Journal of Software Engineering and Applications, 2022, 15, 385-405
https://www.scirp.org/journal/jsea
ISSN Online: 1945-3124
ISSN Print: 1945-3116

Architecture and Methodology of Unit Testing Embedding Pair-Wise Mode for Small Team

Mengqing TanLi[1], Ying Zhang[2], Yulin Wang[2]

[1]School of Software, University of South China, Hunan, China
[2]School of Mechanical Engineering, University of South China, Hunan, China
Email: TLMQ-TEAM@163.com

How to cite this paper: TanLi, M.Q., 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

Received: October 19, 2022
Accepted: November 27, 2022
Published: November 30, 2022

Abstract

In this paper, the new organization for unit testing embedding pair-wise mode is proposed with the core thought focused on the cooperation of programmer and tester by "cross-testing". The typical content of unit testing for the new organizing mode should have three aspects, including self-checking, cross-testing and independent-testing. For cross-testing, executing "pair-wise" mode, mainly tackles data testing, function testing and state testing, which function testing must be done by details and state testing must be considered for completeness. With the specializing of independent-testing, it should be taken as more rigid testing without arbitrariness. Consequently, strategy and measure are addressed for data testing focusing on boundary testing and function/state testing. And organizing method of procedure and key points of tackling unit testing are investigated for the new organizing mode. In order to assess the validity of our study and approach, a series of actual examples are demonstrated for GUI software. The result indicates that the execution of unit testing for the new organizing mode is effective and applicable.

Keywords

Unit Testing, Organizing Architecture, Pair-Wise Mode, Cross-Testing, Data Testing, Function/State Testing, Boundary Testing, Small Team

1. Introduction and Background

Unit testing is the first stage in software testing activity, and its workload is very heavy, and the test approach is mixed. It is undeniable that exploring an effective organizing mode for unit testing in a small team is a very meaningful work because software companies with a small team for software production have a rate of 75% in China [1] [2] [3] [4].

As a kind of quality assurance measure and a node of software testing, unit testing should act as an important role in basic and verifying working [5] [6] [7] According to the general rule of quality assurance for a product, testing work of the software product may include self-testing, interchanging testing, and special testing [8] [9]. Considering the characteristic of software product, self-testing may focus on the code review of the software, and interchanging testing should apply "cross-testing" under "pair-wise" mode [5], and special testing must be independent for programming activity, and it will control the key quality of software units.

On the other hand, programming is a professional technique activity, and a programmer usually has systematic knowledge and skills about some programming languages and tools. Therefore, in organizing programmer tasks in a programming activity, it should not be according to lonely programming with divisions of the graphic user interface, analysis computation, printing output, etc. Otherwise, it should be done in terms of programming language and tools that the programmer had learned, familiarized and skilled in, especially for a small team [6].

However, in the organizing of software unit testing, the programmer must keep cooperation with the tester while their working procedure and effectiveness should be under necessary supervision, and all testing activity can run smoothly whenever key items should be controlled [5].

Overall, it must be noted that the software test organizing will be taken into account throughout the whole product life, and software quality assurance will present the characteristics of everybody involved [7] [10].

2. Related Literature and Work

For the basic importance and universality of unit testing, Fu Bing [8] has given brilliant descriptions especially to improve the quality of programming code. For data testing of unit testing, Ron Patton [3] has paid close attention to the testing of boundary conditions. For function and state testing of unit testing, Li Fan [9] has demonstrated the coverage problem of "N+1-swith" with examples in detail.

Consequently, the changing organization of software development is proposed by Emil Alegroth and Robert Feldt [4] with the term "Squad", however, the detail organizing architecture of test teams for software testing is lacking especially for a small team in China. Meanwhile, the testing process in software companies is deeply depicted in their case study [4], but the cooperation mechanism of test team is not given under new organization mode [5].

Further, as the key content of software testing activity, the test suite construction should be considered with more concern, so previous work [6] on test suite construction of smoke test has shown the magnificence of data-driven in engineering application software. And a more recent study [11] has provided the constructing method of grey-box for test suite of baseline package.

As a consequence, this paper gives the definition of new organization of "Pair-wise" mode, and proposes its cooperation mechanism focusing on "cross-testing". Of course, test design especially on test suite construction is deeply discussed with strategy and concrete methods as well as some factual examples for GUI software.

3. Architecture of Unit Testing Organizing

3.1. New Organization for Unit Testing Embedding "Pair-Wise" Mode

In order to assure the quality of software product especially the programming quality of software unit and module [8] [9], a new unit testing organization is proposed, and it is a test team whose organizing architecture is based on the co-operation of testing activity and programming activity, and the **Figure 1** [5] [9] has shown the detail. In **Figure 1**, the part that filled with grey depicts the personnel composite of small team. In this organizing mode, programmer and tester should conduct a pair-wise cooperator for unit programming and testing [9], and it is called "pair-wise" mode. For this "pair-wise" mode, when programming of a unit is finished, one programmer, acted as a tester, tests the code submitted by another programmer, and so on. Here, it is noticed that the programmer must submit his code to another programmer who will finish test design and testing execution, while the code is also submitted to testing manager with a copy. Advantages of "pair-wise" mode mainly include: 1) the software quality can be assured more effectively to avoid the mistake of individuals, 2) the testing and developing speed can be accelerated because of cooperating each other with intersection testing, 3) the testing cost may be decreased by finding BUG as early as possible, 4) it is easy to produce an atmosphere of cooperation [10].

Meanwhile, the key testing items of programming codes must be tested by an independent tester according to planning and supervising of the developing/testing manager. In this new organizing architecture, team manager focuses on the drawing and execution of testing standard, testing strategies and methods, besides routine of workload arranging and work checking including key sampling doubling as independent tester. It implies that three or four people may construct a test team with our experience, and this team will run with good efficiency [11] [12].

As such, the testing activity between pair-wise cooperators may be called "cross-testing", and that of acting key testing may be called "independent-testing". Consequently, there are three kinds of testing activity in this new unit test

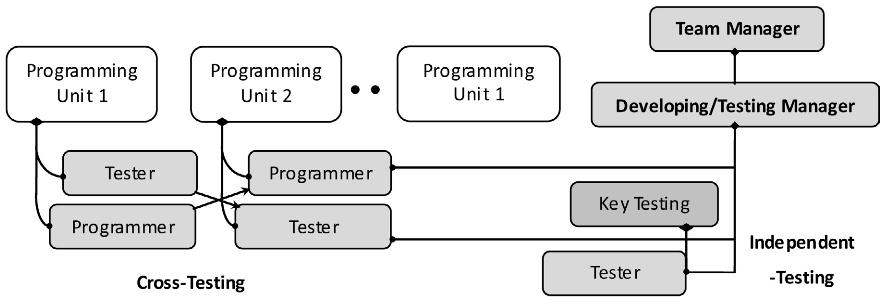


Figure 1. Test team of software unit testing embedding pair-wise mode.

organization, including self checking, cross-testing and independent-testing, which will be discussed in the next section in detail.

3.2. Typical Contents of Unit Testing for New Mode

Typical contents of unit testing for new organizing mode should have three aspects, including self-checking, cross-testing and independent-testing as mentioned above, and its details are shown in Figure 2. In self-checking, it mainly uses code review and "self-constructing test class". For cross-testing, executing "pair-wise" mode, it mainly tackles data testing, function testing and state testing which function testing must be elaborately done for details and state testing must be considered as completely as possible. With the specializing of independent-testing, it should be taken as more rigid testing without arbitrariness. More details are respectively described as follows.

