Research and Application of Automatic Test Case Generation Method Based on User Interface and Business Flow Chart

Lei Xiao, Ru-Xue Bai, Ke-Shou Wu, Rong-Shang Chen*

Department of Computer and Information Engineering, Xiamen University of Technology, China
lxiao@xmut.edu.cn, 2222031181@xmut.edu.cn, kswu@xmut.edu.cn, rschen@xmut.edu.cn

Abstract

Test case design is a critical task in software testing. Manual test case generation is time-consuming and challenging to maintain. To address these issues, this paper proposes a method for automatically generating test cases based on user interface and flowchart analysis. Firstly, YOLOv8 object detection and EasyOCR text recognition are used to identify control information within the interface. Secondly, the Faker library is utilized to generate corresponding test data. Finally, a text generation program is employed to transform control information and test data into a set of interface test cases. Additionally, a circular traversal algorithm is applied to traverse the flowchart, generating test paths that are combined with interface test cases to form a complete set of test cases. To validate the effectiveness of the method, corresponding tools were developed, and 209 test cases were generated for three systems using this approach. Experimental results demonstrate that our proposed method performs well in terms of test case generation efficiency, defect discovery, and maintainability.

Keywords: Test case automatic generation, YOLOv8, Loop traversal algorithm

1 Introduction

Software testing is the process of running software and checking whether it conforms to the expected behavior, aiming to find and correct the defects and errors in software, and is an important means to ensure the quality of software [1]. Test case design is an important and tedious work in software testing, accounting for 40%-70% of the total test workload. Therefore, the automatic generation of test cases has become a research hot spot in the industry. At present, the technology of automatic generation of use cases is mainly through code, model and recording user behavior [2].

Code-based test case generation refers to the use of specific algorithms or tools to automatically generate test cases based on existing code [3]. For example, Liu et al. proposed using code similarity to reuse and generate test cases. This method has a high reuse recall rate and reuse accuracy, and the generated test cases have a higher coverage rate at the same time cost [4]. But it needs to acquire and traverse the code, which has high maintenance costs and is not good for software security. Model-based test case generation refers to the derivation of test cases from the model by building an abstract description of the expected behavior of the software under test. Li and Wong conducted a study to automatically generate test cases from communication extended finite state machine (CEFSM)-based models [5]. Test cases generated address branching coverage not only for data-related decision coverage but also behavioral transition coverage. Belli et al. proposed a model-based mutation testing [6]. Studies show that test cases generated using this approach could detect faults in the systems that were tested by manufacturers and independent testing organizations before they were released. Philip Samue et al. put forward the concept of a sequence dependency graph, which first transforms the sequence graph into a dependency graph, and then generates test cases by traversing the sequence dependency graph [7]. The method of model-based test case generation can produce high coverage according to the system behavior and path specified by the model because the existing model-based generation technology is not flexible enough, the corresponding test cases cannot be modified quickly when the software changes. The method of generating test cases based on recording and playback is an automated testing approach. It involves recording user interactions or test script executions (recording) and then replaying these interactions (playback) to generate and execute test cases [8-10]. This approach is often used for graphical user interface testing. The main tools available for recording and playing back test cases are Selenium, an automated test tool for Web applications; Appium, an automated test tool for mobile applications; AI-Test Ops, an automated test tool for Web, desktop, and mobile applications developed by Dragon-testing Technology. Ltd. Each of these tools provides an easy way to record user interactions and generate corresponding test scripts. However, relying entirely on user actions is not conducive to finding potential errors with low usage, which is mainly used for regression testing.

Test data and test cases play a crucial role in software testing [11-12]. Test data is the input data that validates the function of the system, and test cases are the text used to describe the test steps. However, most of the current test case generation technology mainly focuses on the content and steps of the test case, ignoring the test data, which can

*Corresponding Author: Lei Xiao; E-mail: lxiao@xmut.edu.cn
DOI: https://doi.org/10.70003/160792642025052603009

help to find errors, logical problems, or boundary situations in the code.

Therefore, this paper presents an automatic test case generation method based on interface and flow chart. This method can automatically identify the control type and extract the text information on the control and match the test data according to the text information, so as to automatically generate the interface test case set. At the same time, to ensure the completeness of the automatically generated test case set, a method based on the functional flowchart is designed to automatically generate test cases. The method traverses the flowchart to obtain basic test paths and matches the node information in the basic paths with their corresponding interface test cases, thereby obtaining a complete set of test cases combining interface (static) and functionality (dynamic). The method divides the process of test case generation into interface generation process and flow chart generation process and solves the problem of high frequency of interface change and difficult maintenance in the process of test automation. When the interface changes, simply rescanning the interface will automatically update the corresponding interface and process test cases, effectively adapting to the characteristic of frequent interface changes in agile development mode and enhancing the maintainability and scalability of the test case suite. At the same time, with the help of the Faker library for secondary development, automatically generated test data includes both positive and negative data, ensuring that the automatically generated test case suite can comprehensively uncover potential defects in the software, thereby enhancing test security and completeness.

2 Related Work

Automatic generation of model-based test cases is an important method, including traditional model-based methods and new model-based methods [13]. UML model diagram is one of the main models of application test case generation. In 2018, based on UML sequence diagrams, Li et al. transformed the sequence diagram into a sequential directed graph, utilized a genetic ant colony algorithm for optimization calculation, and generated test cases [14]. Wang et al. proposed a test path generation method based on UML state diagrams, primarily targeting the testing of train control centers [15]. The method involved utilizing a depth-first search algorithm to traverse the directed graph and obtain a set of test paths. Model-based generation technology can generate high coverage according to the system behavior and path specified by the model [16-17]. In 2020, Zhang et al. proposed a software system test method based on a formal model by combining an example of a measurement and control information application software system, which can automatically generate test case sets according to the system's operation flow chart, stage analysis diagram, establishment of scene tree model and scene tree diagram [18]. However, when there is an inconsistency between the model and the actual system, the generated test cases cannot accurately reflect the behavior of the actual system.

Page-based test case generation is an automated test method that generates test cases by analyzing the user interface (page) of an application. Wang et al. proposed a Web application test case generation method based on Selenium page object design pattern and graph traversal algorithm. This method starts from the page and improves the coverage of the page and the maintainability of the test cases [19]. This method can quickly generate test cases and reduce labor costs, but the generated test cases may only cover the visible functions and elements on the page and cannot cover some hidden or complex logic.

The generation of test cases based on recording and playback refers to the generation of corresponding test cases through the acquisition of user operation data [20]. In 2018, Hou et al. proposed a cross-device UI automatic testing method based on control paths, which solved the issue of finding controls across devices [21]. In 2020, Zhang et al. used deep learning object detection to improve the accuracy of control identification [22]. The test case generation technology of recording and playback can quickly create test cases, but it comes with high maintenance costs for recorded scripts and the inability to cover some special cases or boundary conditions [23].

