Journal of Software Engineering and Applications, 2025, 18(9), 336-357
https://www.scirp.org/journal/jsea
ISSN Online: 1945-3124
ISSN Print: 1945-3116

Strategy and Methodology of Unit Testing for GUI Software Based on Dynamic "Pair-Wise" Mode

Mengqing Tanli[1], Jiyi Xiao[1], Ying Zhang[2], Guohua Zhang[3]

[1]School of Software (School of Computer), University of South China, Hengyang, China
[2]School of Mechanical Engineering, University of South China, Hengyang, China
[3]University of South China, Hengyang, China
Email: TLMQ-TEAM@163.com

How to cite this paper: Tanli, M.Q., Xiao, J.Y., Zhang, Y. and Zhang, G.H. (2025) Strategy and Methodology of Unit Testing for GUI Software Based on Dynamic "Pair-Wise" Mode. *Journal of Software Engineering and Applications*, **18**, 336-357.
https://doi.org/10.4236/jsea.2025.189020

Received: July 28, 2025
Accepted: September 8, 2025
Published: September 11, 2025

Abstract

In order to effectively organize and control the unit testing activity, the strategy and methodology of unit testing have been investigated to assure the unit quality of software. Facing challenges of unit testing for GUI software, we proposed our strategy and methodology including dynamic testing organization strategy, incremental updating strategy of testing process, "Layer unit testing" definition and methodology of GUI unit testing, and distinguished "Triple-step method" with two factual examples. For a bit of necessities, some measures of dealing with difficulties in traditional unit testing were proposed. Additionally, we discussed some effective interaction aspects between testing and programming to improve testing efficiency for GUI software. These strategies and methodologies have generally been verified and tested by factual software testing practice or testing experimentation.

Keywords

Strategy and Methodology, Dynamic "Pair-Wise" Mode, "Layer Unit Testing", "Triple-Step Method", GUI Software

1. Introduction and Background

The GUI software is the new type of today software technology that perhaps will be sustained for a long time, though the strengthened technology of GUI has emerged. And facing the composite trend of developing process and testing process, new software developing technology will pursuit new software testing technology with the developing pace of update science and technology [1] [2].

Unit testing is the primary base of software testing, and it must still execute in terms of "General thought of software testing—three principles and laws of software testing", *i.e.* a) Simplifying or localizing the testing object; b) Stepping or staging the testing process; c) Dispersing happened problem and tackling it as early as possible to avoid accumulation [3]-[7].

Unlike low level unit testing of code snippet or member function, GUI-oriented software testing should test the software implementation of whole GUI software, including its own unit testing and integration testing. Necessarily, the unit testing in GUI-oriented software testing should be different from the past unit testing, and has its own strategy and methodology [8] [9]. Synthesizing all kinds of testing theory and technology [2]-[7], and summarizing our testing practise [10]-[12], we can conclude that the execution strategy of unit testing should apply layer strategy [8], just called "Layer unit testing", and it can be effectively executed in majority of GUI software.

Good testing organization is the assurance of efficiency and effective activity for software testing. The "Pair-wise" mode, as we demonstrated in previous articles [8]-[10], is a general organizing mode not only for pure testing team but also for mixed team with programmers and testers in unit testing. Of course, this mode should have its reasonable and necessary change with the developing organization and over time.

In the past, for the basic lecture on unit testing, Fu Bing [4] has perfectly expounded with a full chapter, but the basic footing of unit testing for GUI software is not indicated instead of menu, or window. Referring to testing content in unit testing, Ron Patton [6] has paid close attention to boundary value and sub-boundary conditions. Referring to the testing method of unit testing, Li Fan [2] has deeply discussed the method of state transform diagram, especially in terms of the coverage problem of "N + 1-swith" with examples in detail. However, all these researchers and authors have not explicitly described the coordination problem between programmer and tester during developing period.

As a consequence, the changing organization of software developing is proposed by Emil Alegroth, Robert Feldt [13] with the term "Squad", however, the detail organizing architecture of test team for software testing is lack especially for unit testing and small team, and the requirement of dynamic variety was not analyzed and discussed with the development of computer science and technology, and the changing of developing organization especially in China. The testing process in software companies is deeply depicted in the case study of Emil Alegroth, Robert Feldt [13], but the cooperation mechanism of test team is not given.

For our studies in unit testing, recent studies [8] have deeply investigated the unit testing for static testing organization—"Pair-wise" mode, and it mainly focused on the internal cross-testing in unit testing. Consequently, for the test suite construction of unit testing for GUI software, the previous research [8] [9] has proposed the "Triple-step method" for unit testing and factual effect has been verified by a lot of examples, however the combined coverage problem was not dis-

cussed. And a recent study [10] has put forward the front-end method of white box in the "Grey-box" method, *i.e.* the instrumentation testing with "MessageBox()" for window access controls in unit testing.

Thus, this paper will propose the dynamic testing organizing for unit task and specific unit testing, and some usable strategies and methodologies of unit testing are addressed. As a key point, the "Triple-step method" for "parameters combination" situation is deeply analyzed and investigated. Therefrom, some measures in traditional unit testing were depicted and some effective coordinated interaction aspects between testing and programming are discussed to improve testing efficiency for GUI software.

2. Challenges of Unit Testing for GUI Software

For GUI software, the unit testing is not only related to the internal aspects e.g. internal sub-program or member function, but also it is greatly influenced by the GUI characters [14]. Furthermore, the unit testing must consider the new emerging technology to keep advanced developing pace. Hence, following challenges should be considered for the unit testing of GUI software.

- New software types and developing thought have brought challenges to software testing, in which they are emerged or are continuously emerging, e.g. GUI, App, strengthened technology of GUI, and visual developing, etc. And GUI software has its own features, GUI software testing should be done, distinguishing to traditional software testing [14]-[18]. At the same time, the user-oriented thought should be the core requirement for GUI software testing including unit testing [2]-[5] [10].
- Testing object and environment require the change of software testing technology e.g. remote network, cloud computing, industry field, etc. Changing with the variety of testing object and environment is the update requirement of software testing, so the unit testing should completely tackle the variety of two aspects including GUI implementation and pure low-level function implementation [14] [15].
- New testing organization should be innovated to fit own factual developing situation and get competitive testing cost especially in China [8] [13].
- New effective testing method should be created to improve the testing efficiency of GUI software testing including unit testing. Consequently, efficient and economic consideration should come to ground for the unit testing of GUI software [7] [9]-[12].
- How to execute the unit testing of GUI software testing? Copying blindly traditional test theory in factual testing practice is not a good way, and copying blindly past test methods is not also the good strategy and methodology [8]-[12].
- For rapidly iterative software developing, software testing should keep the pace. It will refer to testing organization also to testing method and technology. On one hand, the statistical sampling method for GUI-oriented testing

can be considered, on the other hand, of course, the rapid regression testing should also be taken into account for update software producing [8] [10] [16].

- How to face various industrial scenarios? Tackling the difficulty in unit testing to various industrial scenarios, and synthesizing all factors to assure the unit quality of GUI software products are very necessary. The basic application software, professional application software, and mobile App, etc. should be considered for various scenarios and situations [13] [15] [17] [18].

3. Strategy and Methodology

3.1. Organization Strategy [8]

For unit testing of GUI software, the "Pair-wise" mode is still a good strategy not only for desktop software testing scenario but also for mobile software testing scenario. However, distinguishing with the testing object, testing condition and testing method, the "Pair-wise" mode should be changed dynamically, and it should be called "Dynamic pair-wise" mode, as shown in **Figure 1** and **Figure 2**. The **Figure 1** has depicted dynamic testing organizing of task-oriented requirement including general testing task and special testing task, and **Figure 2** is the dynamic testing organization for specific unit testing, in which specific unit testing is divided into "Self-Testing", "Cross-Testing" and "Independent-Testing". In general, the "Dynamic" feature should reflect in the task-oriented adjustment of organization type and the task-oriented allocation of staff.

