ISSN: 1813-4890

Optimization and Incremental Construction of Test Suite for Integration Testing

Mengqing Tanli[1], Ying Zhang[2], Yulin Wang[2], and Yan Jiang[1]

[1] School of Software, University of South China, Hunan Hengyang 421001, China

[2] School of Mechanical Engineering, University of South China, Hengyang 421001, China

Abstract

This paper mainly discussed two aspects of test suite construction in incremental integration testing. The first aspect is optimization of baseline test suite for integration testing, in which, how to decrease the executed time by using grey-box approach is proposed and how to reduce the number of test case with optimal route approach in test suite construction is consequently investigated. In order to accelerate the speed of integration testing, the test design is the key task. On one hand, taking into account for baseline test suite is a key point. On the other hand, considering the incremental construction is a more frequent point. For the latter, typical examples are applied that three examples from requirement variety of user is introduced and transforming processing based on fault-tree analysis is put forward for test case design.

Keywords

Test suite, integration testing, testing optimum, incremental construction.

1. Introduction and Background

Software integration testing, as we known, is the very important part in testing activity [1-3], and the software defects found in the integration testing is 40% of sum software faults discovered in all testing tasks [4]. Test-driven programming is excellent programming strategy for varied requirement, and it is very suitable for small team of software producing. From the view of regression testing, testing activity imply a lot of test cycles, and figure 1 has shown this cycle mode of software integration testing for test-driven programming [5-7].

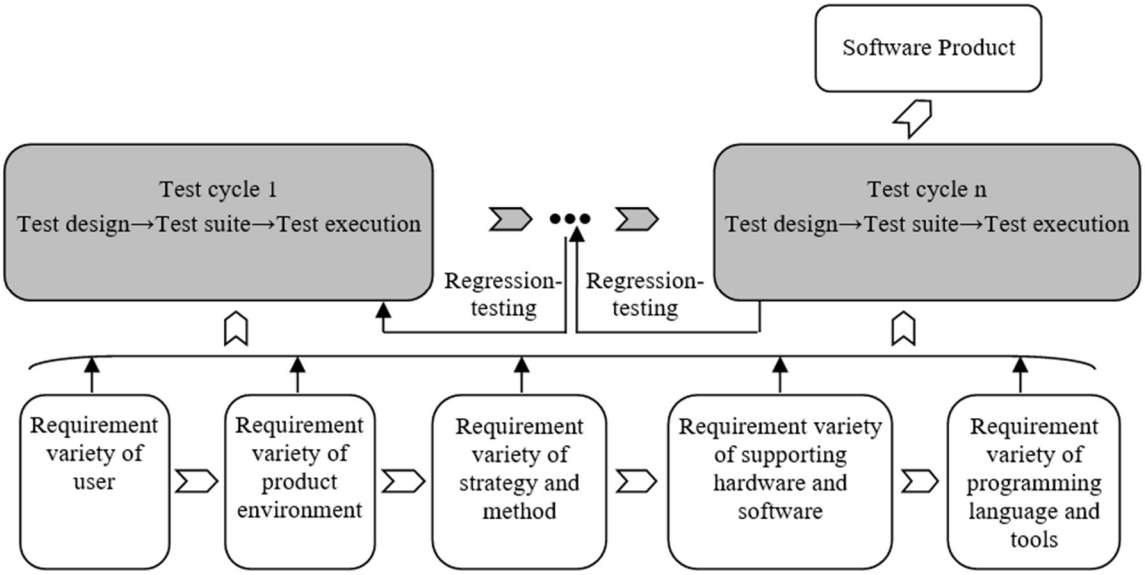

Figure 1. Software integration testing cycle for test-driven programming

ISSN: 1813-4890

Additionally, in figure 1, we introduce five types of requirement variety as following: (1) Requirement variety of user - for example, changing of function, adding of function, etc. (2) Requirement variety of product environment - for example, converting customization to domain user, etc. (3) Requirement variety of strategy and method - for example, single product is transformed into product-family, etc. (4) Requirement variety of supporting hardware and software - for instance, "Windows 98 → Windows XP", "PC computer → Mobile phone" and "32bit → 64bit", etc. (5) Requirement variety of programming language and tools - for instance, "Borland C++ → Visual C++".

In order to discuss effectively and systematically the test suite construction of incremental integration testing, the integration testing processing of PQMS2 (Product Quality Monitoring Software 2.0) is taken as a typical example. Following will investigate optimization of baseline test suite for integration testing firstly, and then incremental construction of integration regression test suite will be discussed in detail, in which three examples from requirement variety of user is introduced and transforming processing based on fault-tree analysis is proposed for test design.

2. Optimization of Baseline Test Suite for Integration Testing [8, 9]

Integration testing should synthetically consider speed, quality and cost [8], but the speed should be regarded as the key point especially for test-driven programming which is aimed at the fast responding speed and flexibility for varied requirement [9]. Thereupon, optimal point of integration testing for test-driven programming should be speed firstly.

No matter how, the baseline test suite should be perfectly constructed [10] at the beginning of integration testing, because it is the front obligatory condition for integration testing. And time axis of test design of baseline test suite for integration testing has been shown in figure 2.