For code review, the programmer must do himself for all finished codes before the unit is submitted for cross-testing or independent-testing. For logic testing, it is must be executed according to actual requirement arranged by manager, either engaging on cross-testing for important unit or executing independent-testing for key unit. In the data testing, keeping more carefully, it must be done in terms of actual status of tested unit. Consequently, in the function and state testing, it should be executed according to factual feature of being tested units and software because the importance of units is distinguished, and it will also be discussed in Section 3.3 in detail. Further, distinguished strategy should be adopted for deferent software and system, and it will also be discussed in

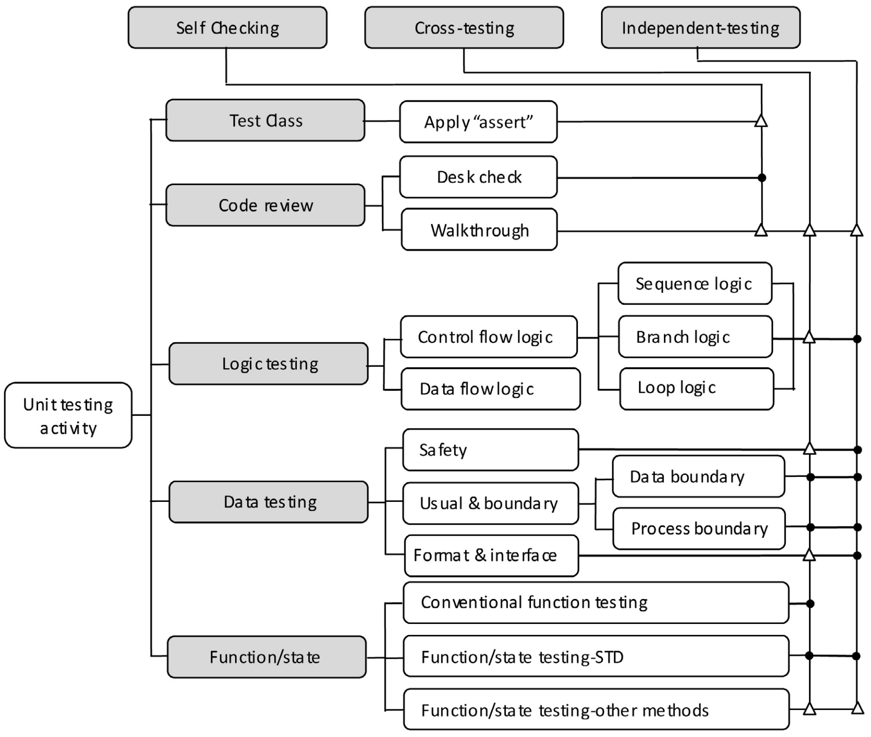


Figure 2. Typical content of unit testing activity.

Section 3.3.

3.3. Strategies and Measures in Unit Testing

Because the workload of unit programming and testing is very heavy and interchanging between programmer and tester repeats very frequently, the efficiency and effectiveness must be taken into account in the strategy and measure of testing activity, which organizing mode and contents have stated in **Figure 1** and **Figure 2**.

3.3.1. Distinguished Strategy of Data Testing

Data testing is the primary affair in unit testing especially for the scenario of GUI software testing with black-box, and the testing item should be chosen as **Figure 2** in terms of the feature of actual software/system. Without loss of systematic unification, actual software system is divided into safety-critical system, general system, and lower system, and the unit of a system has three types, *i.e.* key unit, important unit, and unimportant unit. Thus, execution of data testing should be respectively done according to these actual types. Further, the arrangement of data testing in unit testing must be executed in terms of actual state of software unit, *i.e.*, key and important units should be arranged with more rigid testing, while fewer testing items should be briefly tackled to units for lower system and unimportant units. Moreover, some testing items may be omitted in lower system and unimportant units. The concrete strategy is shown in **Table 1**.

3.3.2. Distinguished Strategy of Function and State Testing

Function and state testing is an ordinary and elaborated work in unit testing. Based on "Sheet and Form" of GUI software and system, the function and state may be numerous for a sheet and form, and the function and state testing must be kept on careful status and sustainable sense throughout the proceeding.

For function testing, doing elaborately will be the best criterion in test design all the time. That is to say, every transformation of every state must be tested. Especially to deal with "error and exception state", any error and exception state should not be omitted or leaked.

For state testing, keeping completeness will be basic requirement in the process of test design. In the disposing procedure, the "return state" should be

Table 1. The choice of testing items of data testing for deferent software/system.

Testing items	Safety-critical system			General system			Lower system	
	KU[a]	IU[b]	UU[c]	KU	IU	UU	IU	UU
Data boundary	○[d]	○	○	○	○	Δ[e]	○	[f]
Process boundary	○	○	Δ	○	○	Δ	○	
Format & interface	○	○	○	○	Δ		Δ	
Safety	○	○	○	○	Δ		Δ	

[a]KU-Key unit, [b]IU-Important unit, and [c]UU-Unimportant unit. [d]"○" presents an item that it must be done, [e]"△" implies an alternative item. and [f]the blanket is an item unexpected to do.

paid more attention in state testing, and all state of program execution must be considered including normal and abnormal state. Of course, some states can be aggregated if it is necessary. Without loss of briefness, the first stage of test design for state testing should be drawing of STD (State Transform Diagram). In the state transform diagram, all states must be divided correctly and precisely, and each label and each side must be described clearly and completely.

3.4. Procedure Organizing Method of Unit Testing Activity

There are four aspects in the cycle of unit testing activity as shown in **Figure 3**, including test design, testing execution, test record/report, track and statistics. It is an undeniable fact that the test design is the best important aspect, which the workload has occupied 60% of software testing [10]. And testing execution will determine the actual result of software testing and it is also the best concrete work. At the same time, test record and report is basic work, by which the saving file is taken as the important accordance and source for management and control of software testing activity. As a consequence, testing track and statistics will affect the derivation of software testing and the effectiveness of monitoring or supervising of whole life cycle of software testing activity.

As depicted above, the new organization embedding "pair-wise" mode can be taken as a general arranging methodology; the detailed procedure of unit testing can also be described by embedding pair-wise mode shown in **Figure 4**. The central part of **Figure 4** with grey filling will be the key part for testing activity, and dependency analysis will be referred to incremental testing while test suite is the important carrier of test case package. On the left of **Figure 4**, there are a number of items that will be the supporting materials for test design and testing execution in unit testing activity. 1) Testing standard—it is the criteria for test design, testing execution, test record/report, testing track and statistics. 2) Testing strategy—it mainly refers to reasonable and effective disposal of whole testing activity, including planning, schedule, controlling, monitoring and so on. 3) Testing method—including general and conventional testing method and special and particular testing approach, including grey-box testing and mutation

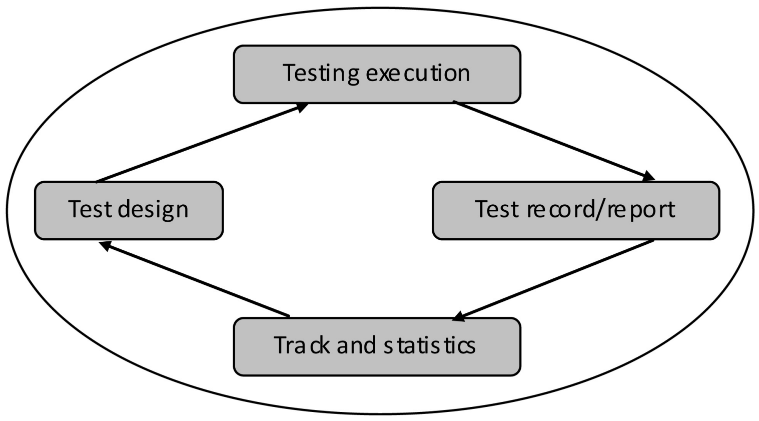


Figure 3. The typical cycle of unit testing activity.