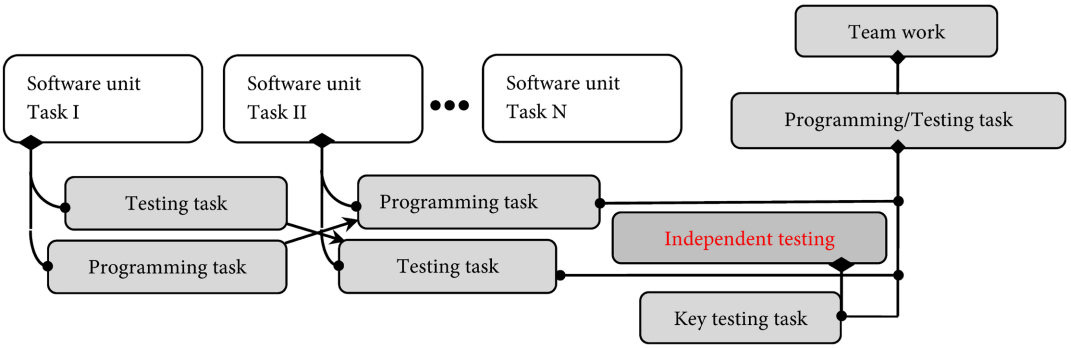

Figure 1. Dynamic testing organizing for unit task.

Consequently, in the "Dynamic pair-wise" organization, for large scale software producer, the testing department should be completely independent to the developing department. In this case, general responsibility of unit testing must be under controlled by the testing department, and the developing department may be responsible for the supervision of self-constructing of test class, or called TDD (Test-Driven Development). Otherwise, for medium and small scale software producer, the testing department may be unified with the developing department, but the testing department should monitor the unit quality using independent testing way, including sampling testing method.

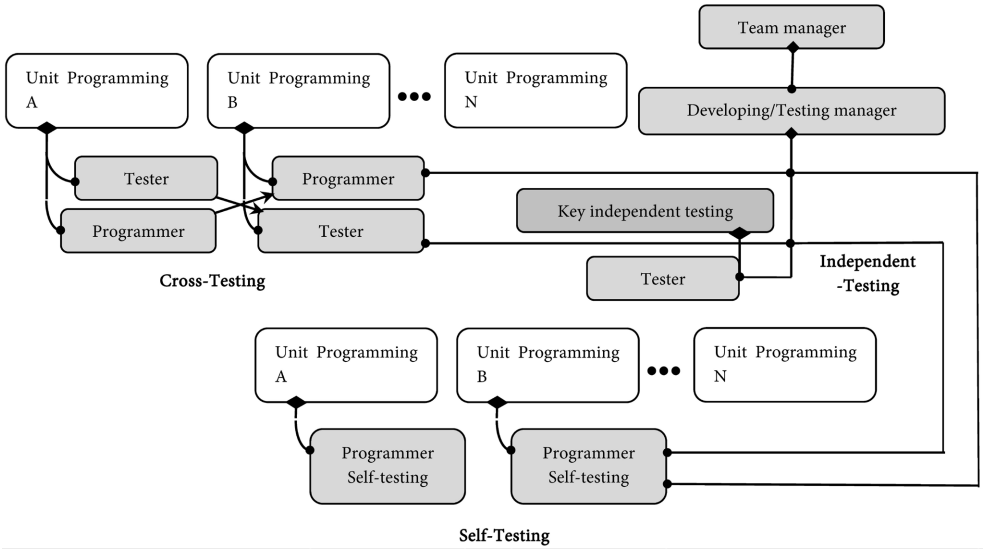

Figure 2. Dynamic testing organizing for unit task.

For "Dynamic pair-wise" organization, if there are some beginners or rookies in the developing/testing department after new hiring, beginners can be leaded and instructed by experienced developer/tester. To some extent, beginners may be arranged as a pair, but their codes should be tested with sampling method by the independent tester.

3.2. Process Strategy [9]

Software unit testing is a composed process, which all needed resources are synthesized to keep testing activity orderly and smoothly.

3.2.1. Incremental Updating Strategy

For GUI software, especially for today software with online version updating, the general strategy is the incremental testing strategy based on variety, and this incremental testing strategy can be described as shown in **Figure 3**. In testing loop1, code review should be taken into account. However, code review only is an alternative task in following testing loop according to factual situation, and it is instead of check and analysis, which may include dependency analysis for units or modules in GUI level [16], interaction analysis for operation or function in code level, etc.

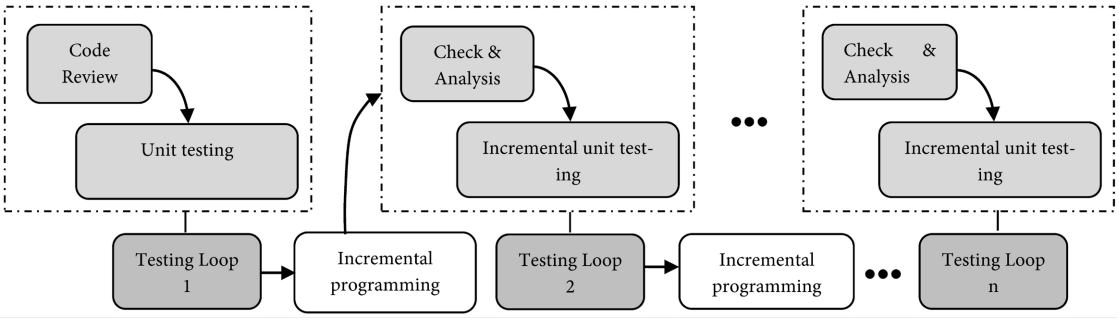

Figure 3. Procedure of incremental unit testing.

Figure 4 has given the testing cycle in unit testing for GUI software especially for large scale software system. At first, codes of programmer are saved in the "Initial pool", and test department prepares to testing activity. After codes are received, necessary analysis is done, and then conduct test cases which is the most important step for testing activity. All test cases and test suite should save in the test suite or test case base, and in construction of test case and doing testing, the "Triple-step method" should be adopted, which mainly includes "Data-restriction testing", "Function testing" and "State testing". After testing recording is finished, it should be submitted with the testing conclusion to testing manager or supervisor. At the same time, main testing record and testing conclusion accompanied by codes must transmitted into next "Refresh pool", and if the testing result is "no pass", the programmer must modify the code to next testing cycle.

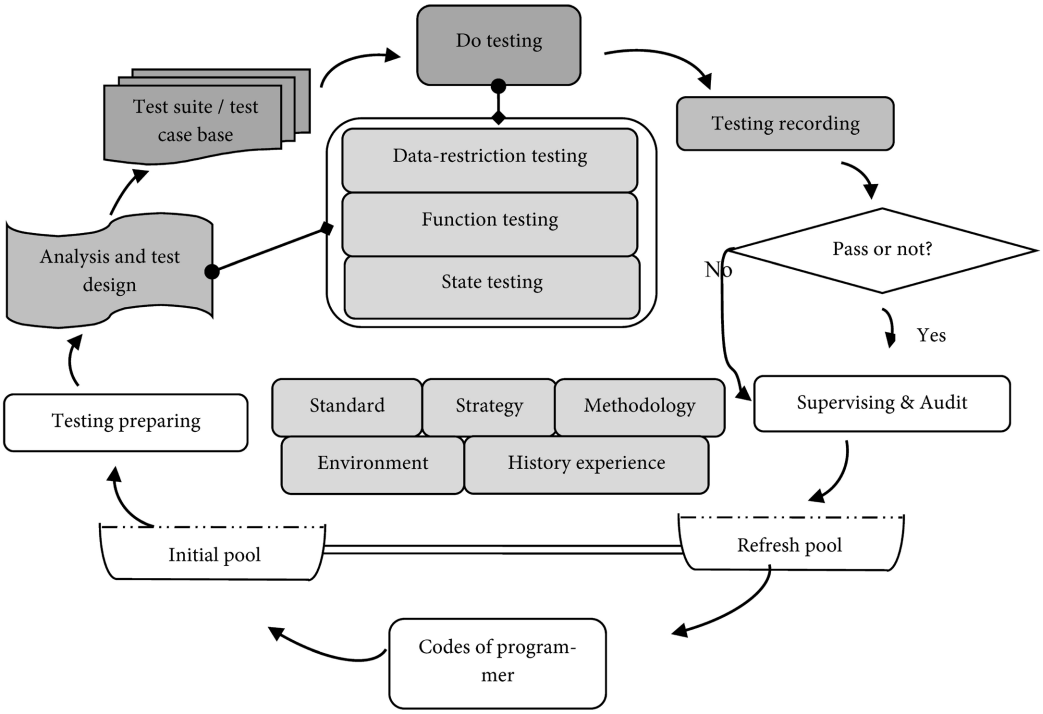

Figure 4. Testing cycle in unit testing.