Test case generation technology based on deep learning uses deep learning techniques such as neural networks to learn patterns from existing software code or test data and then generate new test cases [24-30]. In 2022, Zubair Khaliq et al. used EfficientDet along with GPT-5 and T5 algorithms to automatically generate test cases by combining text and image information [28]. In a study of 30 e-commerce applications, this method achieved a high correctness rate of 93.82% for generated test cases, while eliminating 96.05% of the brittleness, showing significant effectiveness.

To sum up, most of the existing test case generation technologies focus on the generation of test steps, ignoring the automatic generation of test data. Forward test data can help verify functional correctness, and reverse test data can help find defects and problems [31]. At the same time, there is a lack of research on the maintainability of test cases. In this paper, a method of automatically generating test cases based on interface and flow chart and using Faker library to generate test data is proposed. It is expected to improve the efficiency and maintainability of the test.

3 Test Case Generation

This section describes the method proposed in this paper. Figure 1 illustrates the framework for automatically generating test cases. The framework divides the process of generating test cases into two main stages: interface test case generation process and test path generation process.

The process of automatic generation of test cases can be divided into seven stages. First, the control on the interface is identified using object detection technology YOLOv8 [32]. Second, text recognition technology EasyOCR is used to identify the text on the control. Third, use Faker library to generate test data; fourth, use a text generator to generate interface test cases. Fifth, the process traversal algorithm is used to traverse the flow chart to generate the process test path. Sixth, according to the process node information, find the interface test case generated in step 4 and combine the interface test case and test path into a complete test case. See Figure 1 for details.

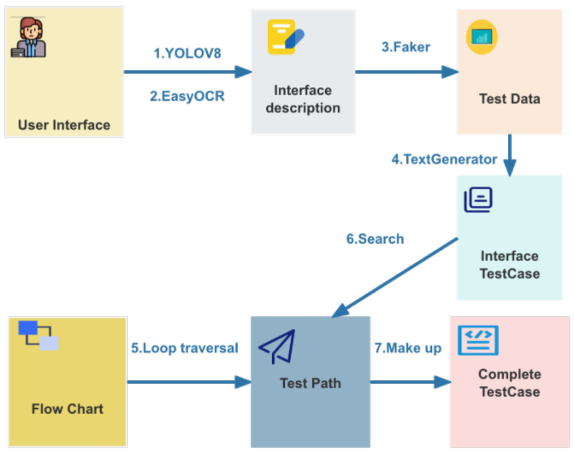


Figure 1. Technical framework

3.1 Interface Test Case Generation

Interface test case generation mainly includes identifying controls, matching test data and generating text test cases. This paper automatically generates test cases from the perspective of interface and strives to achieve a higher coverage of interface functions and reduce the labor cost in designing test cases.

3.1.1 Identification Control

YOLOv8 is the latest series of YOLO based on the object detection model launched by Ultralytics company in January 2023, which not only has the characteristics of high precision and high speed but also supports image classification, object detection and instance segmentation tasks. As a popular object detection technology, YOLO is mainly applied in dynamic object detection, such as mask recognition, license plate recognition, and so on. There are few trained, open source YOLOv8 models available for Web control recognition. Our innovation lies in the collection of a Web control library, training control detection model and the second development of YOLOv8 source code, and text recognition engine EasyOCR weaving into the YOLOv8 output function. At the same time, within the output function, using the Euclidean distance formula to calculate the distance of the control, in order to find the adjacent control of the current control.

We select 8 kinds of controls frequently used in the Web system for identification, which are text box, drop-down box, radio box, check box, button, label, file upload and switch button. The types of controls can be seen in Table 1. The dataset consists of 1267 clear interface images collected from commercial and open-source systems, annotated with 8 types of frequently used component types, with a total annotation count of tens of thousands. The training set and validation set were randomly sampled in a ratio of 4: 1. By evaluating accuracy and recognition speed, YOLOv8n. pt was selected as the pre-trained weight with an initial learning rate of 0. 000455, a maximum of 100 iterations, and a batch size of 8. The model achieves a component recognition accuracy of 92. 4% on the test set after training.

Table 1. Control class list

Control name	Implication	Action
Text box	Used to receive user input text information.	Input
Drop-down box	Used to provide a set of options for customers to choose from.	Select
Radio box	Used to provide multiple options for the user to choose from, but only one can be selected.	Select
Check box	Used to provide multiple options for the user to choose from, but multiple options can be selected.	Select
Button	Used to trigger an operation.	Click
Label	Used to identify the names or descriptions of other controls.	
File upload	Used to select and upload files.	Upload
Switch button	Used to switch between different options or states.	Select

After the interface is tested by the trained and modified YOLOv8 model, the following information can be output:

- Current Control
- Control Text
- X-axis Coordinates
- Y-axis Coordinates
- Adjacent Control

Based on this information, test data will be matched and test cases will be generated.

3.1.2 Match Test Data

Test data is input values used in the software testing process to verify whether the functionality and performance of the software system meet expectations. By designing and preparing different types and combinations of test data, it is possible to help discover potential defects or issues [33-34]. Test data should cover various scenarios, including normal cases, boundary cases, exceptional cases, and so on. This section elaborates on the methods of test data generation.

The test data generation method utilizes the Faker library, NLTK library, and the Chinese WordNet corpus. The Faker library is a Python library used to generate fake data, which can assist developers in quickly generating random data of various types such as names, addresses, emails, text, numbers, etc., during development and testing processes. NLTK is a Python library for natural language processing and is used for tasks such as text data handling, tokenization, part-of-speech tagging, named entity recognition, syntactic analysis, and other natural language processing tasks. The Chinese WordNet corpus is a database for semantic relations and lexical networks of Chinese vocabulary, similar to WordNet in English. It aims to establish a database of semantic relations, including synonyms, antonyms, hypernyms, and other semantic relationships for Chinese vocabulary. The function of viewing synonyms for Chinese words can be realized by combining NLTK and Chinese WordNet library.

The equivalence partitioning method is a common black-box testing technique that advocates dividing input data into different equivalence classes to reduce the number of test cases while still effectively covering the scenarios of each equivalence class. Equivalence classes can be divided into valid equivalence classes and invalid equivalence classes. Valid equivalence classes consist of input data that meets the program requirements and is both reasonable and meaningful. Invalid equivalence classes consist of input data that does not meet the program requirements, is unreasonable, or lacks meaningfulness. The existing Faker library only provides test data for valid equivalence classes. In order to improve the coverage of equivalence classes, we are conducting secondary development on Faker to add a method for generating test data for invalid equivalence classes. In addition, to match test data for more fields, we examine management systems from seven different domains and obtain common fields including username, phone number, email, password, etc. We use NLTK and the Chinese WordNet library to find synonyms for these common fields, in order to build a general-purpose lexicon. We provide a set of methods to obtain test data for both valid and invalid equivalence classes for the words in the general-purpose lexicon. When the control text contains words from the general-purpose lexicon, it will automatically match corresponding positive and negative test data (where valid equivalence classes correspond to positive test data and invalid equivalence classes correspond to negative test data).