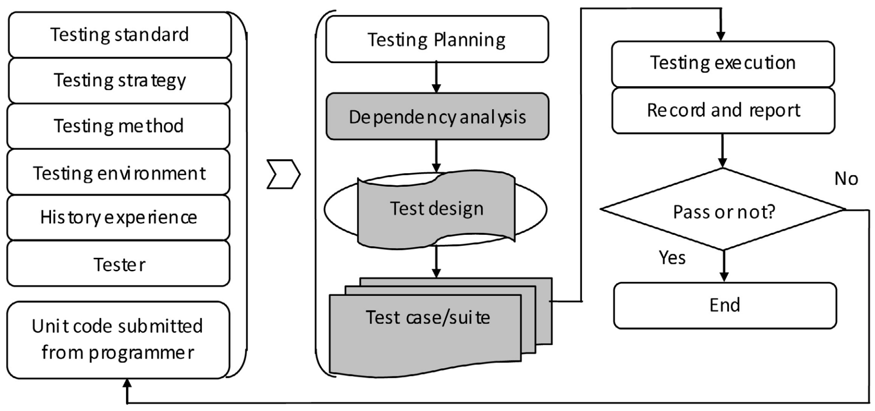


Figure 4. Procedure of unit testing embedding pair-wise mode.

testing approach, etc. 4) Testing environment—including hardware and supporting software and system. 5) History experience—mainly referring to the experience of analysis, test design, test execution, BUG track and so on. 6) Tester—is also the affecting factor for testing organizing mainly considering their knowledge and skills. Without deniable fact, the process of testing activity will be continuously repeated when the testing result is "no pass" and the program and code is returned to programmer, and this status is stated in the tail of **Figure 4**.

4. Key Organizing Method of Unit Testing

4.1. Start Point of Unit Testing for GUI Software

For GUI software, we have found that the "Sheet/Form" can be taken as the basic unit and start point for software unit testing, and it has been verified by factual practice including grey-box testing [11], in which all "Controls" in a sheet should be considered as a whole subject together even if there are data exchanging each other. However, following aspects must be paid attention, including 1) the programmer should finish all tasks of GUI layout and coding of a "Sheet/Form" before it is submitted to cross-testing. 2) All code of event disposing of the "Sheet/Form" must have been debugged and have implemented their function. 3) All programs must be reviewed with desk check by the programmer himself, including obligated to the criteria and unification of coding. 4) For all code, the exception handling should be considered as completely as possible.

4.2. Activity Organizing of "Cross-Testing" and "Independent-Testing"

As shown in **Figure 1**, cross-testing is the primary and central work in unit testing which workload is very heavy and work interchanging occurs with high frequency. The organizing of cross-testing should consider the following requirements.

- Test design must keep the most optimal position and cooperation with programming activity.
- Organizing cross-testing should be prior in order to improve efficiency.

- The programmer must submit a copy of code to manger with good notes when it is submitted to testing.
- In the execution of cross-testing, programmer could not alter his code before the testing result was returned.
- The tester must do his testing task in real earnest and give the real testing result according to the programmer's code while tester may give some reasonable suggestion if necessary.
- If there are some questions and disputes about testing result, the programmer must report to the manger and executes the programming task in terms of manager's approval.
- Tester must seriously record the testing result and save the documentation in safety.

However, independent-testing should not be omitted by testing activity. To some extent, the requirement of independent-testing in testing technique and skill is more serious and rigid, e.g., some contents of safety testing. At the same time, the coding quality of key unit and module must be controlled in independent-testing by particular tester, and it is also the requirement of good quality system to assure the correctness and high performance of software code.

4.3. Time-Axis of Testing-Programming Organizing in "Cross-Testing"

For testing activity, good schedule is the primary condition for testing organizing. Figure 5 has illustrated time axis of general testing activity for "cross-testing". In Figure 5, the below part of figure is the testing activity organizing, and the up part is the programming organizing. In order to keep accordance of progress in testing-programming activity, two methods of "idle-waiting manner" and "parallel-disposing manner" can be applied. And the term "waiting or programming" in Figure 5 implies that two disposing manner are adopted, *i.e.*, one programmer waiting is executed in "idle-waiting manner" as a tester, and programming or testing for two programmers or testers is concurrently done in "parallel-disposing manner".

4.4. Typical Application of "Idle-Waiting" & "Parallel-Disposing"

As a consequence, there are two methods to tackle cross-testing in test team, that is, "idle-waiting manner" and "parallel-disposing manner" as mentioned above.

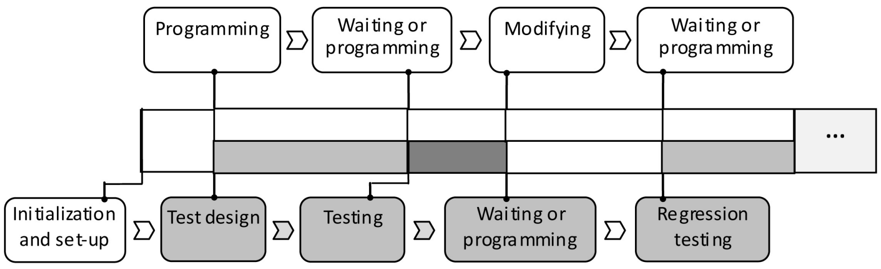


Figure 5. Procedure time axis of general testing activity for cross testing.

If the gap between the programming time and testing time is not very long, the "idle-waiting manner" should be applied. Otherwise, if the gap between the programming time and testing time is very long and programming time is undecided for programming activity, the "parallel-disposing manner" could be used.

For "idle-waiting manner", when a programmer engages into his programming working, another programmer may settle his hands as follows: 1) having a cup of coffee in the resting room, 2) talking about each other with his working fellows in network for technique issues, 3) enjoying himself with music and so on, 4) and doing testing work later at night etc.

For "parallel-disposing manner", when a programmer has not finished his unit programming, another programmer should continuously do his own programming work or do test design of other parts. Of course, for a newer of joining team, he may learn something from the programmer who is programming for a key algorithm or interesting code segment, etc.

4.5. Using of Improved STD for Function and State Testing [3]

4.5.1. Improved STD for GUI Software Testing

Facing particular features of GUI software, the improved STD is proposed. The distinguishing of improved STD includes three aspects. 1) The symbol "●" presents the start point of diagram, because a concrete control and event are existed to start for a state transforming in the GUI software, 2) the end point of diagram is labeled by symbol "⊙" for its graphical representation and expressiveness, and 3) the synthesis of many same or similar states is reasonably done.

4.5.2. Testing Process Based on Improved STD

The procedure of function and state testing based on improved STD could be given as follows.

- A programmer, acted as a tester, receives the finished programming code from another programmer, and does necessary analysis including FTA (Fault-Tree Analysis).
- According to software function, test design is firstly performed for detail divided functions, and taken care for the respective disposing based on factual running and exception handling.
- In terms of Section 3.5.1 above, the improved STD is drawn for the actual software.
- Based on the finished STD, test cases for state testing are consequently designed.
- Executing test case or test suite, and saving testing records and reporting BUG after testing.
- Doing statistical analysis, and tracking BUGs.