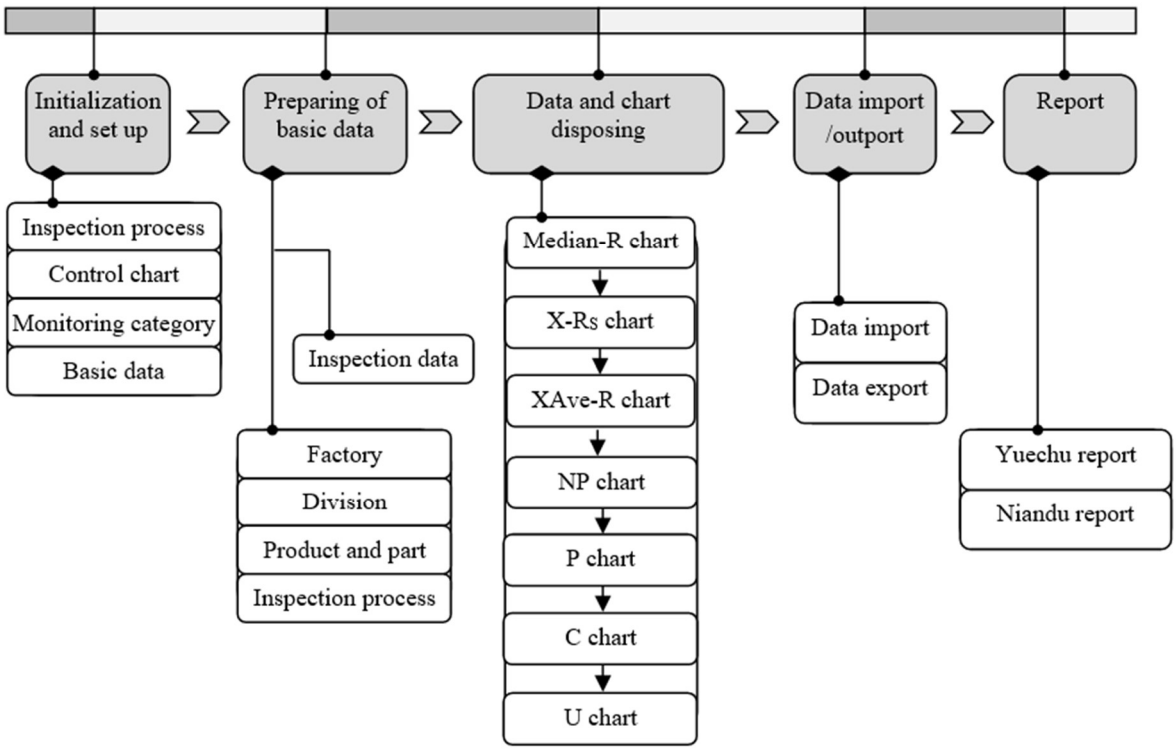

Figure 2. Time axis of test design of baseline test suite for integration testing

ISSN: 1813-4890

With reasonable arrangement and consequential testing design activity, the baseline test suite for PQMS2 is constructed, in terms of schedule in figure 2, and it is finished by means of grey-box approach [11]. In order to evaluate the achieved efficiency, we have done a brief statistics, and the table 1 has shown the statistical result of arrangement of test case in baseline test suite of PQMS2 for grey-box approach.

Table 1. Overall arrangement of test case in baseline test suite

	Initialization and setting	Basic data processing	Main function- inspection data and control chart processing	Data import/export	Report	Sum
White-box	3	10	2	2	2	19
Black-box	13	19	17	3	2	54
Rate	1:4.3	1:1.9	1:8.5	1:1.5	1:1	1:2.8

As the table 1 shows, the number of test case for black-box in "Basic data processing"(Column 2) is 19, and the number in "Main function- inspection data and control chart processing" is 17 , the two items are given prominence to others. We can find that the arrangement of test case is reasonable for factual scenario. On the other hand, the rate concerning white-box testing and black-box testing in "Main function- inspection data and control chart processing" achieves 1:8.5, it exhibits that the grey-box approach has got great efficiency to some extent.

2.1. Optimal Strategy of Baseline Test Suite on Execution Time for Grey-Box Approach [10, 11]

The grey-box strategy is applied in construction of baseline test suite for PQMS2, and the core essence of grey-box strategy is that white-box testing and black-box testing are reasonably synthesized with front white-box testing for message handling mechanism. For details of grey-box approach, please refer related contents of [11]. In order to focus on the key processing and decrease data volume referring to optimal strategy, we only discuss partial key set of baseline test suite.

In baseline test suite of integration testing for grey-box approach in PQMS2, we executed the testing of "adding inspection data and print quality control chart" which are the key processing in software function. By actual operation of testing, we have got the result of executed time as listed in table 2.

Table 2. The executed time of partial test suite in PQMS2 /min

	white-box	black-box	Sum	Note
	P_0-P_1	P_1-P_2-P_3-P_4		
Path r_I	0.17	7.15	7.32	Adding data manually, 20 data
Path r_{II}	0.17	10.33	10.50	Importing data from digital gauge, 12 data
Path r_{III}	0.17	7.91	8.08	Calling data from saving, 60 data
Path r_{IV}	0.17	8.09	8.26	Importing data from Notepad, 40data
Path r_V	0.17	4.82	4.99	Importing data from external, 12data
Sum	0.85	25.5	39.15	A_1=39.15/5=7.83

ISSN: 1813-4890

Note that, executed time of front events by white-box testing are: (1) Pop Menu Item - 0.17 min (listed only in table 2), (2) Menu Item - 0.03 min, (3) Toolbar - 0.03 min and (4) Hot Key - 0.13min. And meaning of processing node in table 2 are: (1) P0 is the start point, (2) P1 is the entrance point of map function or initial member function, and (3) P2, P3, P4 imply respectively the point of data saving, data display, and control chart display and print.

