Functions	2548	636
		# Functions with seeding defects	1431	123
		# Test Cases	91	91
		# Defective functions covered by tests	64	5
	Fold 2	#Functions	2548	636
		# Functions with seeding defects	11329	1431
		# Test Cases	91	91
		# Defective functions covered by tests	59	10
	Fold 3	#Functions	2548	636
		# Functions with seeding defects	1267	287
		# Test Cases	91	91
		# Defective functions covered by tests	56	13
	Fold 4	#Functions	2548	636
		# Functions with seeding defects	1151	403
		# Test Cases	91	91
		# Defective functions covered by tests	50	19
	Fold 5	#Functions	2544	660
		# Functions with seeding defects	1038	516
		# Test Cases	91	91
		# Defective functions covered by tests	47	22

4.3. Evaluation Metrics

We evaluate the effectiveness of our approach with two sets of metrics, including *F-score* as well as *Accuracy* to show the performance of classifiers and *APFD* to show the performance of ranking test cases.

Evaluating the classifiers in test case prioritization. We measured the evaluation of classifiers in the ranking models in test case prioritization with typical measurements: *Precision*, *Recall*, *F-score*, and *Accuracy*. Those measurements are defined based on True Positive (TP), False Positive (FP), False Negative (FN), and True Negative (TN) for test pairs, i.e., a vector that combines a functional unit and a test case.

- TP: # of defective test pairs that are predicted as defective.
- FP: # of undefective test pairs that are predicted as defective.
- FN: # of defective test pairs that are predicted as undefective.
- TN: # of undefective test pairs that are predicted as undefective.

Then, we defined the metrics in the evaluation of prediction as follows,

$$Precision = \frac{TP}{TP + FP} \tag{2}$$

$$Recall = \frac{TP}{TP + FN} \tag{3}$$

$$F\text{-}score = \frac{2 \times Precision \times Recall}{Precision + Recall} \tag{4}$$

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN} \tag{5}$$

where *Precision* is the proportion of returned results that are truly correct; *Recall* is the proportion of the truly correct number in all retrieved results in the test set; *F-score* is the trade-off between *Precision* and *Recall*; and *Accuracy* is the proportion of the correct number in all results in the whole dataset.

Evaluating the ranking of test cases in test case prioritization. Suppose the program under test contains k defects and the set of test cases contains n test cases. We use the `Average Percentage of Fault Detect` (APFD) [46] to evaluate the effectiveness of the ranking of test cases generated by each of the ranking models.

$$APFD = 1 - \frac{TF_1 + TF_2 + TF_3 \cdots + TF_s}{sn} + \frac{1}{2n} \tag{6}$$

where TF_k denotes the test case in the ranking of test cases that first covers the kth defect in the program. That is, a high value of the APFD metric indicates the defects are defective in the early stage.

5. Experimental Results

In this section, we present the evaluation results and empirically answer the two RQs about the effectiveness.

5.1. RQ1. Can We Find a Better Ranking Model to Prioritize Test Cases for CAD Software?

The goal of our work `PriorCadTest` is to re-rank test cases to trigger defects early. This can reduce the time cost of frequently running test cases. `PriorCadTest` relies on a ranking model, which is a binary classifier in machine learning. In RQ1, we check the effectiveness of six different classifiers based on the dataset of Project `ArtOfIllusion`.

Method. The evaluation method in the section is to compare the results of the proposed approach, `PriorCadTest`, when we change the machine learning algorithms (including deep learning algorithms). First, we extracted functions and their corresponding test cases from Project `ArtOfIllusion` and constructed a test coverage matrix between functions and test cases; second, we vectorized functions and test cases using the CodeZhi tool and merged the functions and test cases according to the test coverage matrix; finally, we trained six learning-based ranking models, respectively. The six ranking models are the decision trees (CART), the random forest (RF), the SVM, the Bayesian networks (BN), the convolutional neural networks (CNN), and the recurrent neural network (RNN).