The matching steps for test data consist of a total of 4 steps. First, obtain the control text. Next, determine if the control text contains fields from the general-purpose lexicon. If it does, proceed to the next step; if not, end the process. Furthermore, query the corresponding test data method based on the control text. Finally, the corresponding method is called to obtain the forward and reverse test data. The process for generating test data is shown in Figure 2.

In addition, through the investigation of multiple management systems, we found that the Web system frequently used controls mainly include text box, drop-down box, button and other 8 big controls; each control has a fixed action, as shown in Table 1.

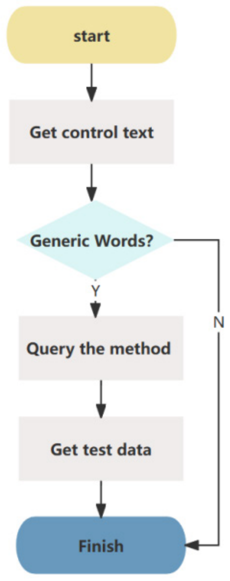


Figure 2. Match test data

3.1.3 Generate Text Test Cases

The main task of generating text test cases is to convert the interface description table processed by YOLOv8, EasyOCR, and Faker libraries into text test cases. When we write test case generation code, we consider the interface coverage, that is, the generated test cases need to cover not only all controls on the interface but also different combinations of control operations [35]. For example, the supplier information in Figure 3 is not a visible mandatory field; it requires clicking the Add Supplier button to display. At the same time, the buttons at the bottom of the interface are usually buttons for completing the current page operation, such as the submit and cancel buttons at the bottom of the interface. Based on this characteristic, considering the interface coverage, this paper decides to use a two-dimensional array structure to store the control element information of the interface. According to the interface, store the control elements from top to bottom and left to right; the storage structure is shown in Figure 4. Each set of elements in the storage structure will store the current control, control text, X-axis coordinate, Y-axis coordinate, and information of adjacent controls, as detailed in Figure 5.

As mentioned above, the type of control determines the corresponding operation. The label control is used to indicate information about the field that is currently being maintained, depending on the type of control that follows the label control. The buttons at the bottom of the interface are usually used as navigation buttons to end the current page action, while the top and middle buttons are usually used as buttons to maintain optional information on this interface. Therefore, when a test case is generated, the bottom button can be used as a termination condition for generating a test case. Based on these two features, we write code for generating test cases.

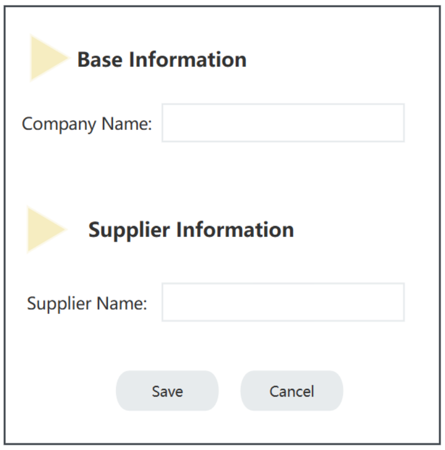


Figure 3. Interface prototype

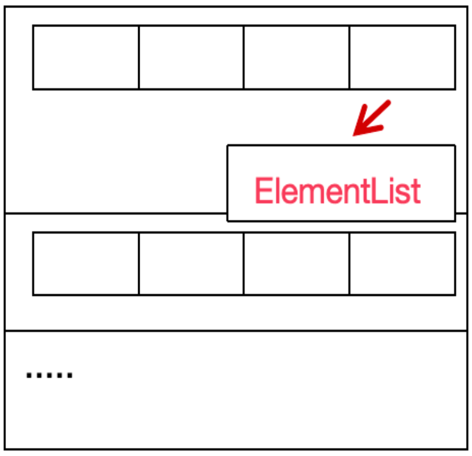


Figure 4. Control element structure

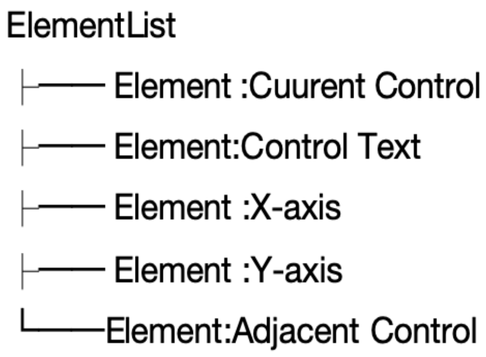


Figure 5. Element list structure

3.2 Process Test Path Generation

A flowchart is a graphical tool used to illustrate the flow, process, or system, enabling a clear depiction of the business steps within a system. It is widely utilized for business process modeling in software systems. The method of generating test cases based on flowcharts is a common software testing technique aimed at designing test cases by analyzing the control flow graph of a program. Path coverage is a software testing method designed to ensure that every possible execution path of a program or process is executed at least once. Path coverage is also an important indicator to measure the effectiveness of flowchart-based test case generation methods.

Depth First Search (DFS) is a traversal algorithm used for graphs or tree structures. Its traversal starts from a starting vertex and explores the graph's nodes as deeply as possible along a single path until reaching a leaf node. It then backtracks and continues to explore the next path. Breadth First Search (BFS) is also a traversal algorithm used for graphs or tree structures. The idea is to start from the initial vertex, visit all neighboring nodes of the initial vertex first, then visit the neighbors of these neighbors one by one, and so on, until the entire graph or tree is traversed. We find that using DFS and BFS algorithms to traverse the Figure 6 flowchart does not fully cover all paths in the flowchart. Therefore, we propose a cyclic traversal algorithm that combines the breadth-first search algorithm with the double-ended queue deque. By using the cyclic traversal algorithm to traverse the flowchart, path coverage in the flowchart can be achieved. Table 2 lists the traversal results of the three algorithms for Figure 6.

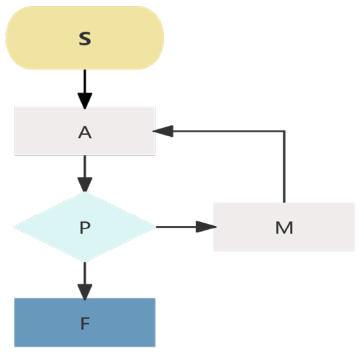


Figure 6. Flow chart

Table 2. Flow chart traverses

Method	Ergodic result
DFS	[S, A, P, M, F]
BFS	[S, A, P, M, F]
Loop	[S, A, P, F]
Traversal algorithm	[S, A, P, M, A, P, F]

3.2.1 Ergodic Flow Chart

Double-ended queue (deque) is a data structure provided in Python by the collections module. It is a special type of queue that allows insertion and deletion operations at both ends simultaneously, as shown in Figure 7. Based on this characteristic, the cyclic traversal algorithm chooses deque as the storage structure. Each item in the deque stores two pieces of information: the value of the current node and the current traversal path, as shown in Figure 8.

