Advanced Test Case Design Methods – Going Far Beyond Boundary and Equivalence Testing

Subei Liu

liu.subei@xbosoft.com

Abstract

Complete and effective test case design will make the test more effective. There are various types of designing techniques, each of which is suitable for identifying a particular type of errors. Additionally, we need to select the right set of relevant test design techniques for the particular application and particular types of functions within an application. In this paper, I'll discuss some advanced design techniques for test cases. This will enable you to develop deeper test cases, get better test coverage and expand your toolbox for test case design and thinking. Along with examples, you can get better understanding of not only on the techniques themselves, but also when to use each one depending on the situation.

Keywords: Black-box test design techniques, Equivalence partitioning, Boundary value analysis, Cause–effect graph, Decision table testing, Functional structure diagram, Orthogonal array testing, Error guessing, Scene graph

Biography

Subei Liu, graduated from agricultural university of Hebei, China with a major in computer science and technology. She has more than 6 years' experience on software testing, testing all kinds of platforms and also trains for new recruits for XBOSoft. She has a deep understanding of software testing, and specializes in test design.

1 Introduction

In our modern society, change is no longer constant. Rather, it's accelerating. And this acceleration is caused by technology. Software has become the nervous system of technology as these technology-based systems are increasingly dependent on software. The software quality of these systems is particularly important because software failures can lead to very serious consequences. Software today is written by humans (notice we said today) and humans make mistakes. So, how do we ensure that the final software delivered to users meets their needs with high quality? Software testing.

However, with agile development methods, software testing often falls to the wayside. Back in the days of waterfall, discussions on software testing naturally included test cases. However, test cases have gotten lost with the agile tendency towards minimal documentation. New software testers may think that they can test software as soon as they get it. They are eager to find all the flaws in the software immediately, just as developers may rush to write code without thinking of design and architecture. However, software testing requires an engineering perspective. Before specific test implementation, we need to understand what to test and how to test. Even if we have no formal test cases per say, we need to formulate our test design to guide the implementation of testing. Otherwise, not only will your testing be ad-hoc, but you'll also miss testing critical areas where defects can often hide. This paper sets out to describe and show examples of various test design methods that we've successfully used in many projects.

2 What are test cases?

A test case is an output of test design, which directly reflects the idea of test design. A software program should function and perform as designed. If the program does not work properly or behave as designed, it means that the software programmer has created a defect in the software. Of course the design could be defective as well, but that is outside the scope of this paper. A beautiful test case is not only an excellent embodiment of the design idea, but also easy to execute and understand its flow, with readability and traceability is generally a set of test inputs, execution conditions and expected results, prepared for a particular goal to verify whether the program meets a specific requirement and does not complete redundant operations, namely, to ensure the following two points:

a. The program did what it was supposed to do

b. The program isn't doing anything it shouldn't be doing

Therefore, as the basis for test implementation, test cases should be carefully designed to cover the design of the program under test.

3 Black-box test case design techniques

The design of test cases is an important link in the process of software testing, and it is the only way to achieve the test goal. The success of a software test depends on the success of its test case design.

The common test case design methods of black box test include: equivalence class partition, boundary value analysis, error guessing, cause-effect graph, decision table, Orthogonal array, functional graph method, etc

3.1 Equivalence partitioning

This is a very common and important test case design method. We divide the input domain into several regions, and choose a few representative data for each region to test, so that we can avoid using a lot of

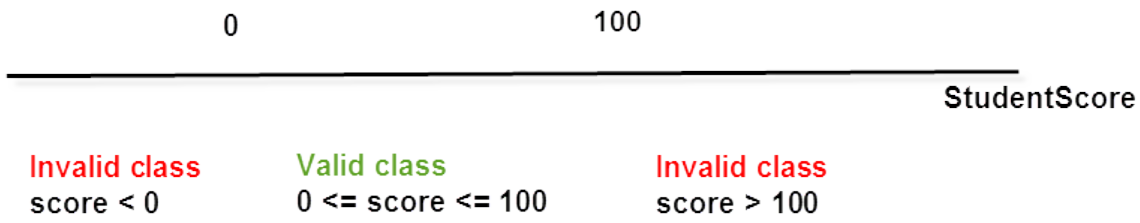
testing data and avoid blindness. Thereby reducing the total number of test cases that must be developed. When use this method, we find the input conditions first, and divide the equivalence, then design test cases.