Result and analysis. We trained six ranking models and evaluated the effectiveness using *Precision*, *Recall*, *F-score*, *Accuracy*, *Time*, and *Memory*. Table 4 shows the evaluation results on the six ranking models for eight rounds of evaluation. As shown in Table 4, the CART and the random forest outperform the SVM, the Bayesian networks according to the values of four evaluation metrics in all the test sets. As shown in Table 4, the CART and the random forest outperform the SVM, the Bayesian networks, the CNN, and the RNN according to the values of four evaluation metrics (*Precision*, *Recall*, *F-score*, and *Accuracy*) in all the test sets. When the SVM model ranks the test cases, it predicts all functions as non-defective functions. Thus, the values of *Precision* and *F-score* are not available (N/A) for the SVM model when we use Module object as the test set. The values of *Precision* and *F-score* are N/A for the SVM model, the CNN model, and the RNN model when we use Module object and Fold 4 as the test set. In terms of *Precision*, the CART model is lower than the random forest model in all test sets. This shows that the random forest model is better at truly predicting defects for three different test sets. The random forest model does not perform well in terms of memory consumption and time consumption based on two different datasets. In terms of *Recall*, the CART model is higher than the random forest model. The CART model is also slightly higher than the random forest model when we compare the values of *F-score* and *Accuracy*. The evaluation metrics of the random forest model also performs well in the evaluation.

Conclusion. The main purpose of this paper is to trigger defects in CAD software by test cases in the early stage. The results in Table 4 show that the CART model and the random forest model are better than the other ranking models.

5.2. RQ2. How Effective Is the Proposed Approach in Test Case Prioritization in Testing CAD Software?

We evaluate whether the ranking of test cases can trigger the defects early. During testing the CAD software, given the same set of test cases, if a sequence of test cases (i.e., a ranked list of test cases) can trigger defects earlier than another sequence of test cases, we consider the earlier sequence is better. The effectiveness of such sequences of test cases can be evaluated with the pre-defined APFD metric [46]. A high value of APFD indicates an effective sequence of test case ranked. We designed RQ2 to evaluate the test case sequences.

Method. In the comparison, we employed the random ranking of test cases as a baseline. We validated the test cases, respectively, for the CART, the random forest, the SVM, the Bayesian networks, the CNN, and the RNN. In our study, the number of triggered defects in top-10, top-20, and top-30 test cases of each test case sequence is reported as a metric to evaluate the capability of the ranking model. A high value of the number of defects detected in top-*k* indicates that the test case ranking performs well.

Result and analysis. Table 5 shows the evaluation results of ranking test cases for the CAD software. Table 5 demonstrates that among the top-10 test cases. The Bayesian networks can find the least number of defects and the random forest detects the most defects from dataset one. In the top-20 test cases, the random ranking finds the least number of defects and the random forest model detects the most defects for most modules, but the values are the same in module procedura, module math, and part 2. The reason is that the number of defective functions is less than 20. In the top-30 test cases, the random ranking model detects the least number of defects and the random forest model detected the most defects for most modules, but the values are equal in module procedura, module math, and Fold 2. The number of defective functions is less than 30. Among the five models in comparison, the random forest model shows the largest APFD value and the random ranking model shows the smallest value.

To show details, we use Table 6 to directly present one example of the top-10 results of ranking test cases by the random forest model. As shown in Table 6, `point_outside_influence_radius_is_0` and `point_inside_radius_is_greater_than_1` have the highest value (0.4605) among the top-10 test cases. This means that these two test cases are more likely to find defects than the other test cases.

Table 4. Results of six ranking models under six evaluation metrics. *Precision* is the proportion of returned results that are truly correct; *Recall* is the proportion of the truly correct results in all retrieved results from the test set; *F-score* is the trade-off between *Precision* and *Recall*; *Accuracy* is the proportion of correct results in all results from the whole data set; *Time* is the time consumption on the test set; and *Memory* is the memory consumption on the test set. N/A indicates the denominator is zero.