From table 2, we can know that the sum of testing executed time is 39.15 min and the average is 7.83 min for "adding inspection data and print quality control chart", which present the key function in baseline test suite of integration testing for applying grey-box technique.

However, testing executed time applying black-box testing only are following:

Testing path I—Pop Menu Item,

M1=39.15 min.

Testing path II—Menu Item,

M2=5×0.03+(17.58+1.01+14.89)=33.63 min.

Testing path III—Toolbar,

M3=33.63 min.

Testing path IV—Hot Key,

M4=5×0.13+(17.58+1.01+14.89)=34.13 min.

Sumarizing,

$$M= M1 +M2+ M3+ M4=140.54 \text{ min.} \quad (1)$$

And the average value is:

$$A2=140.54/4=35.135\text{min.} \quad (2)$$

So, the testing executed efficiency of the key function in baseline test suite for applying grey-box technique is:

$$C = A2/ A1 = 35.135/7.83 = 4.49 \quad (3)$$

As a result, we can know that the efficiency is accelerated 449% for integration testing of "adding inspection data and print quality control chart" in PQMS2.That is, one tester could do 4.49 times workload applying grey-box technique based on message than the method of testing all message paths.

2.2. Optimal Strategy of Baseline Test Suite on Number of Test Case

Correspondingly, in order to accelerate the speed of integration testing, decreasing the number of test case may be a direct way, including the processing in baseline test suite construction and the processing of incremental test design which will be discussed in section 3 [10]. Optimization of number of test case in baseline test suite construction should be transformed into a problem of optimal route with condition, as shown in figure 3.

In figure 3, solid circle is the test case set and broken circle implies the unused repeated test case set without applying grey-box approach. Consequently, "-·-" group is function involved, and "····"group presents white-box testing involved. At the same time, the broken arrow line implies virtual path without applying grey-box approach corresponding to the broken circle, for which "S4→MP→I5" is only given and others are abbreviated. Finally, the meanings of codes

ISSN: 1813-4890

are shown in table 3-5, which table 3 demonstrates the code meaning of white-box testing and table 4 depicts the code meaning of black-box testing for key processing while table 5 is the code of black-box testing for preparation of basic data in figure3. Please note that DC is not directly influence the monitoring category and considered in unit testing.