There are two different cases of equivalence partitioning, valid class and invalid class. A valid equivalence class is a collection of input data that is reasonable and meaningful for a program's specification. A valid equivalence class verifies that a program has achieved the functionality and performance specified in the specification. While an invalid equivalence class is a collection of input data that is unreasonable or meaningless to the program specification.

In general, when designing test cases, one test case should cover as many valid classes as possible, while invalid equivalence classes require one-to-one coverage. For instance, if an input for email address can be saved with valid values letters and numbers together as expected in one test case, for sure it works with only letters or numbers in two different test cases. While for invalid values, email address cannot be saved with invalid values Chinese characters and asterisk together, you have to test for each one separately to see whether each invalid value is not supported as expected.

3.1.1 Example

The input value is student achievement, ranging from 0 to 100.



3.1.2 Strengths and Weaknesses

Strengths

Test with less representative data instead of full test with all data.

Weaknesses

1. Not consider detailed relationship between inputs and outputs, this may cause some logic errors.

2. Needs to understand the detailed requirements, and it is easy to omit equivalence classes when the requirements are not clear.

3.2 Boundary-value analysis

Boundary-value is another classical black box test method and often used together with equivalence partitioning as a supplement. In programming, a number of errors tend to occur at the boundaries of input or output data, so the method of using boundary values can often detect errors. Boundary value analysis considers not only the input conditions but also the test conditions generated by the output space.

Values of the boundary can be taken as min, min+, Max -, Max and norm, rather than typical or arbitrary values in equivalence classes. When define the boundary values, data domain in requirements is very important for choosing which one as boundary. For instance, if the level of precision is two decimal places for numbers/amounts, the one less than 1 is 0.99, while if the level of precision is three decimal places, the one less than 1 will be 0.999.

Common boundary values:

1) For integers of 16 bits, 32767 and 32768 are boundaries.

2) The first and last lines of the report.

3) The first and last element of an array.

4) The 0th, 1st and penultimate times of the cycle.

5) The cursor is at the top left and bottom right of the screen.

3.2.1 Example1

We can use the example in 3.1.1 to analysis boundary values for inputs:

Boundary value of valid equivalence class	Boundary value of invalid equivalence class
0	-1
100	101

3.2.2 Example2

Here is another example to analysis boundary values for outputs:

Rules for park admission tickets:

1). Children under 1.2m in height are free.

2). Children between 1.2m (including 1.2m) and 1.4m in height can get half price.

3). People over 70 years old (including 70 years old) can get free tickets.

4). Half fare for students in school (excluding on-the-job students).

5). No charge for families of revolutionary martyrs and soldiers on active duty.

We can divide the equivalence into full, half and free tickets, see table:

Output	No.	Input	Boundary value of valid equivalence class	Boundary value of invalid equivalence class
Free	1	Children under 1.2m in height	1.19m	1.2m
	2	People over the age of 70	70 years old	69 years old
	3	Families of revolutionary martyrs	Families of revolutionary martyrs	
	4	Soldiers on active duty	Soldiers on active duty	
Half	5	Children between 1.2m (including 1.2m) and 1.4m in height	1.2m, 1.21m, 1.39m	1.4m
	6	Students in school (excluding on-the-job students).	Students in school	on-the-job students
Full	7	Children over 1.4m in height	1.4m	1.39m
	8	People under the age of 70	69years old	70years old
	9	On-the-job students	On-the-job students	

3.2.3 Strengths and Weaknesses

Strengths

1. Test cases are easy to automate.

2. Small design workload

Weaknesses

1. Based on the domain of data, the logical relationship of data is not recognized.

2. A large number of test cases can be generated which will need long execution time of test cases.

3.3 Error guessing

Error guessing is a method of designing test cases based on experience and intuition that surmises all possible errors in the program.

When use this, we list all possible errors and error prone special scenarios in the program, and select test cases based on them.

3.3.1 Example

For input:

1). Single Spaces, multiple Spaces

2). Special characters (<script>)

3). Date format in different browser languages.

4). Time limitation for submit

5). Two people modify the same form.

6). etc.