Test Set	Ranking Model	Precision	Recall	F-Score	Accuracy	Time (ms)	Memory (MB)
math	CART	87.31%	88.54%	87.93%	87.03%	37.49	**200.99**
	RF	91.14%	81.62%	86.12%	85.97%	30.23	246.34
	SVM	53.33%	100%	69.56%	53.33%	29.17	294.53
	BN	22.86%	0.51%	0.99%	46.03%	**27.96**	251.96
	CNN	53.33%	100%	69.56%	53.33%	41.85	353.83
	RNN	53.33%	100%	69.59%	53.33%	20.01	275.79
object	CART	88.12%	84.50%	86.28%	85.66%	126.66	710.58
	RF	89.45%	79.09%	83.96%	83.88%	202.42	826.45
	SVM	N/A	0%	N/A	46.67%	**117.87**	**558.3**
	BN	64.01%	21.82%	32.55%	51.76%	**151.64**	696.19
	CNN	N/A	0%	N/A	46.67%	206.12	897.85
	RNN	N/A	0%	N/A	46.67%	285.83	768.95
procedural	CART	92.89%	83.25%	87.80%	87.67%	40.91	218.17
	RF	92.43%	80.22%	85.89%	85.95%	**35.94**	280.97
	SVM	53.33%	100%	69.56%	53.33%	60.62	219.94
	BN	22.58%	0.5%	0.97%	46.03%	63.37	288.06
	CNN	53.33%	100%	69.56%	53.33%	46.21	338.44
	RNN	53.33%	100%	69.59%	53.33%	74.97	**193.84**
Fold 1	CART	29.78%	43.56%	35.38%	43.71%	65.31	273.88
	RF	50.99%	92%	65.61%	65.88%	75.31	335.16
	SVM	35.38%	100%	52.26%	35.38%	72.88	301.46
	BN	65.04%	71.11%	67.94%	76.26%	**30.17**	**189.84**
	CNN	35.38%	100%	52.26%	35.38%	41.64	257.77
	RNN	35.38%	100%	52.26%	35.38%	36.03	331.65
Fold 2	CART	42.06%	79.67%	55.06%	74.84%	78.34	320.61
	RF	47.27%	63.41%	54.17%	79.25%	62.81	**169.62**
	SVM	19.34%	100%	32.41%	19.34%	69.69	284.75
	BN	16.92%	17.89%	17.39%	67.19%	**35.03**	178.8
	CNN	19.34%	100%	32.41%	19.34%	75.26	249.2
	RNN	19.34%	100%	32.41%	19.34%	41.59	226.92
Fold 3	CART	55.56%	60.98%	58.14%	60.38%	77.24	275.33
	RF	59.75%	84.32%	69.94%	67.3%	38.26	338.93
	SVM	45.13%	100%	62.19%	45.13%	**35.02**	**175.69**
	BN	20.97%	18.19%	19.44%	32.23%	72.33	389.09
	CNN	45.13%	100%	62.19%	45.13%	72.9	336.95
	RNN	45.13%	100%	62.19%	45.13%	59.32	189.21
Fold 4	CART	41.04%	21.59%	28.29%	30.66%	48.16	278.7
	RF	58.96%	44.91%	50.99%	45.28%	51.04	282.38
	SVM	N/A	0	N/A	36.64%	**32.36**	380.15
	BN	7.27%	2.98%	4.23%	14.47%	60.11	226.24
	CNN	N/A	0	N/A	36.64%	35.28	**185.58**
	RNN	N/A	0	N/A	36.64%	63.32	217.7
Fold 5	CART	73.53%	38.99%	50.96%	39.84%	42.56	**221.22**
	RF	87%	80.9%	83.84%	75%	**32.09**	248.61
	SVM	80.63%	100%	89.27%	80.63%	70.12	249.8
	BN	53.06%	15.2%	23.64%	21.25%	44.91	353.38
	CNN	80.63%	100%	89.27%	80.63%	33.8	294.55
	RNN	80.63%	100%	89.27%	80.63%	65.31	253.19

Table 5. Evaluation results of the six models and the baseline of random rankings. We show the result in top-10, top-20, top-30, and the APFD. The top-*k* metric indicates the number of defects that are found by the top *k* test cases in an ordered sequence of test cases. The zero value in the evaluation metrics means that no defective method is detected by the test cases.