3.2.2. "Layer Unit Testing" Definition and Method

How to simplify or localize the testing object? Here, of course, it is confined to the unit testing of GUI software. By the testing practise and experience summarizing, we have proposed the "Layer unit testing" strategy to deal with this difficult problem, and the previous research [9] [10] have initially investigated this issue, but this term is defined this time. "Layer unit testing" is mean that the layer of software can be taken as the center point of unit testing, and the "Form/Sheet/Dialogue" GUI with accumulated function and data handling can be served as layer unit more time. In fact, "Layer unit testing" can be derived from the category structure of GUI software, such as the illustration of Figure 5.

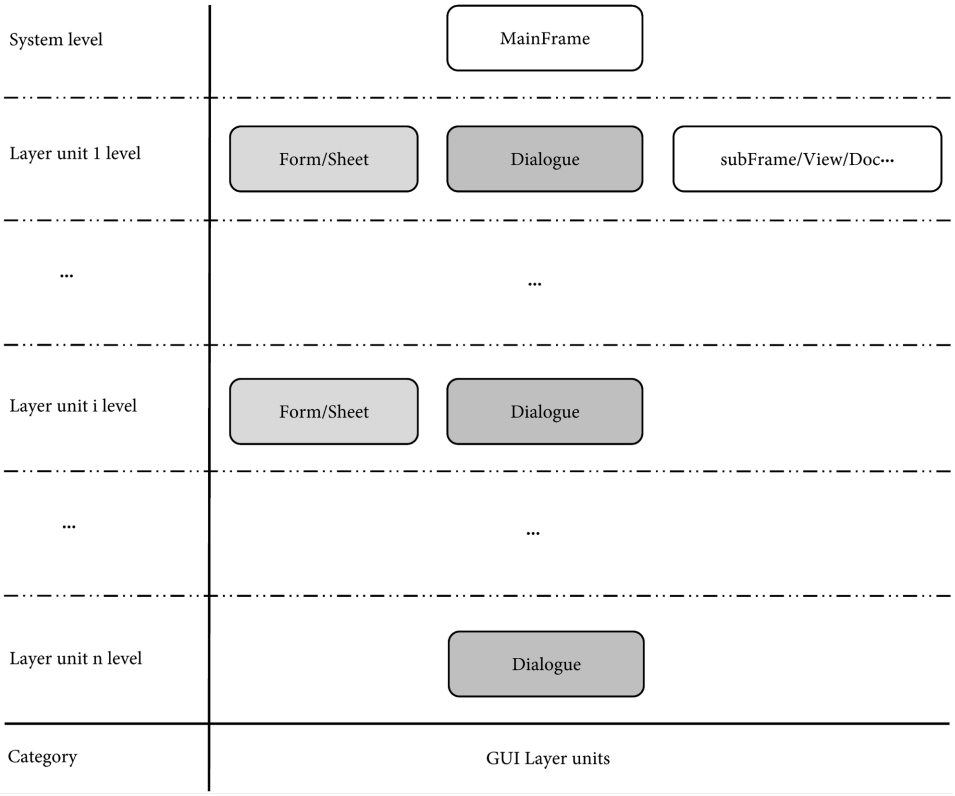

Figure 5. The category structure—Origin of "Layer unit testing".

3.3. Test Tool and Framework Strategy [8]-[10]

At present, for the GUI-oriented software testing, there are not complete and very suitable tool or framework for the "Layer unit testing". However, for one testing step or one special testing scenario, you may select existed tool or framework and self-fabricated tool or framework to accelerate the testing process.

3.3.1. Selection and Application of Existed Testing Tool or Framework

Factors of selection and application generally include:

- The status of testing object—programming language, the scale of software project, and programming tool and environment.
- The requirement of testing time—the early development stage, the later development stage especially for rapid software release.
- The situation of test organization and testers—small team, medium and large scale organization.

3.3.2. Basic Test Theory and Method

Obviously, construction of test tool and framework is based on the effective test theory and method. For GUI-oriented software unit testing, some new test tool and framework should be constructed, e.g. for "Grey-box" testing theory and method, and for "Triple-step method" in "Layer unit testing" technology.

3.3.3. Stages for China

For fabrication of test tool and framework by self, gradual strategy should be used,

and it should adopt several stages to avoid risks. In the beginning, one basic or important part is developed, e.g. constructing a tool for "Triple-step method", we can start at one testing step—the data restriction testing, or the function testing, or the state testing. In fact, we have started to develop test tool for "Triple-step method" mainly focusing on the state testing, temporarily called "intelligent sate testing".

3.3.4. Independent Intellect Property Right

At present, most test tools and framework for low-level unit testing are made abroad, and are most based on white-box technology. In China, the independent intellect property right must be taken into account for developing testing tool and framework, including GUI-oriented testing tool like IDE (Integrated Development Environment).

3.4. "Triple-Step Method" for "Parameters Combination" Situation

In factual testing scenario of GUI software, "Triple-step method", which the process is shown in Figure 6, is a good method to improve testing efficiency and assure software quality. As mentioned above, the applied effect with detailed examples has been investigated in our previous study [8] [9]. Obviously, "Data-restriction testing" at the beginning of testing process is an effective measure for finding BUG as early as possible.

Rather, in the past investigation, only the situation of input "parameters independence" is discussed, e.g. these examples are only given for which input parameters are not interactive. This independent situation is widely existed, but another situation called "parameters combination", which input parameters are interactive at some extent, is also a popular type. The execution of the two kinds of situations for "Triple-step method" has a bit of difference, and main distinguishes are placed in the "Function testing" step, and the details are shown in Figure 7.

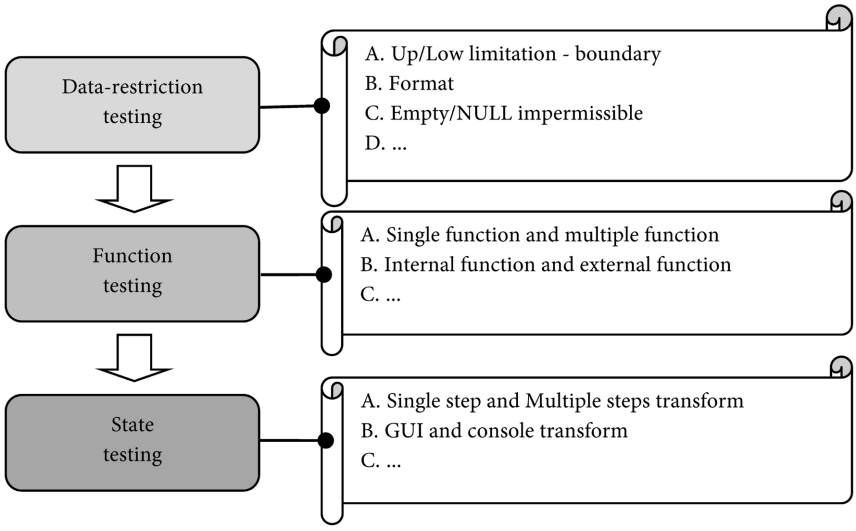

Figure 6. "Triple-step method" of "Layer unit testing" for GUI software.