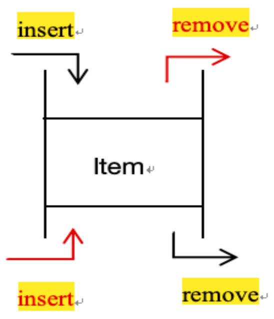


Figure 7. Deque structure

Item
├Element:Current Node
└Element:Ergonomic Path

Figure 8. Item structure

The idea of the loop traversal algorithm is as follows. First, obtain the information of the start node, the end node, the other nodes and the adjacent nodes of each node in the flow chart. Second, initialize the starting node and the initial path; place the initial node and initial path into the deque. Third, take out the first node from the top of the deque as the new node, find the adjacent nodes of the new node based on its value, construct a new path from the starting point to the adjacent node, and then insert the new node and the new path at the bottom of the deque. Fourth, repeat step three. When the first node of the constructed path is the starting node and the last node is the end node, output the path; when the deque is empty, end the loop. It is important to note that the flowchart may have branches, meaning a node may have multiple adjacent nodes. Therefore, in step 3, when finding the adjacent nodes of the current node, a loop should be used. Additionally, keep track of the number of occurrences of each node in the current path. If a node appears more than twice, it indicates that the current path has been taken before, so skip it and look for the next path. For more details, refer to Algorithm 1.

Algorithm 1. Loop traversal algorithm

```
Input: Flowchart information, start node, end node
Output: Flow path
1.   def circularTraversal(graph, start, end):
2.     queue = deque([(start, [start])])
3.     while queue:
         #Retrieves nodes and paths from queue
4.       current_node, path = queue. popleft())
5.       for neighbor in graph[current_node]:
6.            npath = path + [neighbor]
              #Count the number of traversed nodes
7.            counter = Counter(npath)
8.            flag = True
9.            for cout in counter:
10.             if counter[cout] > 2:
11.               flag = False
12.           if flag:
13.             if npath[0] == start and npath[-1]==end:
                       #Output the full test path
14.               yield npath
            #Put a new node and path into deque
15.         queue. append((neighbor, npath))
```

In line 2, place the start node and start path into the deque. In line 4, take the node and path from the top (left) of the deque. On line 3, determine if the deque is empty and terminate the loop if it is. Lines 5-6 construct a new path by finding the next node based on the node. Lines 7-11 count the number of nodes passed by the new path. If the number is more than 2 times, skip it; otherwise, continue the following steps. Lines 13-14 determine whether the first node and the last node of the new path are the start node and end node, and output the new path if so. In line 15, insert a new node and path at the end of the deque (to the right).

3.2.2 Test Case Consolidation and Updating

Test case maintainability refers to the degree to which test cases are easy to understand, modify, and update. A well-maintainable test case can be quickly adjusted when the system changes without causing additional trouble or errors. Nowadays, most software companies adopt the agile development model in order to respond quickly to market demands, reduce project risks, and enhance the market competitiveness of software products. In the agile development model, software systems may undergo frequent changes. Therefore, test cases need to be highly maintainable in order to adapt to these changes and remain effective.

To improve the maintainability of test cases, we divide the test case generation process into interface test case generation and flow test path generation. The flow consists of functionalities that can be demonstrated by the interface. Separating the interface and the flow in this way not only increases the independence of the interface and the flow but also enhances the maintainability of test cases.

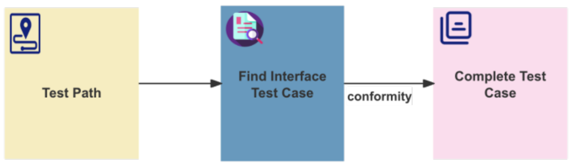


Figure 9. Test case integration

Figure 9 illustrates the integration process of test cases. Firstly, obtain the test path and search for the corresponding interface test case in the interface case library based on the information of the path nodes. Secondly, after finding the corresponding interface test case, integrate it into the test path to form a complete test case.

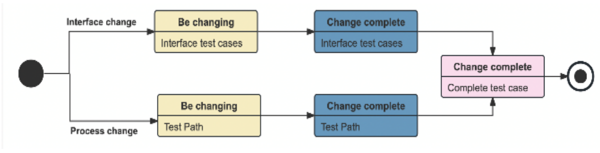


Figure 10. Test case linkage update

Figure 10 illustrates the state transition diagram for test case linkage updates. Firstly, when the interface undergoes changes, but the business process remains unchanged, users can update the interface test cases by uploading a new process diagram. At this point, the interface test cases are in a state of modification. Once the interface test cases have been updated, the system will automatically complete the test case modification by combining the original test path with the updated interface test cases. Secondly, when the business process changes but the interface remains unchanged, users can update the process path by entering new process information. The test path is in a state of modification. Once the test path modification is complete, the system will automatically complete the test case modification by combining the new test path with the original interface test cases.

4 Experimental Design

4.1 Research Questions

In our experiment, we propose the following three research questions:

RQ1: How effective is the method we propose for automatically generating test cases based on interface and flowchart?

While the method for automatically generating test cases has been described earlier, it is also crucial to validate the feasibility and effectiveness of this approach through experiments.

RQ2: Can the method we propose improve the maintainability of test cases?

Researchers propose many methods for generating test cases, but most of them lack discussions on the maintainability of test cases. With software requirements changing rapidly, corresponding test cases need to be able to respond to changes quickly. Therefore, we hope to investigate how well the proposed method performs in test case maintainability.

RQ3: How does our proposed approach perform in defect discovery rates?

Effective test cases should help improve defect detection rates; therefore, validating the performance of the proposed method in defect detection rates is also one of our tasks.

4.2 Experimental Object

In 2023, we collaborated with an evaluation company and undertook the testing work for a Kexin document management system. We collected system interface diagrams, business process diagrams, and test cases for the experiment. The Kexin document management system is a system used for managing and storing files, including modules for file cataloging and file form management. At the same time, to validate the universality of the method, we also selected the open-source Ruoyi management system and a student-developed Suixing management system. The Ruoyi management system is a basic management system. management system. The Suixing management system is a backend system used to manage online car-hailing services. We selected the core modules of these three systems for experimentation. Table 3 presents the specific object information for this experiment.