Data Set	Test Set	Ranking Model	Top-10	Top-20	Top-30	APFD
Dataset 1	math	CART	8	8	8	59.09%
		RF	8	8	8	**85.23%**
		SVM	8	8	8	61.36%
		BN	5	8	8	71.59%
		CNN	8	8	8	61.36%
		RNN	8	8	8	61.36%
		RR	7	8	8	48.86%
	object	CART	14	19	27	65.29%
		RF	17	29	33	**72.83%**
		SVM	16	23	27	69.10%
		BN	13	21	30	65.57%
		CNN	16	23	27	69.10%
		RNN	16	23	27	69.10%
		RR	15	16	16	53.66%
	procedural	CART	24	24	24	70.49%
		RF	24	24	24	80.21%
		SVM	24	24	24	84.38%
		BN	7	24	24	**90.63%**
		CNN	24	24	24	84.38%
		RNN	24	24	24	84.38%
		RR	23	24	24	48.96%
Dataset 2	Fold 1	CART	8	8	10	66.5%
		RF	9	10	10	**73.03%**
		SVM	8	9	10	63.66%
		BN	8	8	9	67.54%
		CNN	8	9	10	63.66%
		RNN	8	9	10	63.66%
		RR	7	8	8	58.11%
	Fold 2	CART	5	5	5	72.9%
		RF	5	5	5	73.38%
		SVM	5	5	5	**73.5%**
		BN	3	5	5	62.07%
		CNN	5	5	5	73.5%
		RNN	5	5	5	73.5%
		RR	4	5	5	69.72%
	Fold 3	CART	6	9	12	66.14%
		RF	8	12	13	**73.51%**
		SVM	6	10	13	67.12%
		BN	7	11	13	68.94%
		CNN	6	10	13	67.12%
		RNN	6	10	13	67.12%
		RR	5	8	11	58.39%

Table 5. *Cont.*

Data Set	Test Set	Ranking Model	Top-10	Top-20	Top-30	APFD
	Fold 4	CART	9	15	18	69.23%
		RF	11	17	19	**71.37%**
		SVM	8	14	17	63.25%
		BN	8	13	16	61.51%
		CNN	8	14	17	63.25%
		RNN	8	14	17	63.25%
		RR	9	13	16	65.78%
	Fold 5	CART	11	15	19	67.83%
		RF	13	18	22	**80.21%**
		SVM	11	16	20	73.22%
		BN	9	12	16	64.44%
		CNN	11	16	20	73.22%
		RNN	11	16	20	73.22%
		RR	7	11	13	57.64%

Table 6. Top-10 of results of ranking test cases based on the random forest model. *Function* is the name of a test case. *Score* is the average probability that a test case is identified in the model. A higher value indicates that the test case is more likely to trigger a defect.

Rank	Function	Score
1	point_outside_influence_radius_is_0	0.4605
2	point_inside_radius_is_greater_than_1	0.4605
3	gradient_estimate_within_delta	0.4538
4	point_between_radius_and_influence_is_between_0_and_1	0.4533
5	gradient_estimate_at_influence_edge	0.4521
6	testDuplicateAll	0.4476
7	testDuplicateWithTracks	0.4467
8	testCopyInfo	0.4434
9	setObjectInfoMaterial	0.4432
10	testDuplicateWithNewGeometryAndTracks	0.4426

Conclusion. Experimental results show that the random forest model is able to find the highest number of defects in the top-*k* metric compared with the other six models. This means that the random forest model is more capable of ranking test cases. Therefore, we consider that the random forest should be an effective choice for finding defects.

6. Discussion

In the paper, we propose an approach to learning to prioritize test cases for CAD software. We present the discussions as follows. We present the threats as follows.

Generality. We evaluated the effectiveness of our approach `PriorCadTest` with an open-source CAD software, Project `ArtOfIllusion`. Experimental results show that the proposed approach to test case prioritization is better than the current practice. In the evaluation, Project `ArtOfIllusion` is implemented in the Java programming language. However, our proposed approach can be applied to test case prioritization in CAD projects that are written in other programming languages, like C or C#. The proposed approach can be generalized via the implementation on qualifying functions in other programming languages.