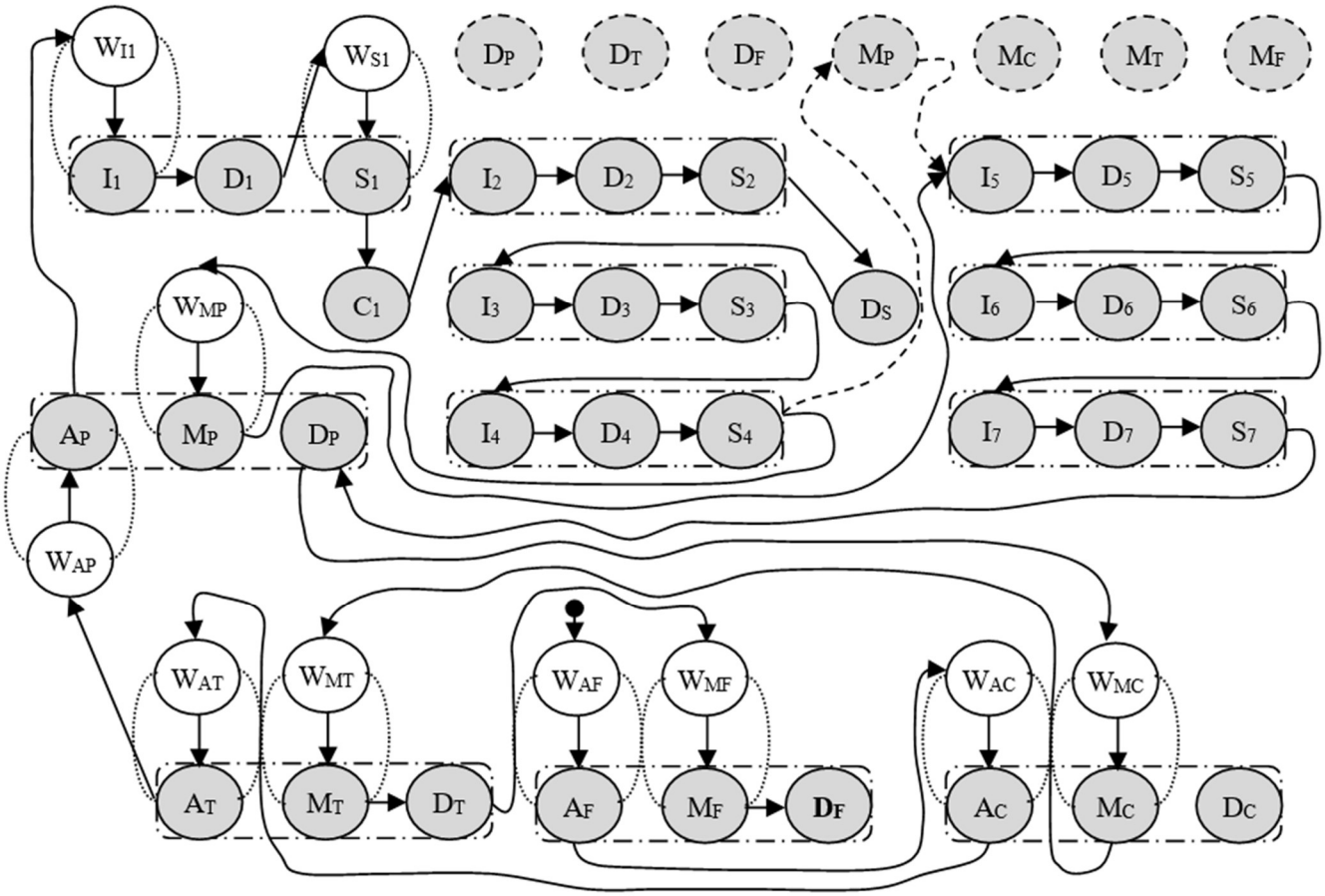

Figure 3. Optimal route of integration testing based on empirical history

Table 3. The code of white-box testing in figure3

Code	Meaning	Code	Meaning
W_{I1}	White-box testing of adding inspection data	W_{S1}	White-box testing of basic setting
W_{MP}	White-box testing of modifying inspection process	W_{AP}	White-box testing of adding inspection process
W_{AT}	White-box testing of adding part	W_{MT}	White-box testing of modifying part
W_{AC}	White-box testing of adding product	W_{MC}	White-box testing of modifying product
W_{AF}	White-box testing of adding division	W_{MF}	White-box testing of modifying division

ISSN: 1813-4890

Table 4. The code of black-box testing for key processing in figure3

Code	Meaning	Code	Meaning
I1	Input manually inspection data-Median chart	D1	Adding data→Display and print Median-R chart
S1	Setting→display and print Median-R chart	C1	Modifying inspection data ID
I2	Import inspection data by digital gauge	D2	Adding data→Display and print X-Rs chart
S2	Setting→display and print X-Rs chart	DS	Deleting inspection data
I3	Input manually saving data and modification	D3	Adding data→Display and print XAve-R chart
S3	Setting→display and print XAve-R chart	I4	Input inspection data saving using Notepad-NP chart
D4	Adding data→Display and print NP chart	S4	Setting→display and print NP chart
I5	Input manually inspection data-modifying-deleting	D5	Adding data→Display and print P chart
S5	Setting→display and print P chart	I6	Input manually saving inspection data
D6	Adding data→Display and print C chart	S6	Setting→display and print C chart
I7	Input manually saving inspection data and deleting	D7	Adding data→Display and print U chart
S7	Setting→display and print U chart		

As demonstrated in figure 3, it is easy to find that total number of test case set is 45 in this partial test suite for the key processing in PQMS2, and the number of saving is 7 (the top-right of figure 3 with broken circle). But this result is got when preparation of inspection data is only taken into account. Of course, the percent of saving is easily given by:

$$c = \frac{7}{45+7} \times 100\% = 13.5\%$$

Table 5. The code of black-box testing for preparation of basic data in figure3

Code	Meaning	Code	Meaning
AP	Adding inspection process	MP	Modifying inspection process
DP	Deleting inspection process	AT	Adding part
MT	Modifying part	DT	Deleting part
AF	Adding division	MF	Modifying division
DF	Deleting division	AC	Adding product
MC	Modifying product	DC	Deleting product

Otherwise, if considering the preparation of basic data – "AP, AT, AC, AF"(also with broken circle), the number of saving is 11, and the percent of saving is:

$$c = \frac{11}{45+11} \times 100\% = 19.6\%$$

Table 6. Efficiency of saving number of test case set in baseline test suite

	White-box	Black-box	Sum	Saving	Percent	Memo
1	10	35	52	7	13.5%	Only preparation of inspection data is considered
2	10	35	56	11	19.6%	Considering the preparation of basic data

We can conclude that it may cut down 19.6% test case in optimal test design for baseline test suite construction. Summarizing, efficiency of saving number of test case set in baseline test suite of integration testing is listed in table 6.

ISSN: 1813-4890

3. Incremental Construction of Regression Test Suite based on Fault-Tree Analysis

Before incremental testing, we should confirm that construction of baseline test suite is finished. And incremental construction of regression test suite must maximize the reuse of baseline results to decrease the number of test case and improve testing efficiency [10].

As aforementioned in figure 1, the advantage of test-driven programming is its good responding characteristics for user requirement. How to transform the requirement variety into changing of test design would be a very important issue in test-driven programming and testing activity.

At first, we can conclude that there are three kinds of derivation of requirement variety from user, that is, investigation and interview of users and customers, backward information of user on internet, and interaction information of user on the spot. The following discussion will be introduced according to the three kinds of gathering routes.

Without loss of generality, when user has new requirements as following:

(1) Requirement I - Coordination of R chart should be independent in XAve-R chart - derived from investigation and interview of customers.

(2) Requirement II - Batch restoring of inspection data - derived from backward information of user on Internet.

(3) Requirement III - Consistence testing is necessary for adding division and department - derived from interaction information of user on the spot.

In terms of these requirement varieties, programmer would modify the codes and submitted to regression testing. As a consequence, incremental construction of regression test suite occurred for testing engineer.

We should notice that the procedure of incremental construction of regression test suite is a cooperation process between testing engineer and programmer, and it is also an interaction testing activity to find software faults and failures by familiarizing testing object.

Requirement I.