5. Examples of Unit Test Activity in Small Team

In unit testing activity as shown in Figure 2, data testing and function/state testing are main testing work, while logic testing is not required for general unit except key or important unit of software system. Of course, self-checking must be finished before data testing and function/state testing are assigned. For data testing, data value boundary testing and process boundary testing are key items firstly, and sample testing is an important method. For function/state testing, the improved STD is a key tool with good expressiveness. At first, some typical disposing methods for unit test activity are briefly introduced as follows.

5.1. Typical Disposing Methods for Unit Test Activity

Besides data testing according to general conventional approach, the sampling testing may be applied for testing of data value boundary and process boundary control in unit testing.

5.1.1. Typical Disposing Method of Data Value Boundary of "Input Control's"

Taking for example with Visual C++, sampling test of boundary value can be disposed as follows.

1) Whatever kind of "Input Control", if the boundary value is not given according to the requirement, it should be certificated and signed with "NO PASS".

a) CString type data etc.—Input character "T……" by keyboard with limited number of designed requirement plus one, and stop when listened to the "Da" voice. Then, if the number of characters input is equal to the design limit, "PASS" can be certificated. Otherwise, "NO PASS" should be signed.

b) Integer type data etc.—It must adopt the method of program executing. Inputting digital "1……" by keyboard with up limited value of designed requirement adding "1" and low limited value of designed requirement subtracting "1". Executing the program, if the error prompt information is displayed, "PASS" can be certificated. Otherwise, "NO PASS" should be signed.

c) Float type data etc.—It must adopt the method of program executing. Inputting data "xxxxxx.x" by keyboard with up limited value of designed requirement adding one accuracy unit and low limited value of designed requirement subtracting one accuracy unit. Executing the program, if the error prompt information is displayed, "PASS" can be certificated. Otherwise, "NO PASS" should be signed.

2) Sampling test for "Input Control" of "ComBoBox/ComBoBoxEx".

a) CString type data etc.—including two situations—Situation 1 is that layout size of "Input Control" is smaller than designed requirement and situation 2 is that layout size of "Input Control" is larger than designed requirement. For the former situation, Inputting character "T……" by keyboard with limited number of designed requirement plus one, and stop when listened to the "Da" voice. Then, if number of characters input is equal to the design limit, "PASS" can be certificated. Otherwise, "NO PASS" should be signed. For the later situation, it must adopt the method of program executing like Integer type data of "Edit/Rich Edit".

b) Integer type data etc.—it is the same as "Input Control" of "Edit/Rich Edit".

c) Float type data etc.—it is the same as "Input Control" of "Edit/Rich Edit" too.

5.1.2. Typical Disposing Method of Process Boundary

In the practice of software product running, many accidents emerged referring to boundary control which had brought about a lot of loss. How to tackle and avoid this kind of software BUG, it has become a serious problem.

As such, disposing measures of process boundary can be generally given as follows.

1) If the control boundary is the status of the usual data which user can operate, the boundary value of control object must be set.

2) If the control boundary can be transformed into usual data as 1), it should be deal with as usual data after transforming.

3) If the control boundary is the status of varied data which user is blind, it should be tackled in primary setting.

4) For tackling boundary control and its limit, the status of extra-condition and network attack should be considered especially for DB of distributed network.

Factual status of software is distinguished and varied with version upgrade, and the unit testing of software will have respective features. However, the unit testing includes generally two aspects—data testing and function/state testing. Further, due to generality of GUI software in desktop system, the following will investigate unit testing with factual examples in PQMS2 (Product Quality Monitoring System version 2.0) in detail.

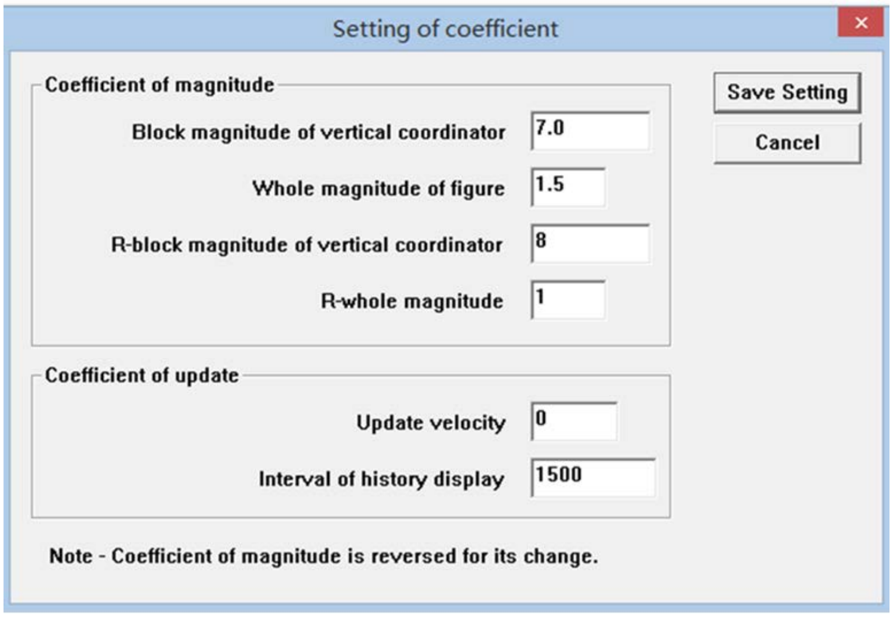
5.2. Data Testing Examples Focusing on Boundary Testing

As shown in **Figure 2**, data testing mainly includes boundary testing, format and interface testing, and safety testing, etc. However, for GUI software, format and interface control can be generally assured by DBMS, and safety testing should be arranged for system testing in particular e.g., leakage testing of memory, SQL injection testing, etc. As a consequence, unit data testing may focus on boundary testing.

5.2.1. Boundary Testing of Key Unit with "Input Control's" - Locally Important

As we all know, for the key unit especially that of safety-critical system, the method of sampling test should not be adopted and full testing is obliged. In PQMS2, the unit of setting coefficient is a key unit of Median chart [13], as shown in **Figure 6**, because its efficacy is significant for the functional running and displaying effectiveness in Median chart. Thus, full testing must adopted in this unit for boundary testing.

In **Figure 6**, there are six "Input Control's for coefficients gathering, and all of those must be tested, instead of sampling test, with data boundary value. Without much difficulty, the test method is consequently according to Section 4.1.1. Thus, four EditBoxes of coefficient of magnitude should be verified in terms of


Setting of coefficient

Coefficient of magnitude

Block magnitude of vertical coordinator: 7.0

Whole magnitude of figure: 1.5

R-block magnitude of vertical coordinator: 8

R-whole magnitude: 1

Coefficient of update

Update velocity: 0

Interval of history display: 1500

Note - Coefficient of magnitude is reversed for its change.

Figure 6. GUI of setting coefficient of median chart in PQMS2.

"Float data" value and two EditBoxes of coefficient of update should be verified according to "Integer data" value. And detail procedure may be depicted as follows,

1) For EditBoxes of "Float data" value, *i.e.*, "Block magnitude of vertical", "Whole magnitude of figure", "R-block magnitude of vertical" and "R-whole magnitude of figure"—Input max value "1000.01" and min value "0.00" by keyboard with accuracy "0.01" in the GUI "Form/Sheet", and execute the program, if the error prompt information is displayed, "PASS" can be certificated. Otherwise, "NO PASS" should be signed.