Industrial practice. In current industrial practice, the test case prioritization in development of CAD software is immature. Most of the CAD projects directly apply the whole test suite to conduct the testing process. This is expensive since the version update is common in the development of CAD software. The basic goal of test case prioritization in this paper can be viewed as a trial on reducing the time cost of testing in CAD software.

7. Conclusions and Future Work

In this paper, we propose an approach to learning to prioritize test cases for CAD software, called `PriorCadTest`. Our proposed approach combines vectors of test cases and vectors of functional units according to the coverage matrix. The combined vectors are employed as the input to train a random forest classifier to category test cases. For a new module or component under test, the trained model is used to assign a score to each candidate test case. In the evaluation, we employ an open-source real-world CAD software, Project `ArtOfIllusion` as the dataset.

In future work, we plan to apply our approach to CAD software in other programming languages. We want to collect defect data from other CAD software to show the results in the evaluation. The approach in this paper aims to test functional units in the source code. In CAD software, a single function may be combined into a mixture of several application scenarios. For example, a function could be a cylindrical model, a conical model, a ball model, or their combinations. In future work, we plan to explore the construction of test coverage matrix across functional units. Improving the effectiveness of our approach is also a direction for future work. Our proposed approach combines the test cases with function features based on the test coverage matrix. The test coverage matrix may be a sparse matrix, which may take a long running time in the model training. Thus, we plan to explore other data structures to replace the test coverage matrix to reduce the space complexity. Meanwhile, to save the execution time of `PriorCadTest`, we will parallel the implementation of our approach in future work. We plan to explore to combine the random forest model with new strategies to save on the time and memory cost.

Author Contributions: Data curation, S.L.; Funding acquisition, J.X.; Investigation, F.Z. and J.X.; Methodology, Y.X.; Project administration, G.Z.; Software, F.Z. and F.Y.; Writing—original draft, S.L.; Writing—review & editing, J.X. All authors have read and agreed to the published version of the manuscript.

Funding: This work is supported by the National Key R&D Program of China under Grant No. 2020YFB1709000, the National Natural Science Foundation of China under Grant Nos. 61872273, 62141221, and 62202344.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The dataset and the prototype of `PriorCadTest` in Java programming language are publicly available at http://cstar.whu.edu.cn/p/priorcad/, accessed on 29 August 2022.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Singh, J.; Perera, V.; Magana, A.J.; Newell, B.; Wei-Kocsis, J.; Seah, Y.Y.; Strimel, G.J.; Xie, C. Using machine learning to predict engineering technology students' success with computer-aided design. *Comput. Appl. Eng. Educ.* **2022**, *30*, 852–862. https://doi.org/10.1002/cae.22489.
2. Lane, H.C.; Zvacek, S.; Uhomoibhi, J. (Eds.) *Computer Supported Education, Proceedings of the 11th International Conference on Computer Supported Education, CSEDU 2019, Heraklion, Crete, Greece, 2–4 May 2019; Volume 2*; SciTePress: Setubal, Portugal, 2019.
3. Hatton, D. eights: BS 8888: 2011 first angle projection drawings from FreeCAD 3D model. *J. Open Source Softw.* **2019**, *4*, 974. https://doi.org/10.21105/joss.00974.
4. Agarwal, S.; Sonbhadra, S.K.; Punn, N.S. Software Testing and Quality Assurance for Data Intensive Applications. In Proceedings of the EASE 2022: The International Conference on Evaluation and Assessment in Software Engineering 2022, Gothenburg, Sweden, 13–15 June 2022; pp. 461–462. https://doi.org/10.1145/3530019.3533678.
5. Karadzinov, L.; Cvetkovski, G.; Latkoski, P. (Eds.) Power control in series-resonant bridge inverters, In Proceedings of the IEEE EUROCON 2017—17th International Conference on Smart Technologies, Ohrid, Macedonia, 6–8 July 2017; IEEE: New York, NY, USA, 2017.
6. Selvaraj, H.; Chmaj, G.; Zydek, D. (Eds.) FPGA Implementation for Epileptic Seizure Detection Using Amplitude and Frequency Analysis of EEG Signals, In Proceedings of the 25th International Conference on Systems Engineering, ICSEng 2017, Las Vegas, NV, USA, 22–24 August 2017; IEEE Computer Society: New York, NY, USA, 2017.