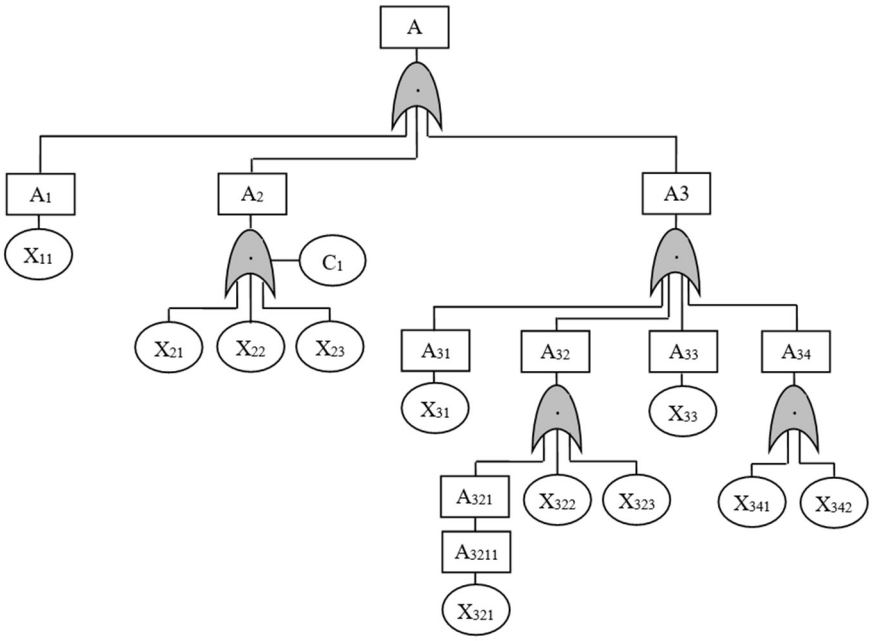

Figure 4. Fault-tree analysis for requirement I

ISSN: 1813-4890

In investigation and interview of users and customers, "coordination of R chart should be independent in XAve-R chart" is gathered as an important requirement. Thereupon, Adding coordination-offset coefficient for R chart is necessary respectively in XAve-R chart.

Before the construction of regression test suite, the dependency analysis must be done to reveal the mutual relationship and influence conducted by programming modification. Fault-tree analysis is very important tool for testing engineer, and figure 4 has shown the fault-tree analysis for this user requirement [12, 13].

In figure 4, the symbol A presents the top event of fault-tree, and Ai,j,k…presents the middle event of fault-tree, while Ci is the additional condition. Correspondingly, details are depicted in table 7.

Table 7. The top and middle event of fault-tree analysis for requirement I

Code	Event statement	Code	Event statement
A	Adding coordination-offset coefficient for R chart is necessary respectively in XAve-R chart	A_{33}	Disposing for GUI influence in OnDraw()
A_1	Adding unit in GUI and variables in .cpp	A_{34}	Adding data interface parameters
A_2	Adding member in View class, Dialogue class and their objects	A_{321}	Disposing in global initialization
A_3	Adding member in data interface	A_{3211}	Disposing for influence - OnProcessUpdate()
A_{31}	Disposing for data saving interface	C_1	X_{21} is done before X_{22}
A_{32}	Disposing for data gathering interface	C_2	A_{34} is done before A_{33} , A_{32} and A_{31}

Consequently, $X_{i,j,k}$…presents the final event of fault-tree, and table 8 has demonstrated the detail meaning of the code.

Table 8. The final event of fault-tree analysis for requirement I

Code	Event statement	Code	Event statement
X_{11}	Adding static text, edit box controls and their variables	X_{322}	Disposing of data getting in OnDraw() – GetSetting()
X_{21}	Declaration of "int RChart_offset"	X_{323}	Disposing of data getting in OnUpdate()– etSetting()
X_{22}	Disposing in dialogue initialing – InitDialog()	X_{33}	Disposing for GUI influence in OnDraw()- RChart_offset
X_{23}	Disposing in View class – OnSetfigure()	X_{341}	Adding definition of unit in strSetting[]
X_{31}	Disposing in data saving – OnSaveSetting()	X_{342}	Adding disposing of chart initialization for definition of unit in strSetting[]
X_{321}	Disposing for influence in calling – OnGetCurrentProcess()		

We can do test design according to the results of fault-tree analysis, that is to say, according to the final event of fault-tree, choosing reasonable test case form baseline test suite or designing new test case and adding them into test suite. As a result, table 9 is the disposing result in detail.

ISSN: 1813-4890

Table 9. Choice [a] and adding [b] of test case for requirement I

ID of test case	Testing content	Referring
PQMS2-ENT-INT-TC304-MF	Add division and department from sheet	X_{341}
PQMS2-ENT-INT-TC307-MF	Add division and department to monitoring category	X_{341}
PQMS2-ENT-INT-TC314-MF	Add product from sheet	X_{341}
PQMS2-ENT-INT-TC324-MF	Add part from sheet	X_{341}
PQMS2-ENT-INT-TC327-MF	Add product_ part to monitoring category	X_{341}
PQMS2-ENT-INT-TC334-MF	Add inspection process form toolbar	X_{341}
PQMS2-ENT-INT-TC337-MF	Add inspection process to monitoring category	X_{341}
PQMS2-ENT-INT-TC360-MF	Input test data from saving manually	X_{341}
PQMS2-ENT-INT-TC906-MF	Display and print XAve -R chart from monitoring category	$X_{11}, X_{21}, X_{322}, X_{341}, X_{342}$。
[a]Unit testing is not considered. [b]Adding test case in unit testing - PQMS2-ACF-UNI-TC001～TC025-MF, etc.		