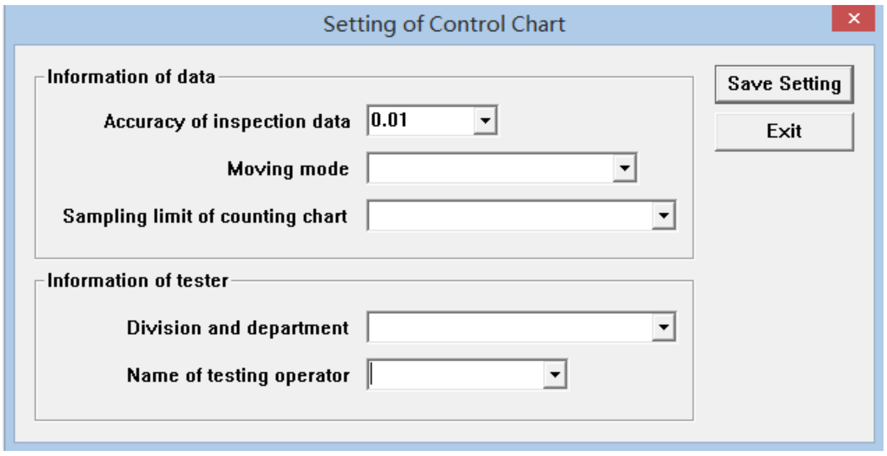
2) For EditBox "Update velocity" with "Integer data" value—Input max value "10,001" and min value "−1" by keyboard with accuracy "1", and execute the program, if the error prompt information is displayed, "PASS" can be certificated. Otherwise, "NO PASS" should be signed.

3) Similarly, for EditBox "Interval of history display" with "Integer data" value—Input max value "100,001" and min value "999" by keyboard with accuracy "1", and execute the program, if the error prompt information is displayed, "PASS" can be certificated. Otherwise, "NO PASS" should be signed.

5.2.2. Boundary Testing of Key Unit with "Input Control's"—Whole Important

As a consequence, the approach of sampling test should not be applied for "Input Controls" of key units with whole importance, such as the setting unit of control chart [13], because it very important to other input components and the whole system in PQMS2. And the GUI of the setting unit of control chart is shown in Figure 7.

Thereby, all "Input Controls" of the setting unit of control chart must be tested for boundary value. Similarly, these five ComBoBoxes should be verified


Setting of Control Chart

Information of data

Accuracy of inspection data: 0.01

Moving mode:

Sampling limit of counting chart:

Information of tester

Division and department:

Name of testing operator:

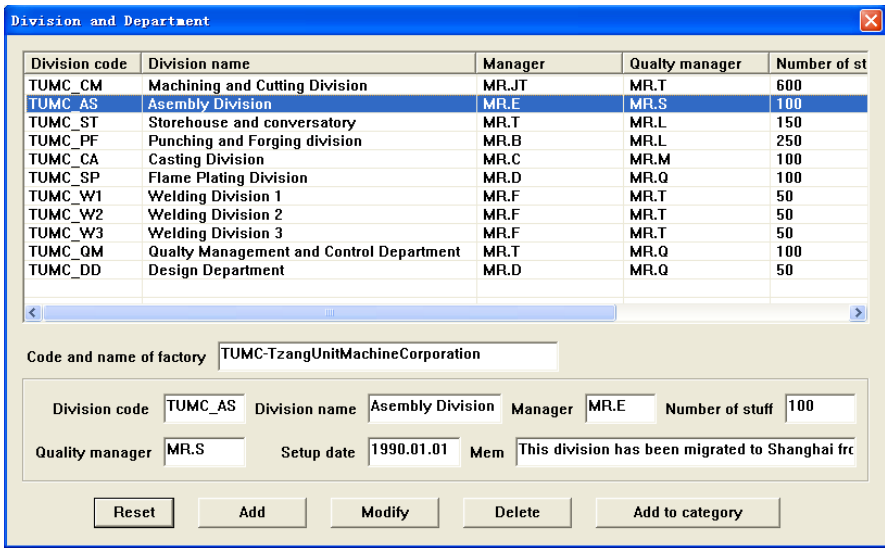
Figure 7. GUI of setting of control chart in PQMS2.

in terms of "CString data" value. *i.e.*, (1) for ComBoBox "Accuracy of inspection data"—Input character "T……" by keyboard with limited 7 times, and stop when listened to the "Da" voice. Then, if the number of characters input is equal to 6, "PASS" can be certificated. Otherwise, "NO PASS" should be signed. (2) For ComBoBox "Moving mode"—Input continuously character "T……" by keyboard with limited 51 times, and stop when listened to the "Da" voice. Then, if the number of input characters is equal to 50, "PASS" can be certificated. Otherwise, "NO PASS" should be signed. (3) For other three ComBoBoxes "Sampling limit of counting chart", "Division department", "Name of testing operator"—the disposing is the same as the ComBoBox "Moving mode".

5.2.3. Boundary Testing of Popular Unit with "Input Control's"

For GUI software, a lot of "Form/Sheet" exists as popular units for general data input and function handling, and the sampling testing may be used in this kind of unit testing activity. As a popular unit, the GUI unit of factory division [14] in PQMS2 is shown in Figure 8. Because it is not a very important unit, the sampling testing method may be applied for testing of data boundary value in this unit.

As such, considering the significance of "Input Control" itself, three "Input Control's" should be tested by sampling for boundary value, *i.e.* EditBox "Code and name of factory", EditBox "Division code" and EditBox "Division name". According to the test method above proposed, sampling test for these three EditBoxes with "CString data" value should be adopted, *i.e.*, (1) for EditBox "Code and name of factory"—Input character "T……" by keyboard with limited 51 times, and stop when listened to the "Da" voice. Then, if the number of input characters is equal to 50, "PASS" can be certificated. Otherwise, "NO PASS" should be signed; (2) for EditBox "Division code"—Input character "T……" by keyboard with limited 21 times, and stop when listened to the "Da" voice. Then, if the number of input characters is equal to 20, "PASS" can be certificated. Otherwise, "NO PASS" should be signed; (3) for EditBox "Division name"—the same as EditBox



Division code	Division name	Manager	Qualty manager	Number of st
TUMC_CM	Machining and Cutting Division	MR.JT	MR.T	600
TUMC_AS	Asembly Division	MR.E	MR.S	100
TUMC_ST	Storehouse and conversatory	MR.T	MR.L	150
TUMC_PF	Punching and Forging division	MR.B	MR.L	250
TUMC_CA	Casting Division	MR.C	MR.M	100
TUMC_SP	Flame Plating Division	MR.D	MR.Q	100
TUMC_W1	Welding Division 1	MR.F	MR.T	50
TUMC_W2	Welding Division 2	MR.F	MR.T	50
TUMC_W3	Welding Division 3	MR.F	MR.T	50
TUMC_QM	Qualty Management and Control Department	MR.T	MR.Q	100
TUMC_DD	Design Department	MR.D	MR.Q	50

Code and name of factory: TUMC-TzangUnitMachineCorporation

Division code: TUMC_AS Division name: Asembly Division Manager: MR.E Number of stuff: 100

Quality manager: MR.S Setup date: 1990.01.01 Mem: This division has been migrated to Shanghai frc

Figure 8. GUI of factory division in PQMS2.

"Code and name of factory". Consequently, other five EditBoxes are omitted to verify the boundary value.

5.3. Data Testing/Function Testing Example of Key Unit with Traditional Method

For key unit testing embedding pair-wise mode, the logic testing of control flow may be used for key unit with special requirement [8]. In logic testing of control flow, there are three main kinds, *i.e.*, sequential logic, branch logic and loop logic. The following will demonstrates the traditional method for branch logic testing because branch logic error is easy-to-occur BUG in factual software application.