7. Chi, Z.; Xuan, J.; Ren, Z.; Xie, X.; Guo, H. Multi-Level Random Walk for Software Test Suite Reduction. *IEEE Comput. Intell. Mag.* **2017**, *12*, 24–33. https://doi.org/10.1109/MCI.2017.2670460.
8. Ali, N.B.; Engström, E.; Taromirad, M.; Mousavi, M.R.; Minhas, N.M.; Helgesson, D.; Kunze, S.; Varshosaz, M. On the search for industry-relevant regression testing research. *Empir. Softw. Eng.* **2019**, *24*, 2020–2055. https://doi.org/10.1007/s10664-018-9670-1.
9. Lanchares, J.; Garnica, O.; de Vega, F.F.; Hidalgo, J.I. A review of bioinspired computer-aided design tools for hardware design. *Concurr. Comput. Pract. Exp.* **2013**, *25*, 1015–1036. https://doi.org/10.1002/cpe.2957.
10. Yue, T.; Arcaini, P.; Ali, S. Quantum Software Testing: Challenges, Early Achievements, and Opportunities. *ERCIM News* **2022**, *2022,128*.
11. Denisov, E.Y.; Voloboy, A.G.; Biryukov, E.D.; Kopylov, M.S.; Kalugina, I.A. Automated Software Testing Technologies for Realistic Computer Graphics. *Program. Comput. Softw.* **2021**, *47*, 76–87. https://doi.org/10.1134/S0361768820080034.
12. Xuan, J.; Martinez, M.; Demarco, F.; Clement, M.; Marcote, S.R.L.; Durieux, T.; Berre, D.L.; Monperrus, M. Nopol: Automatic Repair of Conditional Statement Bugs in Java Programs. *IEEE Trans. Software Eng.* **2017**, *43*, 34–55. https://doi.org/10.1109/TSE.2016.2560811.
13. IEEE publisher,In Proceedings of the 5th IEEE/ACM International FME Workshop on Formal Methods in Software Engineering, FormaliSE@ICSE 2017, Buenos Aires, Argentina, 27 May 2017; IEEE: New York, NY, USA, 2017.
14. Radhakrishna, S.; Nachamai, M. Performance inquisition of web services using soap UI and JMeter. In Proceedings of the 2017 IEEE International Conference on Current Trends in Advanced Computing (ICCTAC), Bangalore, India, 2–3 March 2017; IEEE: New York, NY, USA, 2017; pp. 1–5.
15. Ma, P.; Cheng, H.; Zhang, J.; Xuan, J. Can this fault be detected: A study on fault detection via automated test generation. *J. Syst. Softw.* **2020**, *170*, 110769. https://doi.org/10.1016/j.jss.2020.110769.
16. Omri, S. Quality-Aware Learning to Prioritize Test Cases. Ph.D. Thesis, Karlsruhe Institute of Technology, Karlsruhe, Germany, 2022.
17. Frome, F.S. Improving color CAD Systems for Users: Some Suggestions from Human Factors Studies. *IEEE Des. Test* **1984**, *1*, 18–27. https://doi.org/10.1109/MDT.1984.5005571.
18. Grinthal, E.T. Software Quality Assurance and CAD User Interfaces. *IEEE Des. Test* **1986**, *3*, 39–48. https://doi.org/10.1109/MDT.1986.295043.
19. Hallenbeck, J.J.; Kanopoulos, N.; Vasanthavada, N.; Watterson, J.W. CAD Tools for Supporting System Design for Testability. In Proceedings of the International Test Conference 1988, Washington, DC, USA, 12–14 September 1988; p. 993. https://doi.org/10.1109/TEST.1988.207889.
20. Gelsinger, P.; Iyengar, S.; Krauskopf, J.; Nadir, J. Computer aided design and built in self test on the i486TM CPU. In Proceedings of the 1989 IEEE International Conference on Computer Design (ICCD): VLSI in Computers and Processors, Cambridge, MA, USA, 2–4 October 1989; pp. 199–202. https://doi.org/10.1109/ICCD.1989.63355.