There are no definite steps and they are carried out largely by rule of thumb

3.3.2 Strengths and Weaknesses

Strengths

1. Give full play to people's intuition and experience.

2. Brainstorm

3. Easy to use

4. Fast and easy to carry out

Weaknesses

1. It is difficult to know the coverage of the test

2. A lot of unknown areas may be lost

3. It is subjective and hard to replicate

3.4 Cause–effect graph

The cause-effect graph is a method of designing test cases by analyzing various combinations of input conditions graphically. It is suitable for checking various combinations of input conditions of programs.

Both the equivalence method and the boundary value analysis method focus on the input conditions, but do not consider the various combinations of the input conditions and the mutual constraints among the input conditions.

If the various combinations of input conditions must be considered during testing, the number of possible combinations would be astronomical, so the design of test cases must be considered in a form suitable for describing combinations of multiple conditions and corresponding multiple actions, which requires the use of causal diagrams (logical models).

3.4.1 Example

Product description: there's a vending machine software that handles boxed drinks for $1.50 each. If you put in $1.50 coins, press the "coke" or "Sprite" button, and the corresponding drink will be delivered. If you put in $2 coins, return $0.5 coin with the beverage.

Test cases design steps

1). Determine the cause and effect in the requirements.

#	Cause / Input	#	Effect / Output
C1	Put in $1.50 coins	E1	Refund $0.50 coin
C2	Put in $2 coins	E2	Dispense Coke
C3	Press the "Coke" button	E3	Dispense Sprite
C4	Press the "Sprite" button		

2). Determine the constraints between inputs.

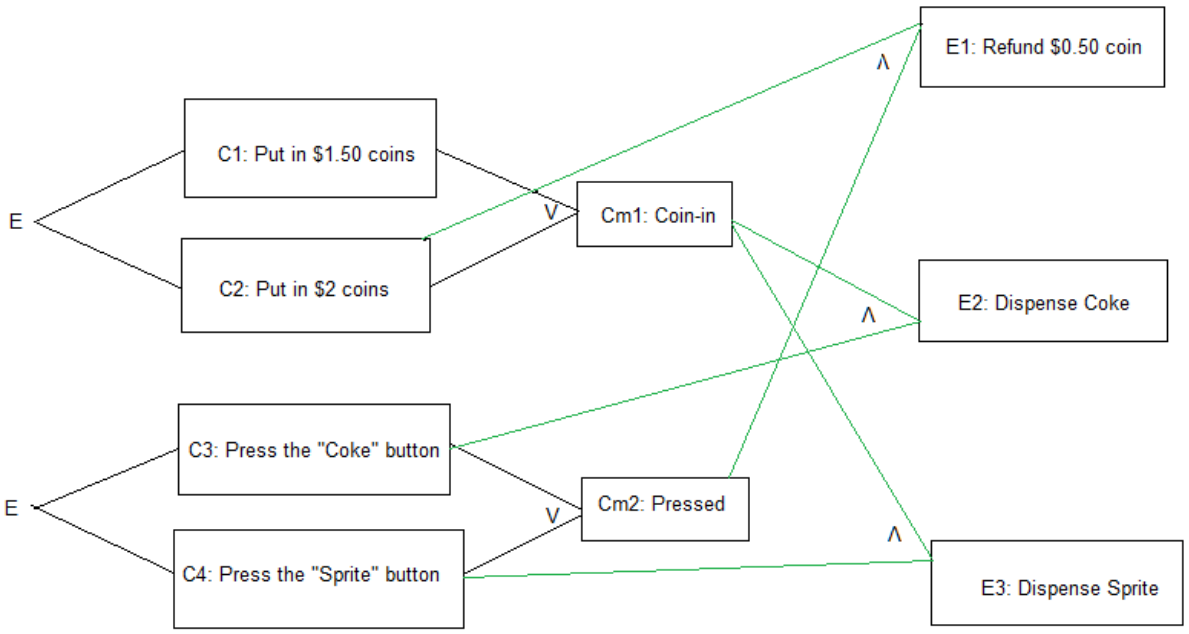
#	Cause / Input	Constraints
C1	Put in $1.50 coins	Exclusive: Either C1 or C2
C2	Put in $2 coins	
C3	Press the "Coke" button	Exclusive: Either C3 or C4
C4	Press the "Sprite" button	

3). Determine the causality between input and output.