In PQMS2, the computation of Median chart is a main part for Median chart application, and choice and computation of control chart parameter is a key logic processing unit. The control flow diagram of computation of control chart parameter of Median chart is shown in Figure 9, and its unit testing is a typical branch logic testing with traditional method. According to usual testing method of branch logic, test cases can be given as list in Table 2. We can find, in Table 2, that this kind of branch logic testing is not very difficult except in a responsible and careful manner.

5.4. Function and State Testing Examples for GUI Software

5.4.1. Function and State Testing of Key Unit for Basic "Form/Sheet"—Locally Important

As shown in Figure 6 above, for function and state testing, the unit of setting coefficient of the Median chart in PQMS2 is relatively simple because fewer functions and states are tackled.

According to the test procedure of GUI software based on "Form/Sheet", the function of data saving function of setting coefficient "Form/Sheet" should be tested firstly, and its test case is executed as shown in Table 3 [9].

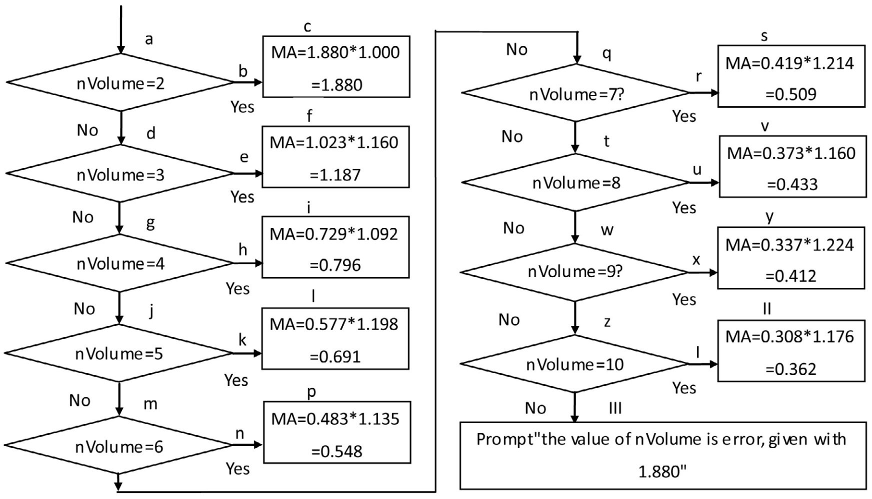


Figure 9. The control flow diagram of parameter computation of control chart.

Table 2. Test case of parameter computation of control chart in Median chart.

ID[a]	Branch	Input	Expected output
TC020	a-b-c	nVolume = 2	MA = 1.880
TC021	a-d-e-f	nVolume = 3	MA = 1.187
TC022	a-d-g-h-i	nVolume = 4	MA = 0.796
TC023	a-d-g-j-k-l	nVolume = 5	MA = 0.691
TC024	a-d-g-j-m-n-p	nVolume = 6	MA = 0.548
TC025	a-d-g-j-m-q-r-s	nVolume = 7	MA = 0.509
TC026	a-d-g-j-m-q-t-u-v	nVolume = 8	MA = 0.433
TC027	a-d-g-j-m-q-t-w-x-y	nVolume = 9	MA = 0.412
TC028	a-d-g-j-m-q-t-w-z-I-II	nVolume = 10	MA = 0.362
TC029	a-d-g-j-m-q-t-w-z-III	nVolume = 1	Prompt information[b]
TC030	a-d-g-j-m-q-t-w-z-III	nVolume = 0	Prompt information
TC031	a-d-g-j-m-q-t-w-z-III	nVolume = −1	Prompt information
TC032	a-d-g-j-m-q-t-w-z-III	nVolume = −20	Prompt information
TC033	a-d-g-j-m-q-t-w-z-III	nVolume = 11	Prompt information
TC034	a-d-g-j-m-q-t-w-z-III	nVolume = 50	Prompt information

a. the front part of all ID are omitted with "PQMS2-MC0-UNI-", b. the prompt information is "the value of nVolume is error, given with 1.880".

For state testing of setting coefficient "Form/Sheet", the improved STD (State Transform Diagram) should be drawn correctly and completely [3]. As our test approach, the improved STD of setting coefficient in Median chart has been finished as shown in **Figure 10**. Consequently, in terms of the result of improved STD, test cases of the unit of setting coefficient of Median chart can be finished as test design, and the result is shown in **Table 4**.

Table 3. Test case of data saving function of setting coefficient sheet in Median chart.

ID	Input	Expected output
PQMS2-MCF-UNI-TC020	Start setting coefficient sheet from Median chart application program, input "7.0" in EditBox "magnitude coefficient of vertical coordinator", input "1.5" in EditBox "magnitude coefficient of Median figure", input "8.0" in EditBox "R magnitude coefficient of vertical coordinator", input "1.0" in EditBox "magnitude coefficient of R figure", input "0" in EditBox "coefficient of update", and input "1500" in EditBox "gap of history manifesting", and click Button "Save Setting" finally.	Figure is renewed, display correctly and no prompt of error information is given.

Table 4. Test case of state testing of setting coefficient sheet in Median chart.

ID	Start state	End state	Input	Expected output
PQMS2-MCS-UNI-TC001	S_0	S_1	Click the menu item of "Setting-Coefficient Setting" or toolbar item "Coefficient Setting"	Display the sheet "setting coefficient of Median chart"
PQMS2-MCS-UNI-TC002	S_1	S_4	No input operation, click the button "Cancel" or "Shut off" in the right-up angle	Cancel setting
PQMS2-MCS-UNI-TC003	S_1	S_2	Input data in the sheet	Display data in the sheet
PQMS2-MCS-UNI-TC004	S_2	S_4	Keep default data and click the button "Save Setting"	Save Setting data and no error information prompt
PQMS2-MCS-UNI-TC005	S_2	S_3	Keep blank in all "Input controls" and click the button "Save Setting"	Display error information prompt
PQMS2-MCS-UNI-TC006	S_3	S_4	Click the confirming button of dialogue or button "Shut off" in the right-up angle of dialogue	The idle state
PQMS2-MCS-UNI-TC007	S_2	S_4	When input data, click the button "Cancel" or "Shut off" in the right-up angle of sheet	Cancel setting
PQMS2-MCS-UNI-TC008	S_4	S_5	Click "Shut off" in the right-up angle of sheet	Exit

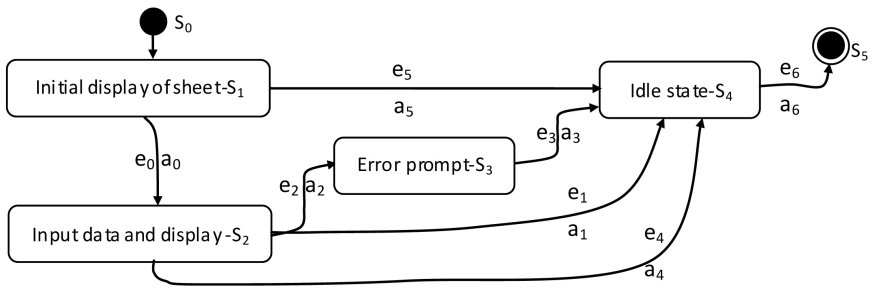


Figure 10. The improved state transform diagram of setting coefficient in Median chart.

5.4.2. Function and State Testing of Key Unit for Basic "Form/Sheet"—Whole Important

Conversely, many key units with basic "Form/Sheet" GUI have much more functions to tackle such as **Figure 8**. The function and state testing for this kind of GUI unit will be more complex and difficult, e.g., the GUI unit of inspection data in PQMS2, which it is a "Form/Sheet" for inputting inspection data and likes **Figure 8** by a bit but it is more complex.