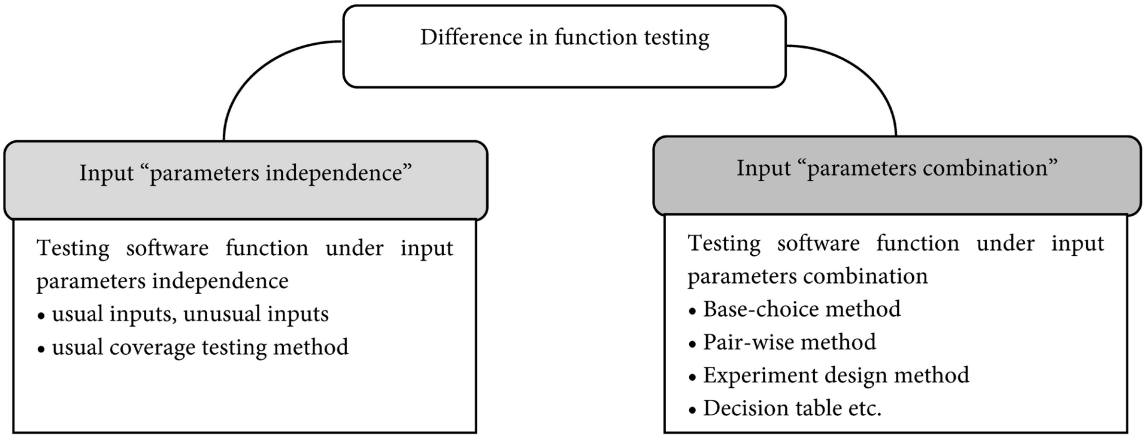

Figure 7. Two situations of "Function testing" in "Triple-step method".

Consequently, for the "Triple-step method" of "Layer unit testing", some modification and improvement are done in terms of our testing practise and systemic consideration. In the first step, the term is modified to "Data-restriction testing" with more accurate expression. The function testing, as the second step, is focused on the testing of key and main function implementation, and some exception handling may be transmitted to the third step or can be combined with the sate testing. In state testing, by our testing practise, "The improved STD method" should be adopted, and the acronym "STD" should be derived from "state transition diagram", and the other terms and detail description, of course, have been depicted in our past research [8]-[12]. However, by empirical summary, we consider that "the action element" may omitted in the improved STD, because its expression can be determined by "the event element", and it will decrease the complexity of the improved STD with less repeated components.

As early depicting, the situation of "parameters combination" exists in popular in "Layer unit testing", *i.e.* some "Layer units" have "Input controls" or components which are interactive each other, and some values of "Input controls" or components will certainly influence others. For this situation, the particular testing method must be adopted, including base-choice test method, "Pair-wise" test method, experimental design method, etc. Here, two examples are used to investigate this situation as follows.

3.4.1. Example 1—The Sheet of Authority Setting

The sheet of authority setting is a very important function implementation to set the authority of visiting users in PQMS2 (Product Quality Monitoring System), and four kinds of authority are defined including viewing, adding, modifying and deleting, while other authorities are omitted, e.g. printing authority and saving authority. In GUI layout, the sheet of authority setting is based on a "Dialogue Class", and "Input Controls" include four "Labels" and four "Check Boxes", and the details of authority setting sheet are showed in **Figure 8**. Obviously, four kinds of authority are interactive in this authority setting sheet.

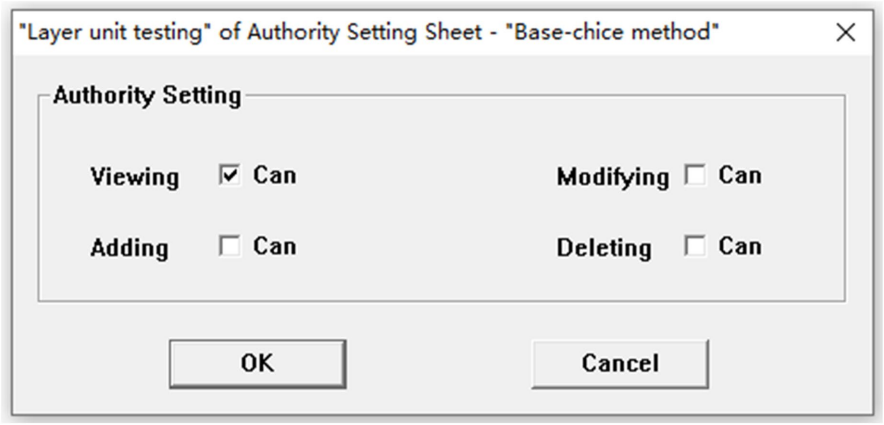

Figure 8. The sheet of authority setting in PQMS2.

As a consequence, "Layer unit testing" of this sheet should be done with "Triple-step method" and considered to dispose in terms of "Input parameters combination". With formalization of "Triple-step method", the following is detail discussion.

1) Data-restriction testing

For the layout of this sheet, it is relative uncomplicated with less "Input Controls" and components, but the importance of this sheet is obviously no doubt. Because the authority setting is very important for whole safety in PQMS2, the sampling style should not be used in data-restriction testing, and the total-testing manner must be chosen firstly.

As one testing of typical "Input Control", the data-restriction testing of "Check Box" may executed as follows, which every "Check Box" is considered to be two values only.

- Click all the "Check Boxes", and display that the "Check-box" is checked.
- Click all the "Check Boxes" again, and display that the "Check-box" is not checked.

2) Function testing—"Base-choice testing method"

Consequently, as mentioned above, because there are interactions among "Input Controls", the function testing of this authority setting sheet must be considered under multiple parameters combination. At the same time, it is easy to know that viewing authority is the base condition for others.

(1) Parameter-Value couples/P-Vs

a) Viewing authority—Can, Can not

b) Adding authority—Can, Can not

c) Modifying authority—Can, Can not

d) Deleting authority—Can, Can not

In these P-Vs, the viewing authority is the basic value, so the construction of test case could be done with "Base-choice testing method" in this example. It is easy to know that, if the "All combinations testing method" is used, $2 \times 2 \times 2 \times 2 = 16$ test cases will be needed.

(2) Testing covering items

TCOVER1 Viewing can;
TCOVER2 Viewing can not;
TCOVER3 Adding can;
TCOVER4 Adding can not;
TCOVER5 Modifying can;
TCOVER6 Modifying can not;
TCOVER7 Deleting can;
TCOVER8 Deleting can not.

(3) Construction of test case

Using the "Base-choice testing method", the basic value "Viewing authority—Can" is used to construct test case in priority, as illustrated in the second column of Table 1. Eventually, five test cases are generated, and the details are shown as following Table 1.

From the result of Table 1, we can note that the number of test case constructed with "Base-choice testing method" is less than the "All combinations testing method", and efficiency improvement achieved 320%, *i.e.* 3.2 times. Hence, the test case construction with "Base-choice testing method" is quite fitted to this kind of testing scenario of "Input parameters combination", *i.e.* existing basic value in P-Vs.

Table 1. Test cases of function testing of authority setting sheet—Base-choice testing method.

Test case ID	Input				Cover combination—No repeat	Expected output
	Viewing authority	Adding authority	Modifying authority	Deleting authority		
GCEX-ASS-UNI-TC000-AD	Can	Can	Can	Can	TCOVER1, TCOVER3, TCOVER5, TCOVER7	Correct
GCEX-ASS-UNI-TC001-AD	Can	Cannot	Can	Can	TCOVER4	Correct
GCEX-ASS-UNI-TC002-AD	Can	Can	Cannot	Can	TCOVER6	Correct
GCEX-ASS-UNI-TC003-AD	Can	Can	Can	Cannot	TCOVER8	Correct
GCEX-ASS-UNI-TC004-AD	Cannot	Can	Can	Can	TCOVER2	Incorrect

3) State testing—"The improved STD method"

In the sheet of authority setting, actual executing functions are less, in which there are one main function handling and one secondary function handling. Besides the function handling, exception handling must be considered in state testing. In general, the improved STD of this authority setting sheet is relatively simple, and the factual drawing result is shown in Figure 9.

According to the improved STD of this authority setting sheet, it is not difficult to conduct test cases of state testing because of the less functional paths and exceptional paths. And the results are demonstrated in Table 2.

Table 2. Test cases of state testing of authority setting sheet—The improved STD method.