21. Sprumont, F.; Xirouchakis, P.C. Towards a Knowledge-Based Model for the Computer Aided Design Process. *Concurr. Eng. Res. Appl.* **2002**, *10*, 129–142. https://doi.org/10.1177/1063293X02010002636.
22. Wang, Y.; Nnaji, B.O. Solving Interval Constraints by Linearization in Computer-Aided Design. *Reliab. Comput.* **2007**, *13*, 211–244. https://doi.org/10.1007/s11155-006-9023-4.
23. Su, F.; Zeng, J. Computer-Aided Design and Test for Digital Microfluidics. *IEEE Des. Test Comput.* **2007**, *24*, 60–70. https://doi.org/10.1109/MDT.2007.9.
24. Issanchou, S.; Gauchi, J. Computer-aided optimal designs for improving neural network generalization. *Neural Netw.* **2008**, *21*, 945–950. https://doi.org/10.1016/j.neunet.2008.05.012.
25. Veisz, D.; Namouz, E.Z.; Joshi, S.; Summers, J.D. Computer-aided design versus sketching: An exploratory case study. *Artif. Intell. Eng. Des. Anal. Manuf.* **2012**, *26*, 317–335. https://doi.org/10.1017/S0890060412000170.
26. Banerjee, S.; Mukhopadhyay, D.; Chowdhury, D.R. Computer Aided Test (CAT) Tool for Mixed Signal SOCs. In Proceedings of the 18th International Conference on VLSI Design (VLSI Design 2005), with the 4th International Conference on Embedded Systems Design, Kolkata, India, 3–7 January 2005; pp. 787–790. https://doi.org/10.1109/ICVD.2005.67.
27. Bahar, R.I. Conference Reports: Recap of the 37th Edition of the International Conference on Computer-Aided Design (ICCAD 2018). *IEEE Des. Test* **2019**, *36*, 98–99. https://doi.org/10.1109/MDAT.2019.2891761.
28. Ramanathan, M.K.; Koyutürk, M.; Grama, A.; Jagannathan, S. PHALANX: a graph-theoretic framework for test case prioritization. In Proceedings of the 2008 ACM Symposium on Applied Computing (SAC), Fortaleza, Brazil, 16–20 March 2008; pp. 667–673. https://doi.org/10.1145/1363686.1363848.
29. Chi, J.; Qu, Y.; Zheng, Q.; Yang, Z.; Jin, W.; Cui, D.; Liu, T. Relation-based test case prioritization for regression testing. *J. Syst. Softw.* **2020**, *163*, 110539. https://doi.org/10.1016/j.jss.2020.110539.
30. Wong, W.E.; Horgan, J.R.; London, S.; Agrawal, H. A study of effective regression testing in practice. In Proceedings of the Eighth International Symposium on Software Reliability Engineering, ISSRE 1997, Albuquerque, NM, USA, 2–5 November 1997; pp. 264–274. https://doi.org/10.1109/ISSRE.1997.630875.
31. Gupta, P.K. K-Step Crossover Method based on Genetic Algorithm for Test Suite Prioritization in Regression Testing. *J. Univers. Comput. Sci.* **2021**, *27*, 170–189. https://doi.org/10.3897/jucs.65241.
32. Chen, J.; Zhu, L.; Chen, T.Y.; Towey, D.; Kuo, F.; Huang, R.; Guo, Y. Test case prioritization for object-oriented software: An adaptive random sequence approach based on clustering. *J. Syst. Softw.* **2018**, *135*, 107–125. https://doi.org/10.1016/j.jss.2017.09.031.
33. Liu, T. *Learning to Rank for Information Retrieval*; Springer: New York, NY, USA, 2011. https://doi.org/10.1007/978-3-642-14267-3.