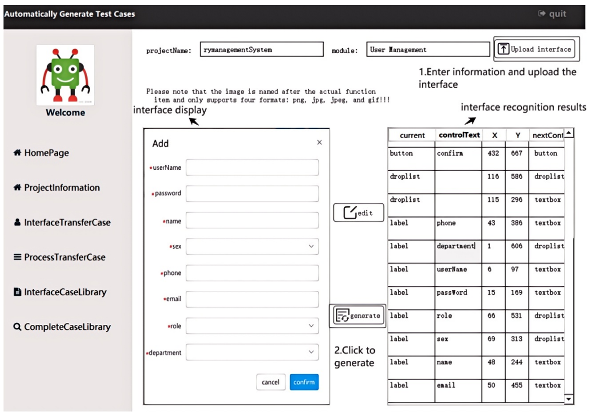
Table 3. Experimental object

System name	System source	Function module
Ruoyi	Open source	User management
		Parameter management
		Announcement management
		Dictionary management
		Role management
		Menu management
		Position management
		Department management
Kexin	Industry	Document-acquisition management
		File type management
		File directory management
		File category management
		Document-destruction management
Suixing	Science	User management
		Fine management
		Order receiving management
		Voucher management

4.3 Experimental Tool

For this experiment, we make an automatic test case generation tool according to the method proposed in this paper and use this tool to generate 209 test cases for the three systems. This section shows how the tool generates test cases.

The process of generating interface test cases based on the interface is as follows. Firstly, input the project name and functional module; secondly, upload the interface prototype diagram, and the system automatically recognizes the interface and outputs interface element information. Finally, click "Generate", and the system can automatically generate corresponding interface test cases. Figure 11 shows the operation interface of the interface diagram to the test case. Figure 12 shows the detailed interface of the system-generated interface test cases.



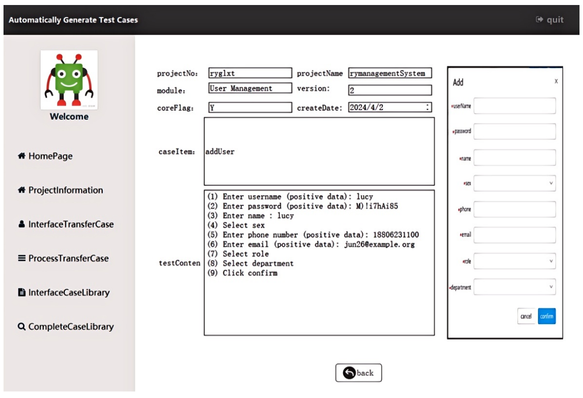
projectName: rymanagementSystem module: User Management

Please note that the image is named after the actual function item and only supports four formats: png, jpg, jpeg, and gif!!!

*userName
*password
*name
*sex
*phone
*email
*role
*department

current	controlText	X	Y	nextCont
button	confirm	432	667	button
droplist		116	586	droplist
droplist		115	296	textbox
label	phone	43	386	textbox
label	department	1	606	droplist
label	userName	6	97	textbox
label	passWord	15	169	textbox
label	role	66	531	droplist
label	sex	69	313	droplist
label	name	48	244	textbox
label	email	50	455	textbox

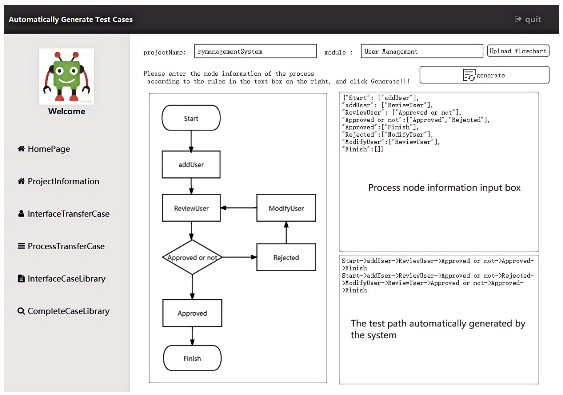
Figure 11. Interface case generation interface



projectNo: ryglxt projectName rymanagementSystem
module: User Management version: 2
coreFlag: Y createDate: 2024/4/2
caseItem: addUser
testConten
```
(1) Enter username (positive data): lucy
(2) Enter password (positive data): M)!i7hAi85
(3) Enter name : lucy
(4) Select sex
(5) Enter phone number (positive data): 18806231100
(6) Enter email (positive data): jun26@example.org
(7) Select role
(8) Select department
(9) Click confirm
```

Figure 12. Interface case display interface

The process of generating test paths based on the flow is as follows. First, enter the project name and functional module; next, upload the flowchart and input the flow node information; finally, click "Generate", and the system can output test paths and complete test cases. Figure 13 shows the interface for generating test paths. Figure 14 presents the detailed interface of the generated complete test cases.



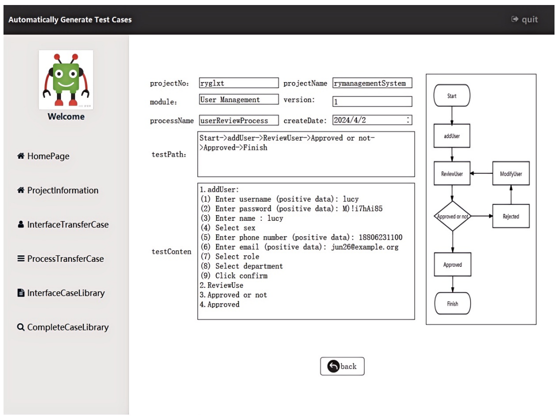
projectName: rymanagementSystem module : User Management

Please enter the node information of the process according to the rules in the text box on the right, and click Generate!!!

```
["Start": ["addUser"],
"addUser": ["ReviewUser"],
"ReviewUser": ["Approved or not"],
"Approved or not":["Approved","Rejected"],
"Approved":["Finish"],
"Rejected":["ModifyUser"],
"ModifyUser":["ReviewUser"],
"Finish":[]]
```

```
Start->addUser->ReviewUser->Approved or not->Approved-
>Finish
Start->addUser->ReviewUser->Approved or not->Rejected-
>ModifyUser->ReviewUser->Approved or not->Approved-
>Finish
```

Figure 13. Test path generation interface



projectNo: ryglxt projectName rymanagementSystem
module: User Management version: 1
processName userReviewProcess createDate: 2024/4/2
testPath:
```
Start->addUser->ReviewUser->Approved or not-
>Approved->Finish
```
testConten
```
1. addUser:
(1) Enter username (positive data): lucy
(2) Enter password (positive data): M)!i7hAi85
(3) Enter name : lucy
(4) Select sex
(5) Enter phone number (positive data): 18806231100
(6) Enter email (positive data): jun26@example.org
(7) Select role
(8) Select department
(9) Click confirm
2. ReviewUse
3. Approved or not
4. Approved
```

Figure 14. Complete case display interface

4.4 Evaluation Index

In order to evaluate the proposed method of automatically generating test cases, the following evaluation criteria are defined in this paper.