Test case ID	Input					Expected output
	"Check-box"-Viewing authority	"Check-box"-Adding authority	"Check-box"-Modifying authority	"Check-box"-Deleting authority	Other controls	
GCEX-ASS-UNI-TC010-AD	Keep default	-	-	-	Click "OK"	Save and exit sheet
GCEX-GCS-UNI-TC011-AD	Keep default	-	Click	-	Click "OK"	Save and exit sheet
GCEX-GCS-UNI-TC012-AD	Keep default	Click	-	-	Click "Cancel"	Exit sheet
GCEX-GCS-UNI-TC013-AD	Keep default	-	-	Click	Click the right-up "×"	Exit sheet
GCEX-GCS-UNI-TC014-AD	Keep default	-	-	-	Click "Cancel"	Exit sheet
GCEX-GCS-UNI-TC015-AD	Keep default	-	-	-	Click the right-up "×"	Exit sheet

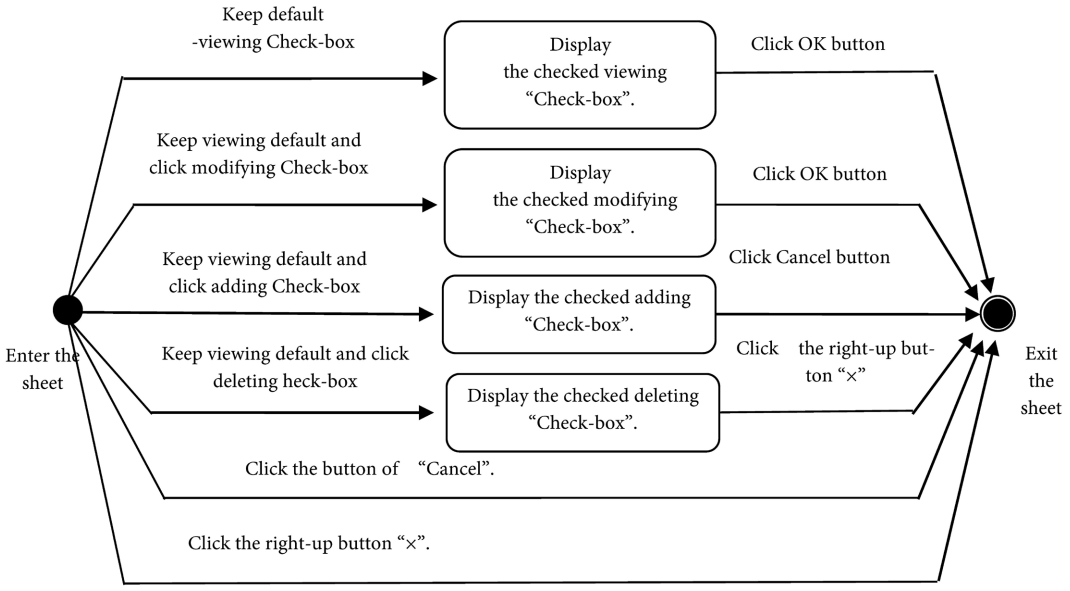

Figure 9. The improved STD of the authority setting sheet.

3.4.2. Example 2—The Sheet of Basic Setting in Quality Control

In our factual software system—PQMS2, the sheet of basic setting in quality control is an initializing unit for the system, which provided some initial parameters for the generation of quality figures. The details of the sheet of basic setting in quality control, updated are shown in **Figure 10**. For the layout of this sheet of authority setting, it is also generated with a "Dialogue Class", and "Input Controls" include four groups, *i.e.* (a) Accuracy item—one "Label", and three "Radio Buttons"; (b) Sample volume item—one "Label", and one composed control with "Edit Box" and "Spinner"; (c) Count limitation item—one "Label", and three "Ra-

dio Buttons"; (d) Tester item—one "Label", and one "Combo Box". By requirement analysis, we can know that four items for quality figure are interactive in this setting sheet.

Similarly, "Layer unit testing" of this sheet should be done with "Triple-step method" and considered according to "Input parameters combination", and the detail discussion is as follows.

1) Data-restriction testing

Because this sheet is a key unit with whole constraint in PQMS2, the data-restriction testing of this setting sheet should not use sampling style, and the total-testing manner must be applied.

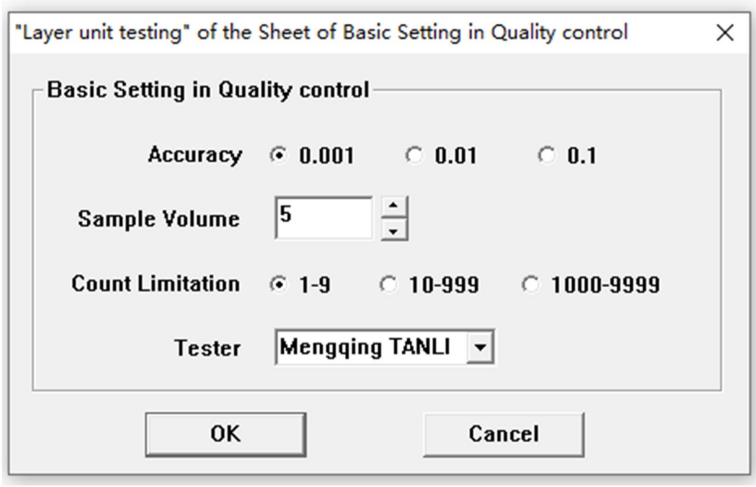

Figure 10. The sheet of basic setting in quality control-updated.

All "Input Controls" of this setting unit of control chart must be tested for boundary value. Consequently, these input controls in four groups should be verified in terms of respective type. *i.e.*, (1) For "Accuracy" item and "Count limitation" item—every "Radio button" is also considered to be two values only like "Check Box", and testing method is to click every "Radio button" in turn. (2) For "Sample volume" item—click the "Spinner" to the up limit value and the low limit value to check whether the values are correct for specification. (3) For "Tester" item, in the "Combo Box", input continuously character "T....." by keyboard, and stop when listened to the "Da Di" voice until can not input again. Then, check whether the number of input characters is less than 15.

2) Function testing

Similarly, because there are interactions among "Input Controls" of this sheet, the function testing of this sheet must be considered under multiple parameters combination. At the same time, there is not basic value in this example, "Basic-choice testing" method is not a good choice, and the construction of test case could be done in terms of "Pair-wise testing method", and details of "Pair-wise testing method" please refer to [2]. Eventually, 13 test cases are generated with combination coverage, and the details are shown in **Table 3**.

Table 3. Test cases of the sheet of basic setting in quality control—"Pair-wise testing method".

Test case ID	Input				Cover combination—No repeat	Expected output
	Accuracy	Sample volume	Count limitation	Tester		
PQMS2-MPC-UNI-TC010-AD	0.1	List item	1 - 9	List item	TCOVER1, TCOVER4, TCOVER13, TCOVER22, TCOVER25, TCOVER28	Be sure
PQMS2-MPC-UNI-TC011-AD	0.1	List item	10 - 999	Self-filled	TCOVER5, TCOVER14, TCOVER23, TCOVER26, TCOVER32	Be sure
PQMS2-MPC-UNI-TC012-AD	0.1	List item	1000 - 9999	Empty	TCOVER6, TCOVER15, TCOVER24, TCOVER27, TCOVER36	Be sure
PQMS2-MPC-UNI-TC013-AD	0.01	List item	10 - 999	List item	TCOVER2, TCOVER8, TCOVER16, TCOVER31	Be sure
PQMS2-MPC-UNI-TC014-AD	0.001	List item	1 - 9	List item	TCOVER3, TCOVER10, TCOVER19	Be sure
PQMS2-MPC-UNI-TC015-AD	0.01	List item	1 - 9	Self-filled	TCOVER7, TCOVER17, TCOVER29	Be sure
PQMS2-MPC-UNI-TC016-AD	0.01	List item	1000 - 9999	Empty	TCOVER9, TCOVER18	Be sure
PQMS2-MPC-UNI-TC017-AD	0.001	List item	10 - 999	Self-filled	TCOVER11, TCOVER20	Be sure
PQMS2-MPC-UNI-TC018-AD	0.001	List item	1000 - 9999	Empty	TCOVER12, TCOVER21	Be sure
PQMS2-MPC-UNI-TC019-AD	0.1	List item	1 - 9	Empty	TCOVER30	Be sure
PQMS2-MPC-UNI-TC020-AD	0.1	List item	10 - 999	Empty	TCOVER33	Be sure
PQMS2-MPC-UNI-TC021-AD	0.1	List item	1000 - 9999	List item	TCOVER34	Be sure
PQMS2-MPC-UNI-TC022-AD	0.1	List item	1000 - 9999	Self-filled	TCOVER35	Be sure

Accordingly, from the result of Table 3, the totality of test cases of using "Pair-wise testing method" is 13, but the number of test cases using "All combinations testing method" will be $3 \times 3 \times 3 \times 3 = 81$. it is revealed that the number of test case constructed with "Pair-wise testing method" is very less than the "All combinations testing method".