34. Mirarab, S.; Tahvildari, L. A Prioritization Approach for Software Test Cases Based on Bayesian Networks. In Proceedings of the Fundamental Approaches to Software Engineering, 10th International Conference, FASE 2007, Held as Part of the Joint European Conferences, on Theory and Practice of Software, ETAPS 2007, Braga, Portugal, 24 March–1 April 2007; pp. 276–290. https://doi.org/10.1007/978-3-540-71289-3_22.
35. Lin, C.; Yuan, S.; Intasara, J. A Learning-to-Rank Based Approach for Improving Regression Test Case Prioritization. In Proceedings of the 28th Asia-Pacific Software Engineering Conference, APSEC 2021, Taipei, Taiwan, 6–9 December 2021; IEEE: New York, NY, USA, 2021; pp. 576–577. https://doi.org/10.1109/APSEC53868.2021.00075.
36. Petric, J.; Hall, T.; Bowes, D. How Effectively Is Defective Code Actually Tested?: An Analysis of JUnit Tests in Seven Open Source Systems. In Proceedings of the 14th International Conference on Predictive Models and Data Analytics in Software Engineering, PROMISE 2018, Oulu, Finland, 10 October 2018; pp. 42–51. https://doi.org/10.1145/3273934.3273939.
37. Zou, W.; Xuan, J.; Xie, X.; Chen, Z.; Xu, B. How does code style inconsistency affect pull request integration? An exploratory study on 117 GitHub projects. *Empir. Softw. Eng.* **2019**, *24*, 3871–3903. https://doi.org/10.1007/s10664-019-09720-x.
38. Belli, E.; Vantini, S. Measure Inducing Classification and Regression Trees for Functional Data. *arXiv* **2020**, arXiv:2011.00046v1.
39. Samigulina, G.A.; Samigulina, Z.I. Immune Network Technology on the Basis of Random Forest Algorithm for Computer-Aided Drug Design. In Proceedings of the Bioinformatics and Biomedical Engineering—5th International Work-Conference, IWBBIO 2017, Granada, Spain, 26–28 April 2017; Proceedings Part I; Rojas, I., Guzman, F.M.O., Eds.; Lecture Notes in Computer Science; 2017; Volume 10208, pp. 50–61. https://doi.org/10.1007/978-3-319-56148-6_4.
40. Hamid, L.B.A.; Khairuddin, A.S.M.; Khairuddin, U.; Rosli, N.R.; Mokhtar, N. Texture image classification using improved image enhancement and adaptive SVM. *Signal Image Video Process.* **2022**, *16*, 1587–1594. https://doi.org/10.1007/s11760-021-02113-y.
41. Atik, I. A New CNN-Based Method for Short-Term Forecasting of Electrical Energy Consumption in the Covid-19 Period: The Case of Turkey. *IEEE Access* **2022**, *10*, 22586–22598. https://doi.org/10.1109/ACCESS.2022.3154044.
42. Xie, B.; Zhang, Q. Deep Filtering with DNN, CNN and RNN. *arXiv* **2021**, arXiv:2112.12616v1,
43. Guidotti, D. Verification and Repair of Machine Learning Models. Ph.D. Thesis, University of Genoa, Genoa, Italy, 2022.
44. Gu, Y.; Xuan, J.; Zhang, H.; Zhang, L.; Fan, Q.; Xie, X.; Qian, T. Does the fault reside in a stack trace? Assisting crash localization by predicting crashing fault residence. *J. Syst. Softw.* **2019**, *148*, 88–104. https://doi.org/10.1016/j.jss.2018.11.004.
45. Moraglio, A.; Silva, S.; Krawiec, K.; Machado, P.; Cotta, C. (Eds.) *Genetic Programming, Proceedings of the 15th European Conference, EuroGP 2012, Málaga, Spain, 11–13 April 2012*; Lecture Notes in Computer Science; Springer: New York, NY, USA, 2012; Volume 7244. https://doi.org/10.1007/978-3-642-29139-5.
46. Rothermel, G.; Untch, R.H.; Chu, C.; Harrold, M.J. Prioritizing Test Cases For Regression Testing. *IEEE Trans. Softw. Eng.* **2001**, *27*, 929–948. https://doi.org/10.1109/32.962562.