Requirement II.

As a requirement variety, “Batch restoring of inspection data” is derived from backward information of user on internet [14, 15].

This example is a typical requirement variety with new unit and module, and the fault-tree analysis and test design should be conducted considering the follows.

(1) For test-driven programming, in incremental construction of test suite, integration testing should be taken into account firstly.

(2) If organizing with pair-wise programmer and test engineer [16], there are two types of concrete task arrangement. The first one is idle mode, in which tester can do some preparation of testing when idle gap arrived. Another one is parallel mode, in which the pair-wise can do programming and testing in parallel.

(3) For test design of integration testing, test engineer should keep cooperation with programmer.

(4) For test design of unit testing, test engineer may only design test case of important unit according to confirmation of test manager.

Table 10. The event of fault-tree analysis for requirement II

Code	Event statement	Code	Event statement
A_1	Add controls from main/pop-up menu and event map	A_2	Add member function statement and body
A_{21}	Add member function statement	A_{22}	Add member function body
X_{11}	Add controls from main menu/pop-up	X_{12}	Add event map
X_{21}	Add member function statement in MainView.h	X_{221}	Attributes statement and initialization
X_{222}	Judge if inspection process is exist, and get basic information e.g. inspection process, part, etc.	X_{223}	Searching all target file name
X_{224}	Judge if this inspection data is exist	X_{225}	Add inspection data ID into INSPECTIONDATADIRECTORYNAME
X_{226}	Add filename and all data of inspection data file into SAVINGDIRECTORY	C_1	Sequence should be "$X_{221}\rightarrow X_{222}\rightarrow X_{223}\rightarrow X_{224}\rightarrow X_{225}\rightarrow X_{226}$"

ISSN: 1813-4890

Similarly, after the task arrangement of manager, programmer modified the code and submitted to regression testing. As a consequence, testing engineer analyzed the original code with fault-tree tool, and figure 5 has shown the fault-tree analysis for requirement II.

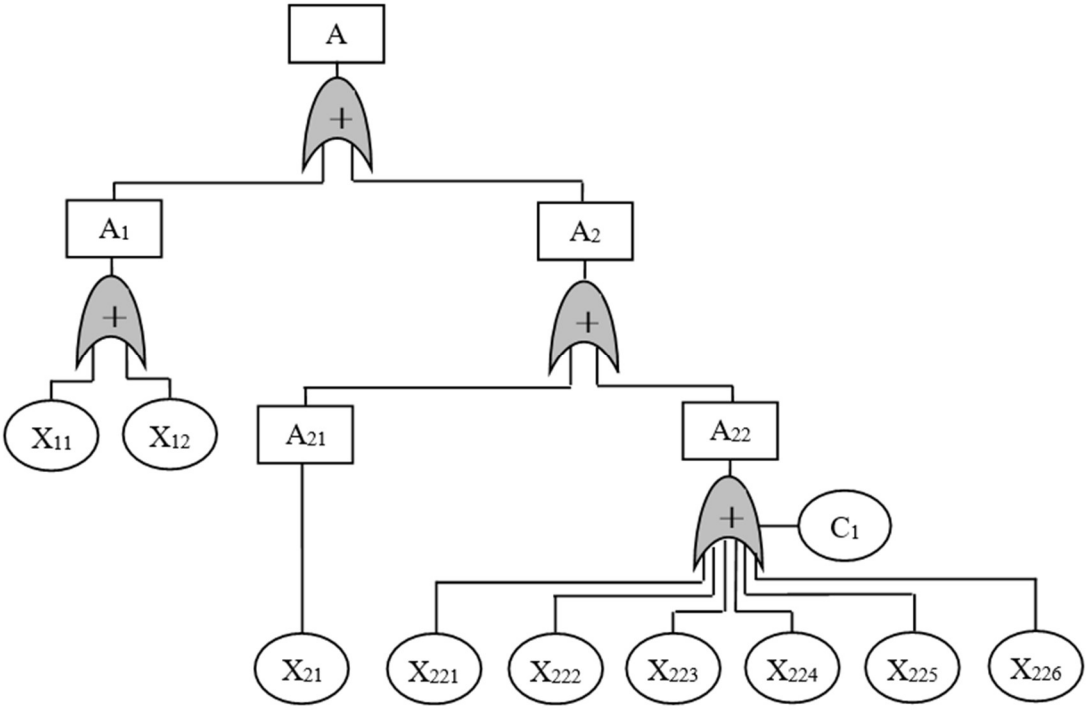

Figure 5. Fault-tree analysis for requirement II

In figure 5, "Batch restoring of inspection data is necessary" is the top event, and it noted with "A". Consequently, $A_{i,j,...}$presents the middle event of fault-tree, and $X_{i,j,k}$...is the final event of fault-tree, while C_i is the additional condition. Correspondingly, details are demonstrated in table 10.

According to the results of fault-tree analysis in figure 5, the final events of fault-tree in table 10 have implied the concrete subject of test design. Thus, the disposing result has been demonstrated in table 11.