3) State testing

This example has relatively complex composition with more input controls, and the improved STD is relatively difficult to draw. On the other hand, the main function disposing is only one in this basic setting sheet, more attention should be payed to various running states for combination input of multiple parameters. Nonetheless, the improved STD of this basic setting sheet is not very complicated, and the details are shown in Figure 11.

After the improved STD of this sheet is finished, test cases of state testing can be conducted with it. Obviously, the construction process is not very difficult because of the less functional paths and exceptional paths. And the result of test cases of state testing is demonstrated in Table 4.

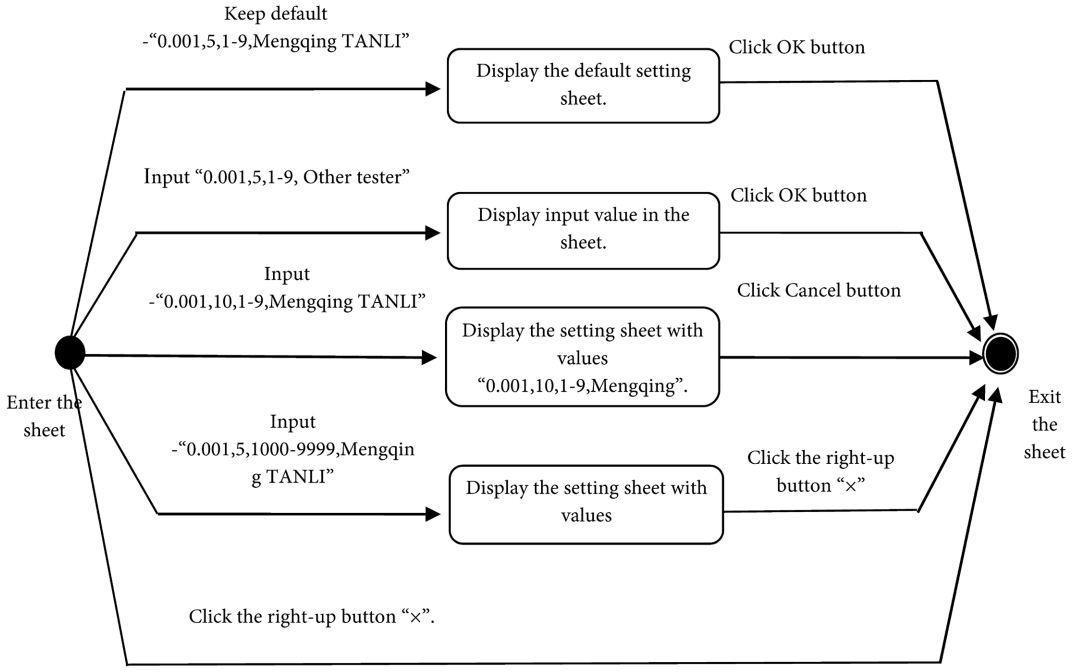

Figure 11. The improved STD of basic setting sheet in quality control.

3.5. Some Measures to Deal with Difficulty in Traditional Unit Testing

3.5.1. Supervising of Traditional Logic Coverage Testing [2]-[4]

Most of traditional logic coverage testing should be executed by programmer self, and independent tester can do sampling testing for very key unit under warrant of testing manager. In general, the execution index of testing quality may adopt the coverage rate in unit testing of bottom level.

For safety-critic software, MCDC is the basic requirement for white-box testing, and for the key unit of commercial software, MCDC should be executed in terms of factual situation of the testing object.

Table 4. State test cases of the sheet of basic setting—The improved STD method.

Test case ID	Input	Expected output
PQMS2-BSS-UNI-TC000-AD	Keep default setting -"0.1, 5, 1 - 9, Mengqing TANLI", and click the button of "OK".	Save setting data and exit the sheet normally
PQMS2-BSS-UNI-TC001-AD	In the Combo Box "Tester", input "Other tester", and click the button of "OK".	Prompt "No this tester."
PQMS2-BSS-UNI-TC002-AD	In the Edit Box "Sample volume", input "10", and click the button of "Cancel".	Without saving setting data and exit the sheet normally
PQMS2-BSS-UNI-TC003-AD	In the item of "Count limitation", click "1000 - 9999", and then click the button of "×" on the right-top of sheet.	Without saving setting data and exit the sheet normally
PQMS2-BSS-UNI-TC004-AD	No operation, and click the button of "×" on the right-top of sheet.	Exit the sheet normally

3.5.2. Exploratory Method of Loop Statement Testing

At present, there are not well-known methods for the loop statement testing with

reasonable coverage index. Here, we give a reference resolution, which synthesizing traditional classic testing methods, mainly using boundary-value testing method and logic coverage testing method. As a consequence, following should be noticed:

1) Serial loop structure generally may be transformed into respective single loop structure to tackle;

2) For embedded loop structure, the internal loop should be firstly tested and then external loop;

3) The boundary condition testing of loop should be the key point;

4) Data of factual scenario is more effective, which has considered accuracy, format, etc.

In order to explain the details of our resolution to the loop statement testing, we give an example in PQMS2, *i.e.* the computation of XAve values for XAve-R control chart, which codes are shown in Figure 12. It is easy to know, in Figure 12, there are two loop including loop 1—group division of values and loop 2—summing of division values and computing average of group values.

Consequently, testing contents of the computation of XAve values for XAve-R control chart include two parts:

1) The 1st loop—Group division of values

In this loop, boundary condition is "Total_Team" and "g + Volume", loop variable is "i" and "k", and increment is "1", and loop statement or expression is "Group [i] [j] = Value [k]". Obviously, two layer loops with one embedded loop should be tested.

```
int i=0, j=0, k=0, g=0, Volume=3, N=60,Total_Team=0;
float Sum=0.0;
float Value[1000], Group[100][10], Ave[100];
Total_Team=N/Volume;
for (i=0;i<Total_Team;i++)
{
        g=i*Volume;
        j=0;
        for(k=g;k<g+Volume;k++)
        {
                Group[i][j]=Value[k];
                j++;
        }
}
for (i=0;i<Total_Team;i++)
{
        Sum=0.0;
        for(j=0;j<Volume;j++)
        {
                Sum+=Group[i][j];
```

Figure 12. Codes of the computation of XAve values for XAve-R control chart.

2) The 2rd loop—Summing of division values and computing average of group values

In the 2rd loop, boundary condition is "Total_Team" and "Volume", loop variable is "i" and "j", and increment is "1", and loop statement or expression is "Sum+ = Group [i] [j]". Similarly, two layer loops with one embedded loop should be tested.

Due to limited space, we only give the test case of loop 2 in Table 5 and test data in Figure 13, because the 1st loop has similar loop structure and the key point of XAve-R computation is located in loop 2.

Table 5. The test case example of loop statement testing.