Of course, the function testing should be finished firstly. For function testing, main items include 1) displaying of inspection data sheet with history default data, 2) manually input data, 3) manually modify data, 4) manually delete data, 5) manually save data, 6) load saving data, 7) import digital gauge data, 8) import Notepad data, 9) re-enter data, 10) save all data.

Additionally, for some key units, the member function testing with traditional method including logic testing should be done especially for critical requirement.

For state testing, in order to describe conveniently, we firstly give the improved STD of inspection data "Form/Sheet" shown in **Figure 11** and this improved STD has given the necessary information for test design of state testing. In **Figure 11**, more disposing functions and states mainly include 1) Open & display "Tree-control", the state is presented with S_2. 2) Exception messages are disposed respectively, such as "Exception message of adding—S_{4-1}", "Exception message of modifying—S_{4-2}", "Exception message of deleting—S_{4-3}", "Exception message of input & import—S_{4-4}", "Exception message of saving all—S_{4-5}" etc. 3) Verification and result display are disposed respectively, referring to S_{3-1}, S_{3-2}. As such, test design can be done according to information given in **Figure 11**.

Consequently, in test case design of state testing, all state transforming information in **Figure 11** must be used, and the format of test case is similar to **Table 4**, but the quantity of test case of state testing are more with the volume of 30, which codes are from PQMS2-EDS-UNI-TC010 to PQMS2-EDS-UNI-TCTC039, and its detail contents are abbreviated here.

5.4.3. Function and State Testing of Popular Unit for Basic "Form/Sheet"

The unit of factory division in PQMS2, which the GUI of this unit is illustrated in **Figure 8**, is a typical pattern of popular GUI unit. For this kind of popular GUI unit, the function and state testing should execute the strategy mentioned in Section 3.3.2, *i.e.*, "Sheet and Form" may be taken as an independent unit for function and state testing for GUI software.

It is obvious that functions of displaying, adding, modifying, deleting and re-entering should be tested firstly. Of course, test cases of function testing will be more for popular GUI unit because of more function disposing. Factually, there are 22 test cases in this example with codes of PQMS2-DDF-UNI-TC010-TC031. At the same time, it should be noticed that all these functions must be carefully tested without forgetting, missing and leaking.

Consequently, state testing may be done according to improved STD as shown

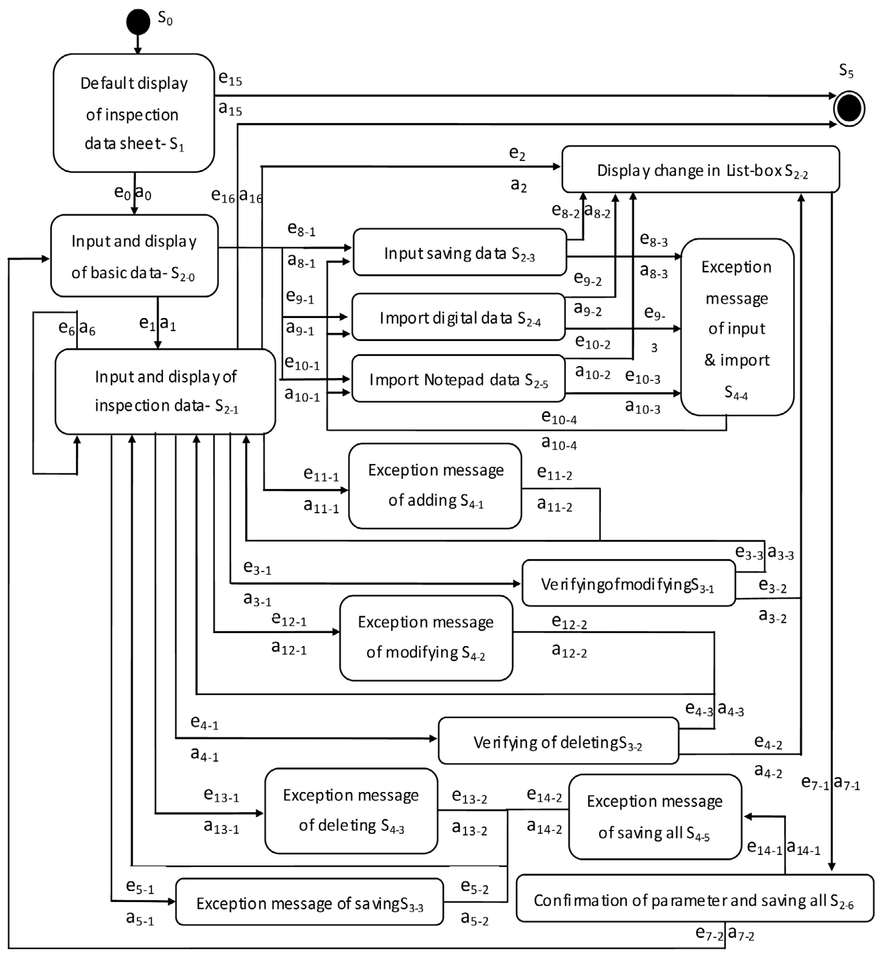

S0
Default display of inspection data sheet- S1
e15
a15
S5
e2
a2
Display change in List-box S2-2
e0 a0
e16 a16
e8-2 a8-2
Input and display of basic data- S2-0
e8-1
a8-1
Input saving data S2-3
e8-3
a8-3
Exception message of input & import S4-4
e9-2
e9-1
a9-1
Import digital data S2-4
a9-2
e9-3
e6 a6
e1 a1
e10-2
Input and display of inspection data- S2-1
e10-1
a10-1
Import Notepad data S2-5
a10-2
e10-3
a10-3
e10-4
a10-4
e11-1
a11-1
Exception message of adding S4-1
e11-2
a11-2
e3-3 a3-3
e3-1
a3-1
Verifying of modifying S3-1
e3-2
a3-2
e12-1
a12-1
Exception message of modifying S4-2
e12-2
a12-2
e4-3 a4-3
e4-1
a4-1
Verifying of deleting S3-2
e4-2
a4-2
e7-1 a7-1
e13-1
a13-1
Exception message of deleting S4-3
e13-2
a13-2
e14-2
a14-2
Exception message of saving all S4-5
e14-1 a14-1
e5-1
a5-1
Exception message of saving S3-3
e5-2
a5-2
Confirmation of parameter and saving all S2-6
e7-2 a7-2

Figure 11. The improved state transform diagram of inspection data in PQMS2.

in **Figure 12**. In **Figure 12**, in order to depict state and state-transform more clearly and to be understood easily [3], two particular states are given for which it is a sub-partitioning for the state of input and display, *i.e.*, “Input and display data in factory division sheet—S_{2-1}” and “Display data variety in list—S_{2-2}”. Similarly, “S_{4-1}, S_{4-2}, S_{4-3}” and “S_{3-1}, S_{3-2}” are also sub-partitioning.

If it is unnecessary, the member function testing with conventional method including logic testing may be omitted for this kind of popular GUI unit. However, if the result of integration testing or system testing [15] indicated that some units have critical BUG, these units must be rigidly tested in regression testing [16] including traditional logic testing, and preferably executing by an independent tester.