Table 11. Choice and adding of test case for requirement II

ID of test case	Testing content	Referring
PQMS2-ENT-INT-TC372-AD	White-box testing for POP-Menu	X_{11}, X_{12}, X_{21}
PQMS2-ENT-INT-TC373-AD	White-box testing for Menu-item	X_{11}, X_{12}, X_{21}
PQMS2-ENT-INT-TC374-AD	White-box testing for Hot-key	X_{11}, X_{12}, X_{21}
PQMS2-ENT-INT-TC145-AD	Integration testing of batch restoring inspection data	X_{11}, X_{12}, X_{21}, X_{221}, X_{222}, X_{223}, X_{224}
PQMS2-FRD-UNI-TC001	Testing whether output of starting from monitoring category is correct or incorrect	X_{11}, X_{12}, X_{21}, X_{221}, X_{222}
PQMS2-FRD-UNI-TC002	Testing the output of starting from Menu-item for correct inspection process	X_{11}, X_{12}, X_{21}, X_{221}, X_{222}
PQMS2-FRD-UNI-TC003	Testing the output of starting from Menu-item for incorrect inspection process	X_{11}, X_{12}, X_{21}, X_{221}, X_{222}

ISSN: 1813-4890

Requirement III.

As the interaction information of user on the spot, "Consistence testing is necessary for adding division and department" is proposed as an important requirement. Similarly, programmer modified the code and submitted to the tester for regression testing, and testing engineer analyzed the original code, and figure 6 has shown the analysis result applying fault-tree [12, 13].

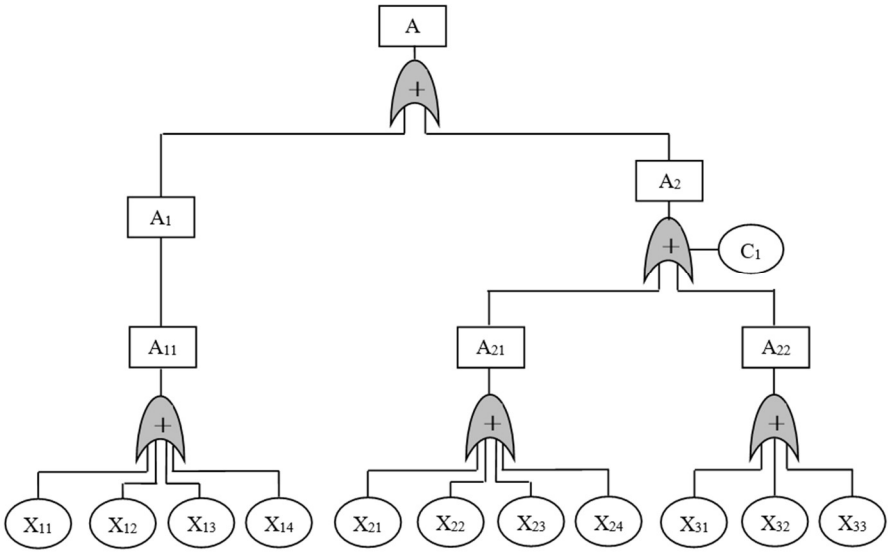

Figure 6. Fault-tree analysis for requirement III

For fault-tree analysis in figure 6, correspondingly, the symbol A presents the top event of fault-tree "Consistence testing is necessary for adding division and department", and $A_{i,j...}$presents the middle event, while C_i is the condition. And details meaning of codes are shown in table 12.

Table 12. The top and middle event of fault-tree analysis for requirement III

Code	Event statement	Code	Event statement
A_1	Using the mode of sheet adding	A_2	Using the mode of category adding
A_{11}	Influence in adding from sheet – CDivisionDIALOG::OnButtonAdd(),CDivisionDIALOG:: OnButtonAddDivisionToTree()	A_{21}	Influence in adding from category – CMainView::OnAddDivision()
A_{22}	Influence in adding from category – CDivisionDIALOG::OnButtonAddDivisionToTree()	C_1	Testing A_1 before testing A_2

Table 13. The final event of fault-tree analysis for requirement III

Code	Event statement	Code	Event statement
X_{11}	Testing if division code is empty.	X_{23}	Testing if it is item of product-part.
X_{12}	Testing if division name is empty.	X_{24}	Testing if it is item of division or department.
X_{13}	Testing if factory name is empty.	X_{31}	Testing if this item already existed.
X_{14}	Testing if adding is finished.	X_{32}	Testing if item number is bigger than 10.
X_{21}	Testing if it is item of inspection data.	X_{33}	Testing if item adding can be finished.
X_{22}	Testing if it is item of inspection process.		

ISSN: 1813-4890

In terms of the results of fault-tree analysis in figure 6, the final events of fault-tree in table 13 have implied the detail subject of test design. In general, test design may include two aspects - choosing reasonable test case form baseline test suite or designing new test case and adding them into test suite [16-19]. However, the node layer and position of fault-tree must be taken into account for test design, and distributing between unit testing and integration testing should be arranged reasonably [19]. Consequently, table 14 has demonstrated the disposing result in detail.