Test case ID	Requirement	Input	Expected output
PQMS2-XRs-UNI-TC000-AD	For (1) i = 0, and (2) loop variable j = three values (the start_value, the value of one increment, the up value), test the value of Sum	Volume = 3, Total_Team = 1; j = 0, 1, 2; Value [0] = −0.04, Value [1] = 0.0, Value [2] = −0.02	i = 0, j = 0, Sum = −0.04; i = 0, j = 1, Sum = −0.04; i = 0, j = 2, Sum = −0.06; Ave[0] = −0.02
PQMS2-XRs-UNI-TC001-AD	For (1) i = 0, and (2) loop variable j = unusual value, test the value of Sum	Total_Team = 1; j = 240	i = 0, j = 240, Sum = 0.0; Ave[0] = 0.0
PQMS2-XRs-UNI-TC002-AD	For (1) i = 0, and (2) loop up limit = 1, test the value of Sum	Total_Team = 1; Volume = 0	Sum = 0.0; Ave[0] = 0.0
PQMS2-XRs-UNI-TC003-AD	For i = 0 - 10, test the value of Sum	Volume = 3, Total_Team = 11; j = 0, 1, 2	i = 0, Sum = −0.06, Ave[0] = −0.02; i = 1, Sum = −0.10, Ave[1] = −0.0333; i = 2, Sum = −0.08, Ave[2] = −0.0267; i = 3, Sum = −0.12, Ave[3] = −0.04; i = 4, Sum = −0.06, Ave[4] = −0.02; i = 5, Sum = −0.06, Ave[5] = −0.02; i = 6, Sum = −0.10, Ave[6] = −0.0333; i = 7, Sum = −0.08, Ave[7] = −0.0267; i = 8, Sum = −0.11, Ave[8] = −0.0367; i = 9, Sum = −0.10, Ave[9] = −0.0333; i = 10, Sum = −0.10, Ave[10] = −0.0333
PQMS2-XRs-UNI-TC004-AD	For i = 0 - 18, test the start_value, medium_value, end_value of Sum	Volume = 3, Total_Team = 19; i = 0, 11, 18	i = 0, Sum = −0.06, Ave[0] = −0.02; i = 11, Sum = −0.06, Ave[11] = −0.02; i = 18, Sum = −0.10, Ave[18] = −0.0333
PQMS2-XRs-UNI-TC005-AD	For i = 19, test the start_value, medium_value, end_value of Sum	Volume = 3, Total_Team = 20; i = 0, 12, 19	i = 0, Sum = −0.06, Ave[0] = −0.02; i = 12, Sum = −0.10, Ave[12] = −0.0333; i = 19, Sum = −0.10, Ave[19] = −0.0333
PQMS2-XRs-UNI-TC007-AD	For i = 20, test the value of Sum	i = 20	i = 20, Sum = 0.0; Ave[20] = 0.0
PQMS2-XRs-UNI-TC008-AD	For i = unusual value, test the value of Sum	i = 210	i = 210, Sum = 0.0; Ave[210] = 0.0
PQMS2-XRs-UNI-TC009-AD	For the loop up limit = 0, test the value of Sum	Total_Team = 0	Sum = 0.0; Ave[210] = 0.0

Group[0] ={-0.04, 0.0, -0.02}; Ave=-0.02
Group[1] ={-0.06, 0.0, -0.04}; Ave=-0.0333
Group[2] ={ -0.02, -0.06, 0.0}; Ave=-0.0267
Group[3] ={-0.02, -0.06, -0.04}; Ave=-0.04
Group[4] ={0.0, -0.02, -0.04}; Ave=-0.02
Group[5] ={-0.05, 0.01, -0.02}; Ave=-0.02
Group[6] ={-0.04, 0.0, -0.06}; Ave=-0.0333
Group[7] ={-0.04, -0.04, 0.0}; Ave=-0.0267
Group[8] ={-0.06, -0.03, -0.02}; Ave=-0.0367
Group[9] ={-0.04, -0.04, -0.02}; Ave=-0.0333
Group[10] = {-0.06, 0.0, -0.04}; Ave=-0.0333
Group[11] = {0.0, -0.02, -0.04}; Ave=-0.02
Group[12] = {-0.06, -0.04,0.0 }; Ave=-0.0333
Group[13] = {-0.06, -0.04, -0.02}; Ave=-0.04
Group[14] = {-0.06, -0.02, -0.04}; Ave=-0.04
Group[15] = {-0.05, 0.0, -0.06}; Ave=-0.0367
Group[16] = {-0.02, -0.04, -0.04}; Ave=-0.0333
Group[17] = {-0.02, -0.02, -0.06}; Ave=-0.0333
Group[18] = {-0.04, -0.06, 0.0}; Ave=-0.0333
Group[19] = {-0.02, -0.06, -0.02}; Ave=-0.0333

Figure 13. Test data of the computation of XAve values for XAve-R control chart.

3.5.3. Actual Application of Instrumentation Testing [10] [12]

As programmer known, instrumentation technology is already used in debugging. However, in software testing, the instrumentation technology is usually synthesized with other methods or processes. The typical example is that the member function of "MessageBox()" is used to display the results of program running. Of course, in order to get the statistical information of program execution, the instrumentation is also usually applied in software testing.

In our project, the instrumentation testing is mainly used as a white-box testing step to utilize the "Grey-box" testing approach in integration testing for GUI software. The details can refer to [9] [12].

3.6. Coordinated Interaction between Testing and Programming

As mentioned, the testing activity should be in accordance with the programming activity for an updated software producer. Consequently, the following will analyze and discuss the coordination between testing activity and programming activity for GUI software, especially for the programming and testing in "Dialogue" GUI.

3.6.1. Good Choice of Input Controls

In programming of GUI unit, if you choose reasonable input controls, not only will it improve the efficiency of data inputting or gathering and the usability of this GUI unit, but also the number of test case will be decreased for GUI unit testing.

For example, as shown in **Figure 14**, the graduate course sheet used a "Combo Box" input control for "Nature Science Group" item to substitute three "Check Boxes" *i.e.* "Check Boxes"—"Writing of Sci. & Tech.", "Check Boxes"—"Experimentation" and "Check Boxes"—"Lecture". Obviously, this choice improved the briefness of GUI, input efficiency of data and the usability of software unit. At the same time, the number of test case in unit testing was decreased. Considering three list items in one "Combo Box", there are 3 test cases for one "Combo Box", otherwise, there are 3*3*3 = 27 test cases in function testing for nine "Check Boxes". In this case, it means that the software testing process using "Combo Box" will save about 27/3 = 900% testing time with less test cases.

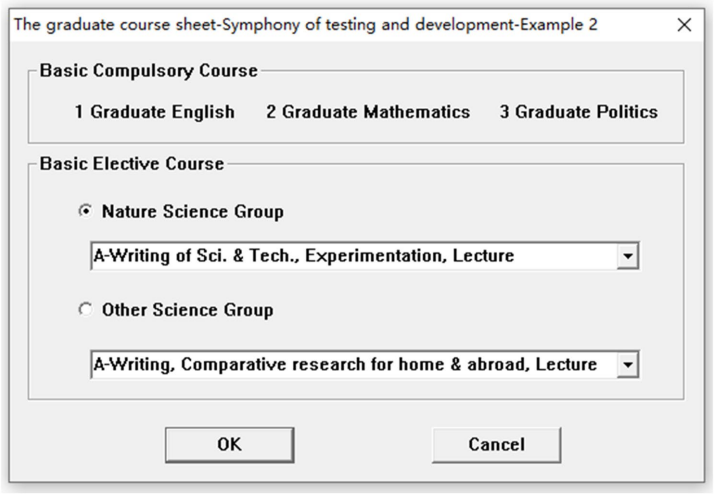

Figure 14. The graduate course sheet.

3.6.2. Good Combination of Input Controls

Similarly, programmer adopts good combination of input controls for data and information gathering in GUI unit, not only can increase the usability of software, but also it can improve the efficiency of software testing with less test cases.

As shown in Figure 10 above, in the sheet of basic setting in quality control-updated, if the combined input control of "Edit Box" and "Spin Button" in "Sample volume" item is instead by only one "Edit Box", the input of "Sample volume" will be not convenient. Moreover, the test cases will also be more. As a result, the test cases will be saved from 15 to 13 in function testing, more details can refer to "3.4.2 Example 2".

3.6.3. Good Layout of Input Controls

As a consequence, programmer construct input controls in layout of GUI unit with good manner, e.g. more carefulness etc., it also will get good benefit in the process of software testing.

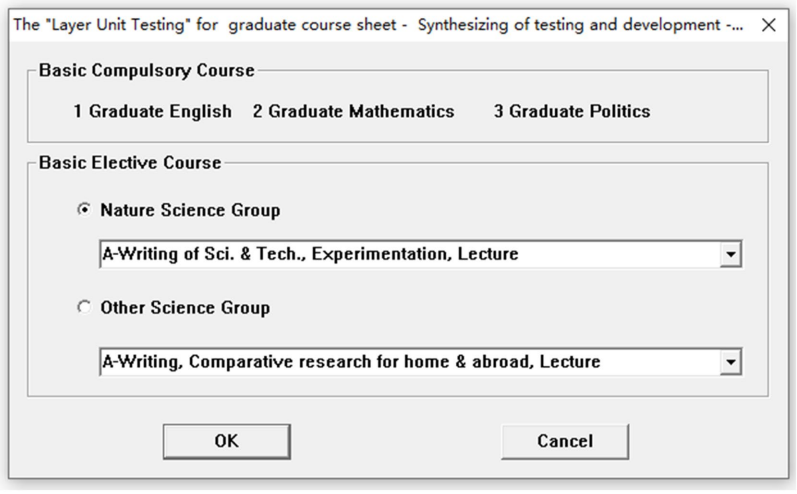

Figure 15. The redundant space status of graduate course sheet.