#	Cause / Input	#	Middle status	#	Effect / Output
C1	Put in $1.50 coins	Cm1	Coin-in	E1	Refund $0.50 coin
C2	Put in $2 coins			E2	Dispense Coke
C3	Press the "Coke" button	Cm2	Pressed	E3	Dispense Sprite
C4	Press the "Sprite" button				

4). Draw the cause-effect graph.

To distinguish, black lines are used for inputs and green lines for outputs.



5). Convert graph to a decision table. (see 3.5)

6). Design test cases from decision table. (see 3.5)

3.4.2 Strengths and Weaknesses

Strengths

1. The causal-effect graph method can help us select test cases efficiently and design multiple input condition combination test cases according to certain steps.

2. Causal-effect diagram analysis can also point us to problems in software specification descriptions.

Weakness

1. The causal relationship between input conditions and output results is sometimes difficult to get from the software requirements specification.

2. Even if these causal relationships are obtained, the complex causal relationships lead to a very large causal diagram and a huge number of test cases.

3.5 Decision table testing

Decision tables are tools for analyzing and expressing situations where different operations are performed under multiple logical conditions.

In some data processing problems, the implementation of certain operations depends on the combination of multiple logical conditions, that is, different operations are performed for the combined values of different logical conditions. Decision tables are suitable for such problems.

3.5.1 Example

Go on with the example for vending machine software.

Test cases design steps:

5). Convert graph to a decision table.

a. Take the values of C1, C2, C3 and C4 from small to large in binary system.

	Cause / Input			
#	C1	C2	C3	C4
1	0	0	0	0
2	0	0	0	1
3	0	0	1	0
4	0	0	1	1
5	0	1	0	0
6	0	1	0	1
7	0	1	1	0
8	0	1	1	1
9	1	0	0	0
10	1	0	0	1
11	1	0	1	0
12	1	0	1	1
13	1	1	0	0
14	1	1	0	1
15	1	1	1	0
16	1	1	1	1

b. Analyze whether the intermediate results are valid or not.

The green highlighting rows are the valid input configurations.

	Cause / Input				Middle	
#	C1	C2	C3	C4	Cm1	Cm2
1	0	0	0	0	N	N
2	0	0	0	1	N	N
3	0	0	1	0	N	N
4	0	0	1	1	N	N
5	0	1	0	0	N	N
6	0	1	0	1	Y	Y
7	0	1	1	0	Y	Y
8	0	1	1	1	Y	N

9	1	0	0	0	Y	N
10	1	0	0	1	Y	Y
11	1	0	1	0	Y	Y
12	1	0	1	1	Y	N
13	1	1	0	0	N	N
14	1	1	0	1	N	N
15	1	1	1	0	N	N
16	1	1	1	1	N	N

c. Obtain the following simplified version (that is, when intermediate results Cm1 and Cm2 are true).

	Cause / Input				Effect / Output		
#	C1	C2	C3	C4	E1	E2	E3
1	0	1	0	1	T	F	T
2	0	1	1	0	T	T	F
3	1	0	0	1	F	F	T
4	1	0	1	0	F	T	F

6). Design test cases from decision table.

#	Input	Expected Result
1	Put in $2 coins, Press the "Sprite" button.	Refund $0.50 coin, Dispense Sprite
2	Put in $2 coins, Press the "Coke" button.	Refund $0.50 coin, Dispense Coke
3	Put in $1.5 coins, Press the "Sprite" button.	Dispense Sprite
4	Put in $1.5 coins, Press the "Coke" button.	Dispense Coke

3.5.2 Strengths and Weaknesses

Strengths

The decision table fully consider the combination of input fields, and each rule covers multiple input conditions and consider the constraint relationship of input, the risk of missing measurement is reduced.

Weakness

When too many input items, the number of rules goes up by 2 to the n, the decision table will be very large, using the decision table to merge will cause logic loss, business confusion and error.

3.6 Orthogonal array testing

Orthogonal array testing is a scientific method of designing experiments by selecting a suitable number of representative points from a large number of data.

For example, when test for an account sign up, there are three fields (Username, Password, Confirm password for input. Each field has 2 options (fill in or not fill in). In traditional design, we need to test 2*2*2=8 times. However, use orthogonal table L4(2^3), 4 times can cover the testing.