5.4.4. Function and State Testing of General Unit without “Input Control”

In GUI software, the execution of some functions do not rely on the “Sheet/Form”, and run by other patterns such as internal computation, database, console, and so on. For this kind of function and state testing, the function testing may be executed in the level of member function testing or even level of code segment of the operation method. However, if testing in low level occurred for key unit, the

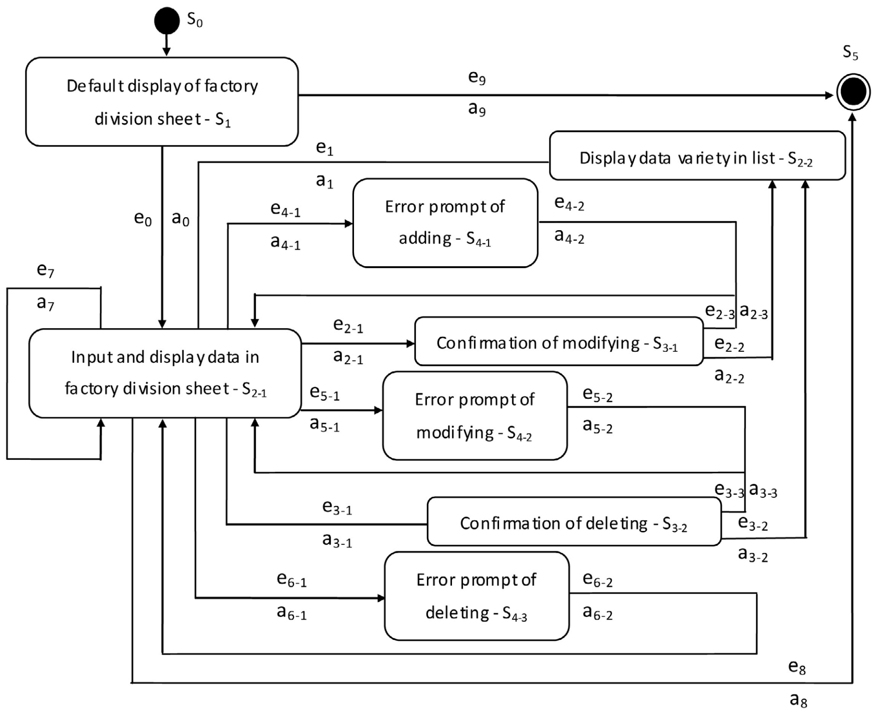

S0
Default display of factory division sheet - S1
e9
a9
S5
e1
a1
Display data variety in list - S2-2
e0
a0
e4-1
a4-1
Error prompt of adding - S4-1
e4-2
a4-2
e7
a7
e2-3
a2-3
Input and display data in factory division sheet - S2-1
e2-1
a2-1
Confirmation of modifying - S3-1
e2-2
a2-2
e5-1
a5-1
Error prompt of modifying - S4-2
e5-2
a5-2
e3-3
a3-3
e3-1
a3-1
Confirmation of deleting - S3-2
e3-2
a3-2
e6-1
a6-1
Error prompt of deleting - S4-3
e6-2
a6-2
e8
a8

Figure 12. The improved state transform diagram of factory division in PQMS2.

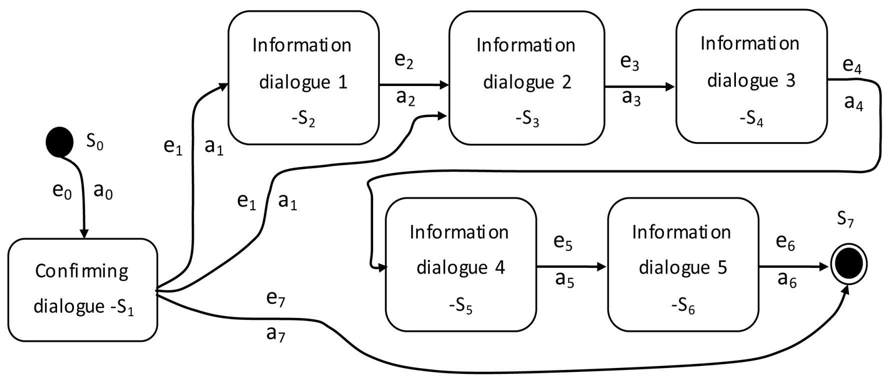

Information dialogue 1 -S2
e2
a2
Information dialogue 2 -S3
e3
a3
Information dialogue 3 -S4
e4
a4
S0
e1
a1
e0
a0
e1
a1
Information dialogue 4 -S5
e5
a5
Information dialogue 5 -S6
e6
a6
S7
Confirming dialogue -S1
e7
a7

Figure 13. The improved state transform diagram of deleting all data from category.

independent tester may act as a judge in whole testing activity. For state testing, regardless "Input control", the improved STD should still be adopted.

Figure 13 shows the improved STD of "deleting all data from category", which is a function for initialization of PQMS2. As such, test case design can be done in terms of result of Figure 13, and the state testing of this example include 9 test cases with code of PQMS2-DAS-UNI-TC010-TC018. For the execution of state testing of this kind unit, the probe should be applied to display the medium and final result of testing with visual mode.

6. Result and Conclusion

In China, the situation of software products with a small team is very popular at a great rate, and exploring an effective and cooperative test team for software production is an important work. And how to organize the testing activity and how to execute the unit testing task will be a very meaningful affair in the routine. In this paper, according to aspects of organizing architecture and organizing execution of unit testing activity embedding pair-wise mode for a small team, the detailed contents of organizing architecture and organizing execution are completely discussed. For this new organization, organizing architecture mainly includes personnel organizing and task division. And then organizing method of procedure and key points of talking unit testing are discussed in detail. Without loss of generality of our study, and a series of factual executable examples are investigated for GUI software including the update testing approach and traditional approach. The results of unit testing activity in operation practice indicate that organizing architecture embedding pair-wise mode for a small team is reasonable and organizing execution of unit testing activity fitting practice operation is effective.

Conflicts of Interest

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

References

[1] Boehm, B.W. (1979) Classics in Software Engineering. Yourdon Press, New Jersey.

[2] Runeson, P. (2009) Guidelines for Conducting and Reporting Case Study Research in Software Engineering. *Empirical Software Engineering*, **14**, 131-164.

[3] Patton, R. (2006) Software Testing. 2th Edition, Pearson Education Inc., New York.

[4] 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

[5] Tang, D., TanLi, M. and Li, T. (2022) Software Test Organizing for Small Team Based on "Pair-Wise" Mode. *Proceedings of* 2022 *International Conference on Smart Transportation and Future Mobility*, Changsha, 2-4 September 2022, Unpublished.

[6] TanLi, M., Zhang, Y., Jiang, Y. and Wang, Y. (2021) Baseline Test Suite Construction of Smoke Test for Extreme Programming. *IOP Conference Series*: *Materials Science and Engineering*, **1179**, Article ID: 012001. https://doi.org/10.1088/1757-899X/1179/1/012001

[7] TanLi, M.Q., Jiang, Y. and Wang, Y.L. (2020) Infrastructure Building of Software Testing for Engineering Software Based on Cooperation of University and Company. *Proceedings of the* 10*th International Workshop on Computer Science and Engineering*, Shanghai, 19-21 June 2020, 18-26.

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

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

[10] Xu, Y.-Y. (2015) A Study of Test Case Reuse Based on CBR. *Computer Engineering and Software*, **36**, 117-120.

[11] TanLi, M., Zhang, Y., Wang, Y. and Jiang, Y. (2015) 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

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

[13] 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

[14] TanLi, M., Jiang, Y., Wang, Y., Wang, X. and Peng, R. (2018) Digital Inspection of Cutting and Machining Based on Manufacturing Quality for Shop Floor. *DEStech Transactions on Engineering and Technology Research*, 1-7. https://doi.org/10.12783/dtetr/icmeit2018/23372

[15] TanLi, M., Zhang, Y. and Wang, Y. (2020) System Testing Based on Software Performance. *Computer Engineering and Software*, **41**, 1-5, 41.

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