Accuracy rate: To evaluate the quality of generated test cases is used. The calculation formula is shown in formula 1.

$$AR = (CUCN \div TUCN) \times 100\% \tag{1}$$

AR represents Accuracy rate, CUCN (Correct Use Case Number) represents the number of correct use cases, and TUCN (Total Use Case Number) refers to the total number of use cases.

Interface coverage rate: To evaluate whether the generated test cases cover all elements on the interface. The calculation formula is shown in formula 2.

$$ICR = (OEC \div TEC) \times 100\% \tag{2}$$

ICR (Interface Coverage Rate) represents the interface coverage rate, OEC (Overall Element Coverage) stands for the number of covered elements in test cases, and TEC (Total Elements Count) refers to the total number of elements in the interface.

Path coverage rate: To evaluate whether the generated test paths based on the flow cover all paths in the flowchart. The calculation formula is shown in formula 3.

$$PCR = (OPN \div TPN) \times 100\% \tag{3}$$

PCR represents path coverage rate. OPN (Observed Path Number) represents the number of traversed paths, and TPN (Total Path Number) refers to the total number of paths.

Functional coverage rate: To evaluate whether the test cases cover every functionality point in the software system. The calculation formula is shown in formula 4.

$$FCR = (OFN \div TF) \times 100\% \tag{4}$$

FCR represents the functional coverage rate, OFN (Observed Functional Number) stands for the number of covered functions, and TF (Total Functions) refers to the total number of functions.

Defect discovery rate: To evaluate the quality of testing and the quality of the software. The calculation formula is shown in formula 5.

$$DDR = (DUCN \div TUCN) \times 100\% \tag{5}$$

DDR represents the defect discovery rate, DUCN (defect use cases number) represents the number of use cases where defects were found, and TUCN (total use cases number) represents the total number of use cases.

5 Results and Analysis

5.1 Answering Research Question 1

RQ1: How effective is the method we propose for automatically generating test cases based on interface and flowchart?

We generate test cases for three systems and simultaneously calculate the accuracy and coverage of the generated test cases. Accuracy reflects the quality of the generated test cases, while coverage reflects the extent to which the generated test cases cover the system. The higher the accuracy and coverage, the more effective the generated test cases are. The experimental results are shown in Table 4.

From Table 4, it can be seen that the test cases generated using this method perform well in terms of accuracy and coverage. The accuracy of the Ruoyi system is higher than that of the other two systems because the input prototype diagram of the Ruyi system is clearer, and there is a significant style difference between different types of controls, making it easier to identify the model. Meanwhile, the functional coverage of the Ruoyi management system is higher than that of the other two systems. This is because the Ruoyi system's interface diagram and flowchart cover more functionalities, while functionalities not covered by the interface and flowchart, such as background functions like timers, are fewer. Through experiments, we find that the method proposed in this paper is influenced by the type, size, and clarity of the input interface, as well as the pre-training data set of YOLOv8 for object detection. The more accurate and diverse the annotated dataset, and the larger and clearer the input interface image, the better the recognition effect, resulting in higher accuracy.

Table 4. Validity test result

System name	Screen quantity	Interface test cases number	ICR (%)	Business flow number	Test path number	PCR (%)	FCR (%)	DDR (%)
Ruoyi	16	94	100	9	15	100	96. 92	**17. 43**
Kexin	10	57	95. 26	5	7	100	93. 54	**39. 06**
Suixing	8	28	94. 95	5	8	100	94. 74	**22. 58**

Table 5. Maintainability comparison results

System name	Variable interface number	Variable flow Number	Functions involved number	Processes involved number	Cases change number		Spend time (min)	
					Labour	Our	Labour	Our
Ruoyi	3	1	3	4	10	7	26. 3	2. 73
Kexin	2	2	2	2	17	9	18. 58	2. 15
Suixing	2	2	2	4	14	12	12	3. 08

5.2 Answering Research Question 2

RQ2: Can the method we propose improve the maintainability of test cases?

Due to the frequent changes that occur in software systems during development and evolution, test cases need to have good maintainability. To assess the maintainability of generated test cases, we selected modules that underwent changes during the testing process in three systems, re-maintained the test cases, and calculated the manual cost of re-maintenance and the number of test cases changed. Meanwhile, we also compare the proposed method with the traditional manual test case generation approach. The comparison results are shown in Table 5.

From Table 5, it can be concluded that our proposed method is better suited to adapt to system changes compared to the manual approach. Not only are there fewer changes in test cases, but also the incurred manual costs are lower. This is primarily due to the following two reasons. Firstly, we divide the test cases into interface and flow, separating the interface from the flow. When either the interface or the flow changes, it does not affect the other. Secondly, we automate the process of updating test cases. When there is a change in the interface, users only need to upload the new interface diagram, and the system can automatically update the original interface test cases. Similarly, when there is a change in the flow, users only need to upload the new flowchart and input the new flow information to automatically update the original test paths. Additionally, complete test cases consist of both test flow paths and interface test cases. Hence, when there are changes in either the interface test cases or the test paths, the complete test cases will also be automatically updated.

5.3 Answering Research Question 3

RQ3: How does our proposed approach perform in defect discovery rates?

The defect detection rate is an important metric for evaluating the quality of testing and software, which can assess the level of risk in software development. A high defect detection rate may indicate significant issues within the software, while a low defect detection rate may be due to two main reasons. Firstly, the generated test cases may miss certain functionalities; secondly, the software has undergone several rounds of testing and corrections, resulting in fewer existing defects. As the systems selected for this study have all undergone internal testing and have few existing defects, we calculated both the functional coverage of the generated test cases and the defect detection rate during the experiment. Furthermore, to validate the feasibility of the proposed method, we compared it with the method of generating test cases using ChatGPT3.5, a natural language processing model developed by Open AI capable of generating dialogues, answering questions, providing suggestions, and more. In the comparative experiment, we inputted the requirement documents into ChatGPT3.5 allowing it to automatically generate test cases based on the requirements documents. The comparative results are shown in Table 6. Through experiments on defect detection rates, we find that the discovery rate of defects is influenced by various factors. Among them, there exists a certain correlation between functional coverage rate and defect detection rate.

Table 6. Comparison of defect discovery rate

System name	Generate use cases number		Functional coverage rate(%)		Defect discovery rate(%)	
	ChatGPT3.5	our	ChatGPT3.5	our	ChatGPT3.5	our
Ruoyi	70	109	81. 53	96. 92	8. 57	17. 43
Kexin	43	64	74. 19	93. 54	18. 6	39. 06
Suixing	31	36	88. 88	94. 74	9. 67	22. 58

Generally, the higher the functional coverage rate, the more defects may be discovered. As shown in Table 6, the functional coverage rate and defect detection rate of the test cases generated in this study are higher than those generated by Chat ChatGPT3.5. There are mainly two reasons for this. First, due to oral requirements changes during development that are not synchronized with the requirement documents, and the issue of insufficiently detailed requirement document writing, the actual developed functionalities may not align with the requirement documents. As a result, the functional coverage rate of the test cases generated by ChatGPT3.5 based on the requirement documents is lower, thereby affecting the defect detection rate. Second, our proposed method utilizes the Faker library for secondary development to generate test cases containing both valid and invalid equivalence class test data. In contrast, the test data generated by ChatGPT3.5 only includes positive test data, neglecting negative test data. However, negative test data often helps discover various defects when the system encounters abnormal or unexpected inputs.