Composition of orthogonal tables

An orthogonal table is a special form, usually expressed as Ln(m^k). The representation of L is “this is an orthogonal table”, n represents the number of trials or rows of orthogonal tables, k represents the number of factors that can be arranged or the number of columns in the orthogonal table, m represents the maximum number of values that can be obtained by any single factor, and n = k * (m - 1) + 1.

3.6.1 Example

Create an account by provide: Username, Password, Confirm Password.

Test cases design steps:

1). Analyze the factors (variables): Username, Password, Confirm Password.

2). Value of variable: Yes (fill in), No (not fill in).

3). Choose an appropriate orthogonal table: here we use L4(2^3).

0	0	0
0	1	1
1	0	1
1	1	0

Reference:

Orthogonal table online: https://www.york.ac.uk/depts/maths/tables/orthogonal.htm

Available Tools: http://www.pairwise.org/tools.asp

4). Map the value of the variable to the table.

#	Username	Password	Confirm Password
1	No	No	No
2	No	Yes	Yes
3	Yes	No	Yes
4	Yes	Yes	No

5). Take the combination of the various factor levels of each row as a test case.

6). Add a combination of test cases that you think are needed but do not appear in the table.

3.6.2 Strengths and Weaknesses

Strengths:

Test cases are reduced reasonably by using orthogonal test method, so test rime and cost is reduced too.

Weaknesses:

This method treats each state point equally, the key is not prominent. It is easy to spend a lot of time on test design and execution in functions or scenarios that are not commonly used by users, while there is no extra focus on testing in the use of important paths.

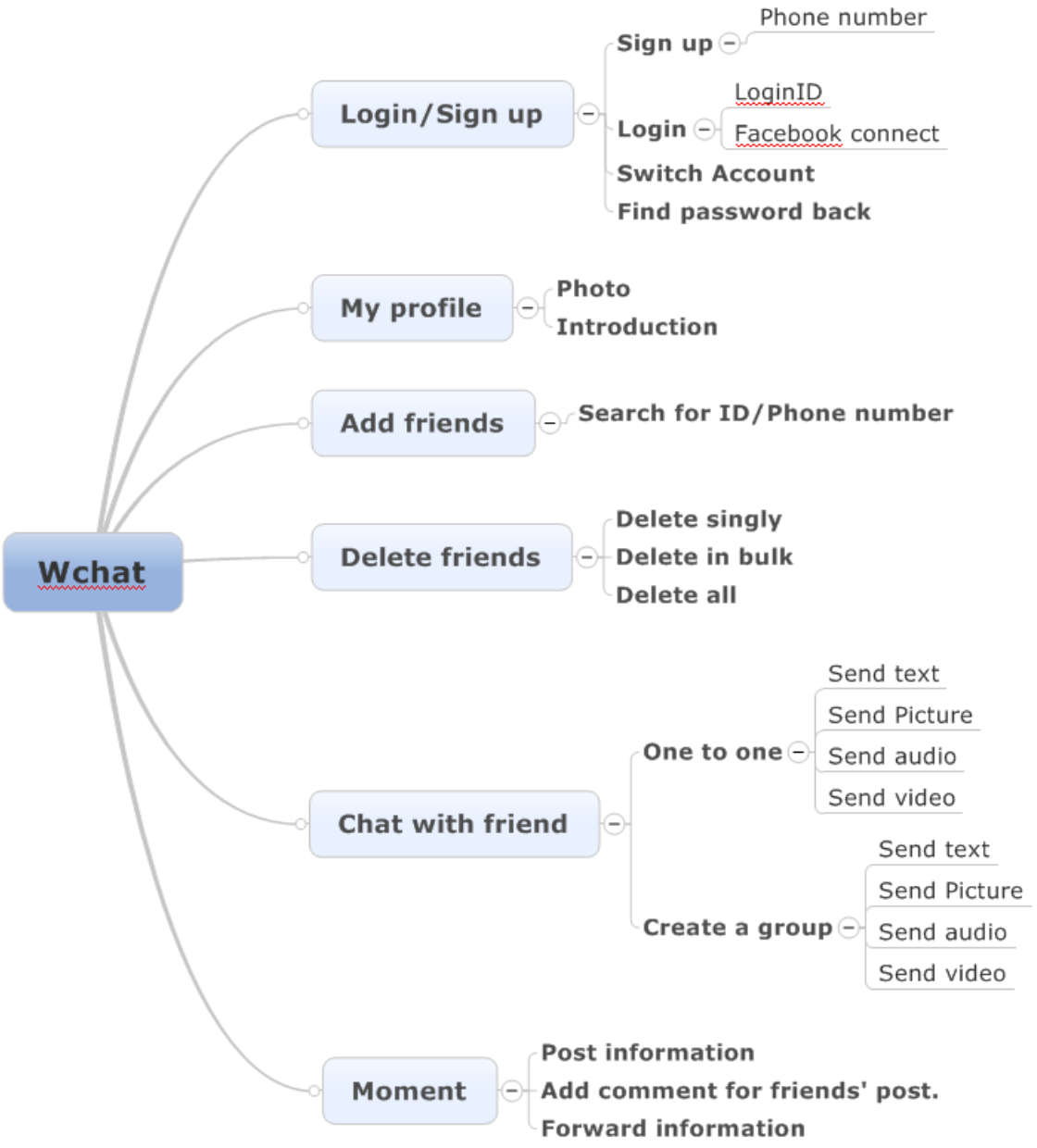
Although the orthogonal test design has the above shortcomings, it can find the optimal level combination through all tests, so it is very popular among practical workers.