In Figure 15, because "Combo Box" input controls have redundant width, the boundary testing must check the exception handling under redundant characters input, *i.e.* 61 characters of "A-Writing, Comparative research for home & abroad, Lecture ...", rather under less input for specification. Obviously, the workload with redundant width has been increased for software testing.

3.6.4. Adding Input Controls

Farther, in order to get good GUI, programmer add necessary input controls, it will also improve the testing process with good display for data-input, data-processing and data-result.

Figure 16 is the sheet of product or class in PQMS2, which gave basic product and class data in a manufacturing factory. In this GUI unit, the below part is the area of input data—mainly using "Edit Box" and "Combo Box", and the above part is the display area of input data—using "List Control". For this kind of GUI, the above table is a good layout for displaying results for inputting, modification and deleting. Of course, this table has increased the usability of user. At the same time, it has also improved the unit testing because the table can in time display the testing result of input test data.

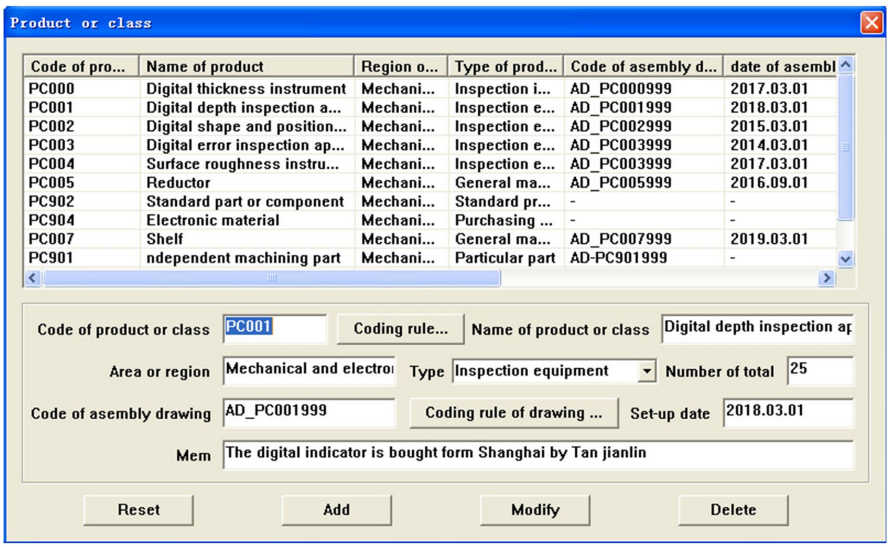

Figure 16. The product or class sheet in PQMS2.

4. Summary and Conclusions

Many programmers might indulge in routine coding, but do you really believe your codes meet the requirements of users and customers? This issue must use particular testers to answer and keep friendly cooperation with each other in GUI unit testing. Consequently, how can the strategy and methodology of GUI unit testing be investigated? At first, strategy and methodology of unit testing must generally be considered to improve testing efficiency and assure the software quality. At the same time, testers are the carrier and executor of the software unit testing, so the testing organization, *i.e.* organizing problem of the group engaging in

software unit testing, must be considered as an inevitable problem. Of course, the update testing organization should be high-efficient and effective, and task-oriented dynamic type instead of complex multilevel structure. Besides testers, arranging the testing process and activity are the center work to achieve testing task success, and this process should be cooperated with developing process. In order to achieve the software unit testing task, tools and framework are very necessary to accelerate testing process and decrease labour of tester. At the same time, the GUI unit testing should differ from the traditional unit testing, how to capture the key and significant feature and part of GUI, and effectively execute the GUI-oriented unit testing are the key points for the whole testing activity.

Thus, as the initial, primary and basic step of software testing, unit testing should have a good organization for GUI-oriented software testing, and the "Dynamic Pair-wise" mode is fitted to medium and small testing organizations especially in China. That's no denying that, the incremental unit testing strategy is usable for unit testing with reasonable arrangement of testing cycle, especially for fast iterative developing. More importantly, the "Layer unit testing" strategy and methodology can be taken into account for GUI-oriented software testing including unit testing, and "Triple-step method" is valuable to execute GUI unit testing with regard of input "parameters combination" and input "parameters independence". Additionally, coordinated interaction between testing and programming is very helpful for software production including the progress of GUI-oriented software testing.

Conflicts of Interest

The authors declare no conflicts of interest regarding the publication of this paper.

References

[1] Runeson, P. and Höst, M. (2008) Guidelines for Conducting and Reporting Case Study Research in Software Engineering. *Empirical Software Engineering*, **14**, 131-164. https://doi.org/10.1007/s10664-008-9102-8

[2] Li, F. (2016) Software Testing Technology. China Machine Press.

[3] Myers, G.J. (1979) The Art of Software Testing. John Wiley and Sons Inc.

[4] Fu, B. (2014) Course of Software Testing. Tsinghua University Press.

[5] Tamres, L. (2002) Introducing Software Testing. Pearson Education Inc.

[6] Patton, R. (2006) Software Testing. 2nd Edition, Pearson Education Inc.

[7] Boehm, B.W. (1981) Software Engineering Economics. Prentice-Hall.

[8] TanLi, M., Zhang, Y. and Wang, Y.L. (2022) Architecture and Methodology of Unit Testing Embedding Pair-Wise Mode for Small Team. *Journal of Software Engineering and Applications*, **15**, 385-405. https://doi.org/10.4236/jsea.2022.1511022

[9] Tanli, M., Xiao, J. and Zhang, Y. (2023) Strategy and Methodology of Integration Testing for GUI Software. *Journal of Software Engineering and Applications*, **16**, 361-396. https://doi.org/10.4236/jsea.2023.168019

[10] Tanli, M., Xiao, J. and Zhang, Y. (2023) Guideline of Test Suite Construction for GUI Software Centered on Grey-Box Approach. *Journal of Software Engineering and Applications*, **16**, 113-143. https://doi.org/10.4236/jsea.2023.165007

[11] Tanli, M., Zhang, Y., Jiang, Y., *et al.* (2021) Baseline Test Suite Construction of Smoke Test for Extreme Programming. *Proceedings of* 2021 *International Conference on Communication Engineering and Logistics Management*, Changsha, 24-26 July 2021, 151-158.

[12] Tanli, M., Zhang, Y., Wang, Y. and Jiang, Y. (2021) Grey-Box Technique of Software Integration Testing Based on Message. *Journal of Physics*: *Conference Series*, **2025**, Article ID: 012096. https://doi.org/10.1088/1742-6596/2025/1/012096

[13] Alégroth, E. and Feldt, R. (2017) On the Long-Term Use of Visual Gui Testing in Industrial Practice: A Case Study. *Empirical Software Engineering*, **22**, 2937-2971. https://doi.org/10.1007/s10664-016-9497-6

[14] Memon, A.M. (2002) GUI Testing: Pitfalls and Process. *Computer*, **35**, 87-88. https://doi.org/10.1109/mc.2002.1023795

[15] Tang, D., TanLi, M., Jiang, Y., Wan, X. and Peng, R. (2019) Product Quality Monitoring of Shewhart Chart Based on Function Integration for Manufacturing Factory. *Journal of Physics*: *Conference Series*, **1302**, Article ID: 042044. https://doi.org/10.1088/1742-6596/1302/4/042044

[16] Tanli, M., Zhang, Y. and Wang, Y.L. (2020) Research on Fault Tree Technique in Software Regression Testing. *Computer Engineering and Software*, **41**, 5-8, 25.

[17] Strandberg, P.E. (2018) Automated System Level Software Testing of Net-Worked Embedded Systems. Licentiate Theses, Mälardalen University Press.

[18] Chen, Z.H. (2005) Research and Implementation of Test Method in Task Arrangement of Resource Satellite. *Radio Engineering*, **35**, 62-64.