6 Validity Threats Discussion

This section discusses experimental validity and existential threats

Internal validity: We utilized the same tool to generate test cases for all three systems to ensure consistency in experimental operations and employed identical metrics to evaluate experimental outcomes.

External validity: Variations may exist in experimental results across systems with different styles and business domains; hence, we selected three systems representing diverse styles and businesses for the experiment. The proposed method exhibited satisfactory accuracy and coverage across the three systems, suggesting its potential generalizability.

Threat: Due to constraints imposed by the controlled dataset and text-generated code, the proposed method can only identify 8 common controls on the interface, failing to encompass other controls within the interface. Additionally, the method, starting from the interface and flowcharts, is unable to generate test cases for hidden functionalities without interface representation or covered by the flow.

7 Conclusion

In this paper, an automatic test case generation method based on an interface and flow chart is proposed to reduce labor costs and improve test efficiency. This method integrates YOLOv8 and other object detection technology, EasyOCR text recognition technology and cyclic traversal algorithm. To our knowledge, this is the first way to separate processes and interfaces to enhance test case maintainability. In addition, it is the first to leverage the Faker library to generate positive and negative test data, enhancing the comprehensiveness of test cases. On this basis, we developed a tool to automatically generate test cases and used it to generate 209 test cases for 17 functional modules of three web systems. We find that the test cases generated by our method can improve the test efficiency. Compared to ChatGPT3.5, our approach achieved an average of 13.53% improvement in functional coverage and 14.08% improvement in defect detection. However, due to the influence of experimental data sets, there is still room for improvement in the generality and accuracy of our method. Therefore, in the future, we will collect more UI interface diagrams for training to enhance the universality of the control recognition model. We will also conduct more experiments for different systems to generate test cases, aiming to identify and solve problems, optimize code, and improve the efficiency and accuracy of automated test case generation procedures.

Acknowledgment

This work is supported by Natural Science Foundation of Fujian Province under grant No. 2022J011238), Xiamen Major Public Technology Service Platform under grant No. 3502Z20231042), Research on Key Technologies of Intelligent and Cloud Service Testing (2022 Central Government Guide Local Development Science and Technology Special Project) under grant No. 2022L3029, Research on regression test optimization method based on historical execution information in continuous integration environment under grant No. 303021003.

References

[1] C. E. Lai, C. Y. Huang, Developing a Modified Fuzzy-GE Algorithm for Enhanced Test Suite Reduction Effectiveness, *International Journal of Performability Engineering*, Vol. 19, No. 4, pp. 223-233, April, 2023. https://doi.org/10.23940/ijpe.23.04.p1.223233

[2] X. L. Chen, A Review of Test Case Automatic Generation Based on UML Models, *Modern Computer*, No. 7, pp. 61-65, March, 2018.

[3] J. Wen, *Research on automatic Generation of Test Cases based on code block coverage*, M.D. Thesis, Xi'an University of Technology, Xi'an, China, 2017. https://doi.org/10.27398/d.cnki.gxalu.2017.000016

[4] Q. Y. Liu, Q. H. Yang, M. Hong, M. Y. Liu, Y. Y. Liu, Test case reuse and generation method based on code similarity, *Computer Engineering and Design*, Vol. 44, No. 10, pp. 2950-2955, October, 2023. https://doi.org/10.16208/j.issn1000-7024.2023.10.010

[5] J. J. Li, W. E. Wong, Automatic test generation from communicating extended finite state machine (CEFSM)-based models, *Proceedings Fifth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing. ISIRC 2002*, Washington, DC, USA, 2002, pp. 181-185. https://doi.org/10.1109/ISORC.2002.1003693

[6] F. Belli, C. J. Budnik, A. Hollmann, T. Tuglular, W. E. Wong, Model-based mutation testing—approach and case studies, *Science of Computer Programming*, Vol. 120, pp. 25-48, May, 2016. https://doi.org/10.1016/j.scico.2016.01.003

[7] P. Samuel, A. T. Joseph, Test Sequence Generation from UML Sequence Diagrams, *Ninth ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing*, Phuket, Thailand, 2008, pp. 879-887. https://doi.org/10.1109/SNPD.2008.100

[8] H. Zhao, W. L. Gao, A Comparison of Methods for Software Test Case Generation Based on Model, *Modern Computer*, No. 4, pp. 20-26, February, 2017. https://doi.org/10.3969/j.issn.1007-1423.2017.04.005

[9] R. Pan, *Behavior Analysis in Scripting of Web Function Test Based on Selenium Script*, M. D. Thesis, Nanjing University of Posts and Telecommunications, Nanjing, China, 2021. https://doi.org/10.27251/d.cnki.gnjdc.2021.000741

[10] Y. F. Hou, *Research on Selenium Automated Test Framework Based on User/Browser Pool Task Scheduling Method*, M. D. Thesis, Beijing Jiaotong University, Beijing, China, 2021. https://doi.org/10.26944/d.cnki.gbfju.2021.000462

[11] D. H. Gao, *Study of Test Data Generation Based on SVR-MChOA Algorithm*, M. D. Thesis, Qingdao University of Science and Technology, Qingdao, China, 2023. https://doi.org/10.27264/d.cnki.gqdhc.2023.000718

[12] W. Z. Liao, X. Y. Xia, X. J. Jia, Test Data Generation for Multiple Paths Coverage Based on Ant Colony Algorithm, *Acta Electronica Sinica*, Vol. 48, No. 7, pp. 1330-1342, July, 2020. https://doi.org/10.3969/j.issn.0372-2112.2020.07.011

[13] Q. L. Pu, Y. P. Wang, T. Liu, Y. Sun, J. X. Li, Use Case Generation Method Based on Model Testing, *Computer Measurement and Control*, Vol. 29, No. 12, pp. 22-26, December, 2021. https://doi.org/10.16526/j.cnki.11-4762/tp.2021.12.005

[14] Y. M. Li, *Research on test case generation algorithm based on sequence diagram and construction of automated test platform*, M. D. Thesis, Beijing Jiaotong University, Beijing, China, 2018. https://kns.cnki.net/KCMS/detail/detail.aspx?dbname=CMFD201802&filename=1018104264.nh

[15] X. X. Wang, Method of software test paths generation for train control center based on UML state chart diagram, *Railway Computer Application*, Vol. 25, No. 8, pp. 9-12+15, August, 2016. https://doi.org/10.3969/j.issn.1005-8451.2016.08.003