Table 14. Choice and adding of test case for requirement III

ID of test case	Testing content	Referring
PQMS2-ENT-INT-TC305-MF	Integration testing-Testing if division code is empty.	X_{11}
PQMS2-ENT-INT-TC306-MF	Integration testing-Testing if division name is empty.	X_{12}
PQMS2-FBA-UNI-TC001	Unit testing-void CDivisionDIALOG::OnButtonAdd()	X_{13}
PQMS2-ENT-INT-TC304-MF	Integration testing-Testing if adding is finished.	X_{14}
PQMS2-FAD-UNI-TC001	Unit testing-void CMainView::OnAddDivision()	X_{21}
PQMS2-FAD-UNI-TC002	Unit testing-void CMainView::OnAddDivision()	X_{22}
PQMS2-FAD-UNI-TC003	Unit testing-void CMainView::OnAddDivision()	X_{23}
PQMS2-FAD-UNI-TC004	Unit testing-void CMainView::OnAddDivision()	X_{24}
PQMS2-FBA-UNI-TC002	Unit testing-void CDivisionDIALOG::OnButtonAddDivisionToTree()	X_{31}
PQMS2-FBA-UNI-TC003	Unit testing-void CDivisionDIALOG::OnButtonAddDivisionToTree()	X_{32}
PQMS2-ENT-INT-TC307-MF	Integration testing-Testing if item adding can be finished.	X_{33}

4. Result and conclusion

Construction of integration test suite must hold the priority criterion of testing efficiency mainly focusing on testing speed, and there are two methods to do. The first one is decreasing the execution time of test case, for which, we have applied the grey-box approach. The second one is reducing the number of test case, for which, we have a improved resolution by optimal route of integration testing. For incremental integration testing, test design includes two main aspects, that is, choosing reasonable test case from test suite and conducting new test case according to fault-tree analysis. The analysis tool of fault-tree may be applied for dependency analyzing for test design, not only used in controls adding for visual GUI software, but also used in software unit adding and member function adding. Correspondingly, some experiences and skills should be utilized in incremental construction of test design: (1) Integration testing should be taken into account firstly for test-driven programming. (2) Two types of task arrangement, the idle mode and the parallel mode, can be applied if organizing with pair-wise programmer and test engineer. (3) Test engineer should keep cooperation with programmers for test design of integration testing. (4) Test engineer may only design test case of important unit according to confirmation of test manager in test design of unit testing.

References

[1] J. S. Bradbury, J. R. Cordy, J. Dingel. An Empirical Framework for Comparing Effectiveness of Testing and Property-Based Formal Analysis, Proceedings of the ACM SIGPLAN-SIGSOFT Work on Program Analysis for Software Tools and Engineering. (Lisbon, Portugal, 2005). p. 160-170.

[2] P. Runeson. Guidelines for conducting and reporting case study research in software engineering, Empirical Software Engineering, Vol. 14 (2009), No. 2, p. 131–164.

[3] B. W. Boehm. Classics in software engineering (New Jersey: Yourdon Press, USA 1979).

[4] Alessandro Orso, Integration Testing of Object-Oriented Software, POLITECNICO DI MILANO.2001.

[5] K. Beck, Test-Driven Development by Example (China Electric Press, China 2003).

[6] H. Do, G. Rothermel, S. Elbaum, Infrastructure Support for Controlled Experimentation with Software Testing. Proceedings of the 2004 International Symposium on Empirical Software Engineering. (USA, April 13, 2005). p. 60-70.

[7] MENGQING TANLI, YAN JIANG, YULIN WANG, et al., Infrastructure Building of Software Testing for Engineering Software Based on Cooperation of University and Company. Proceedings of 2020 the 10th International Workshop on Computer Science and Engineering-WCSE2020. (Shanghai, China, June 19-21, 2020). p. 18-26.

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

[9] Mengqing TANLI, Ying ZHANG, Yan Jiang, et.al., Baseline Test Suite Construction of Smoke Test for Extreme Programming. Proceedings of 2021 International Conference on Communication Engineering and Logistics Management. (Shanghai, China, July 24-26, 2021).

[10] XU Yuan-yuan. A Study of Test Case Reuse Based on CBR, Computer Engineering & Software, Vol. 36 (2015), No. 9, p. 117-120. (In Chinese).

[11] Mengqing TANLI, Ying Zhang, Yulin Wang, et.al., Grey-box Technique of Software Integration Testing Based on Message. Proceedings of 2021 3rd International Conference on Artificial Intelligence and Computer Science. (Beijing, China, July 29-31, 2021). p. 198-206.

[12] R. Patton, Software engineering (Pearson Education Inc., USA, 2006).

[13] TANLI Meng-qing, ZHANG Ying, WANG Yu-lin. Reasearch on Fault Tree Technique in Software Regression Testing, Computer Engineering & Software, Vol. 41 (2020), No. 9, p. 5-8, 25. (In Chinese)

[14] Dan Tang, Mengqing TanLi, Yan Jiang. Product Quality Monitoring of Shewhart Chart Based on Function integration for Manufacturing Factory. Proceedings of the 4th Annual International Conference on Information System and Artificial Intelligence, (Changsha, Hunan, China, May 17-18, 2019) p. 123-130.

[15] TANLI Meng-qing, ZHANG Ying, WANG Yu-lin. System Testing Based on Software Performance. Computer Engineering & Software, Vol. 41(2020), No. 11, p. 1-4, 25. (In Chinese).

[16] F. Li, Software Testing Technology (Mechanical Industry Press, China 2016).(In Chinese).

[17] G. J. Myers. The Art of Software Testing (New Jersey, USA 2004).

[18] Elfriede, Dustin, Effective Software Testing: 50 Specific Ways to Improve Your Testing (China Electric Press, China 2004).

[19] Chen Zhanhua, Research and Implementation of Test Method in Task Arrangement of Resource Satellite, Radio Engineering, Vol. 35 (2005), No. 2, p. 62-64. (In Chinese).