[16] L. X. Yang, *Research and Application of Model Driven Web Automated Testing Platform*, M. D. Thesis, Southwest University of Science and Technology, Sichuan, China, 2022. https://doi.org/10.27415/d.cnki.gxngc.2022.000975

[17] X. H. Gao, *The Research of GUI Test Cases Generation Based on Models*, M. D. Thesis, Shanghai Normal University, Shanghai, China, 2014. https://kns.cnki.net/KCMS/detail/detail.aspx?dbname=CMFD201501&filename=1014334011.nh

[18] W. X. Zhang, M. Zhang, Z. H. Dou, M. Y. Ma, B. Wei, Aerospace Application Software Testing Method Based on Process and Scenario Analysis, *Measurement & Control Technology*, Vol. 39, No. 1, pp. 30-35, January, 2020. https://doi.org/10.19708/j.ckjs.2020.01.006

[19] S. Y. Wang, J. N. Zheng, J. Z, Sun, Test case generation method for Web applications based on page object, *Journal of Computer Applications*, Vol. 40, No. 1, pp. 212-217, January, 2020. https://doi.org/10.11772/j.issn.1001-9081.2019060969

[20] Q. R. Zhang, S. Huang, L. L. Sun, A Survey of Web Function Automation Testing, *Software Guide*, Vol. 22, No. 3, pp. 227-236, March, 2023. https://doi.org/10.11907/rjdk.222368

[21] J. Hou, N. J. Gu, S. J. Ding, Y. K. Du, UI Automating Test Method for Cross-Device Based on Widget Path, *Computer Systems & Applications*, Vol. 27, No. 10, pp. 240-247, October, 2018.

[22] W. Y. Zhang, APP component recognition method based on object detection, *Journal of Computer Applications*, Vol. 40, No. z1, pp. 157-160, July, 2020. https://doi.org/10.11772/j.issn.1001-9081.2019081420

[23] L. Jia, J. Xu, X. Jin, H. Tian, Test case auto-execution system for Web application, *Computer Engineering and Applications*, Vol. 45, No. 4, pp. 82-85, February, 2009. https://doi.org/10.3778/j.issn.1002-8331.2009.04.023

[24] V. A. S. Júnior, E. Özcan, J. M. Balera, Many-objective test case generation for graphical user interface applications via search-based and model-based testing, *Expert Systems with Applications*, Vol. 208, Article No. 118075, December, 2022. https://doi.org/10.1016/j.eswa.2022.118075

[25] J. Fischbach, J. Frattini, A. Vogelsang, D. Mendez, M. Unterkalmsteiner, A. Wehrle, P. R. Henao, P. Yousefi, T. Juricic, J. Radduenz, C. Wiecher, Automatic creation of acceptance tests by extracting conditionals from requirements: NLP approach and case study, *Journal of Systems and Software*, Vol. 197, Article No. 111549, March, 2023. https://doi.org/10.1016/j.jss.2022.111549

[26] X. Y. Dang, J. F. Li, An Evolutionary Generation Method

for Path Coverage Test Data based on Mutation Testing, *Software Engineer*, Vol. 26, No. 1, pp. 46-49, January, 2023.
https://doi.org/10.19644/j.cnki.issn2096-1472.2023.001.010

[27] E. Alégroth, R. Feldt, P. Kolström, Maintenance of automated test suites in industry: An empirical study on Visual GUI Testing, *Information and Software Technology*, Vol. 73, pp. 66-80, May, 2016.
https://doi.org/10.1016/j.infsof.2016.01.012

[28] Z. Khaliq, S. U. Farooq, D. A. Khan, A deep learning-based automated framework for functional User Interface testing, *Information and Software Technology*, Vol. 150, Article No. 106969, October, 2022.
https://doi.org/10.1016/j.infsof.2022.106969

[29] S. R. Choudhary, D. Zhao, H. Versee, A. Orso, WATER: Web application test repair, *ETSE '11: Proceedings of the First International Workshop on End-to-End Test Script Engineering*, Toronto Ontario Canada, 2011, pp. 24–29.
https://doi.org/10.1145/2002931.2002935

[30] Q. M. Guo, N. B. Liu, Z. X. Wang, Y. L. Sun, Review of Deep Learning Based Object Detection Algorithms, *Journal of Detection & Control*, Vol. 45, No. 6, pp. 10-20+26, December, 2023.

[31] L. Wang, Y. Zhai, H. Hou, Genetic algorithms and its application in software test data generation, *Proceedings of the International Conference on Computer Science and Electronics Engineering (ICCSEE)*, Hangzhou, China, 2012, pp. 617-620.
https://doi.org/10.1109/ICCSEE.2012.36

[32] A. Kumar, G. S. Lehal, Layout Detection of Punjabi Newspapers using the YOLOv8 Model, *International Journal of Performability Engineering*, Vol. 20, No. 3, pp. 186-193, March, 2024.
https://doi.org/10.23940/ijpe.24.03.p7.186193

[33] S. P. Fang, B. Y. Ma, X. Y. Wang, C. G. Gao, Research on Classification of Software Test Data Automatic Generation Methods, *SOFTWARE*, Vol. 44, No. 8, pp. 41-43, August, 2023.
https://doi.org/10.3969/j.issn.1003-6970.2023.08.008

[34] J. N. Rao, Research and Analysis on GUI Testing Technologies, *Journal of Xichang College (Natural Science Edition)*, Vol. 33, No. 2, pp. 94-98+115, July, 2019.

[35] C. H. Feng, Z. P. Xie, B. W. Ding, Selective generation method of test cases for Chinese text error correction software, *Journal of Computer Applications*, Vol. 44, No. 1, pp. 101-112, January, 2024.
https://doi.org/10.11772/j.issn.1001-9081.2023010080

Biographies

Lei Xiao received her Ph.D. degree from Shanghai University in Shanghai, China in 2020. Since 2014, she has been working as an Associate Professor in the School of Computer and Information Engineering at Xiamen University of Technology. Her research interests are code security and software testing. She is the deputy director of the Fujian Province Software Evaluation Engineering Technology Research Center and a member of the Software Quality Engineering Standard Working Group of the Software and System Engineering Sub-Technical Committee of the China Information Technology Standardization Technical Committee.

Ru-Xue Bai is pursuing a MS degree in Computer and Information Engineering at Xiamen University of Technology. Her current research direction is software testing.

Ke-Shou Wu received his Ph.D. degree from Huazhong University of Science and Technology in 2011. From 2013 to October 2023, he served as a member of the Party Committee and Vice President of Xiamen University of Technology. In 2023, he assumed the position of Deputy Secretary of the Party Committee and Dean of Fujian Polytechnic Normal University. His current main research interests include software architecture, embedded system security, cloud computing, and its applications.

Rong-Shang Chen, born in1982, senior engineer. His main research interest in cludes software defect prediction.