3.7 Functional structure diagram

Function structure diagram is used to show the dependency of functions, it is also a feature breakdown structure diagram. Each box in the diagram is a function module or function point. It is recommended to use the form "verb + name" when describing function points

3.7.1 Example

Test for an instant messenger Wchat which has the common functionalities, such as login, add friends, chat with friends, post information, etc.



3.7.2 Strengths and Weaknesses

Strengths

Functional structure can show global structure of production.

Weakness

Only feature breakdown structure, not single points of functionality.

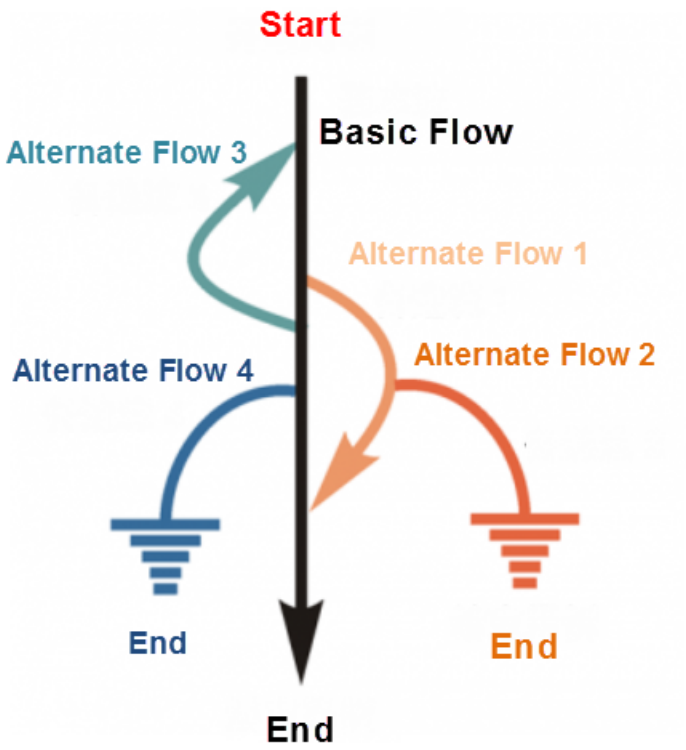
3.8 Scene graph

Almost all software today uses event triggers to control the flow. When an event is triggered, the situation will form a scene, while different triggering orders and processing results of the same event will form an event flow. This idea in software design can also be introduced into software testing, which can more vividly depict the scenario when the event is triggered, which is beneficial for test designers to design test cases and make test cases easier to understand and execute.

The scene approach generally consists of basic and alternate flows, starting with one process and iterating through all the basic and alternative flows to complete all the scenarios.

Basic flow: represented by a straight black line, is the most basic and simple path through the use case (the program starts and ends without any errors).

Alternative flows: in different colors, an alternative flow can start with a base flow, or it can start with an alternative flow, execute under certain conditions, and then rejoin the base flow or terminate the scenario.



Start with the base flow and combine the base flow with the alternative flow to determine the test case scenario:

1	Basic Flow			

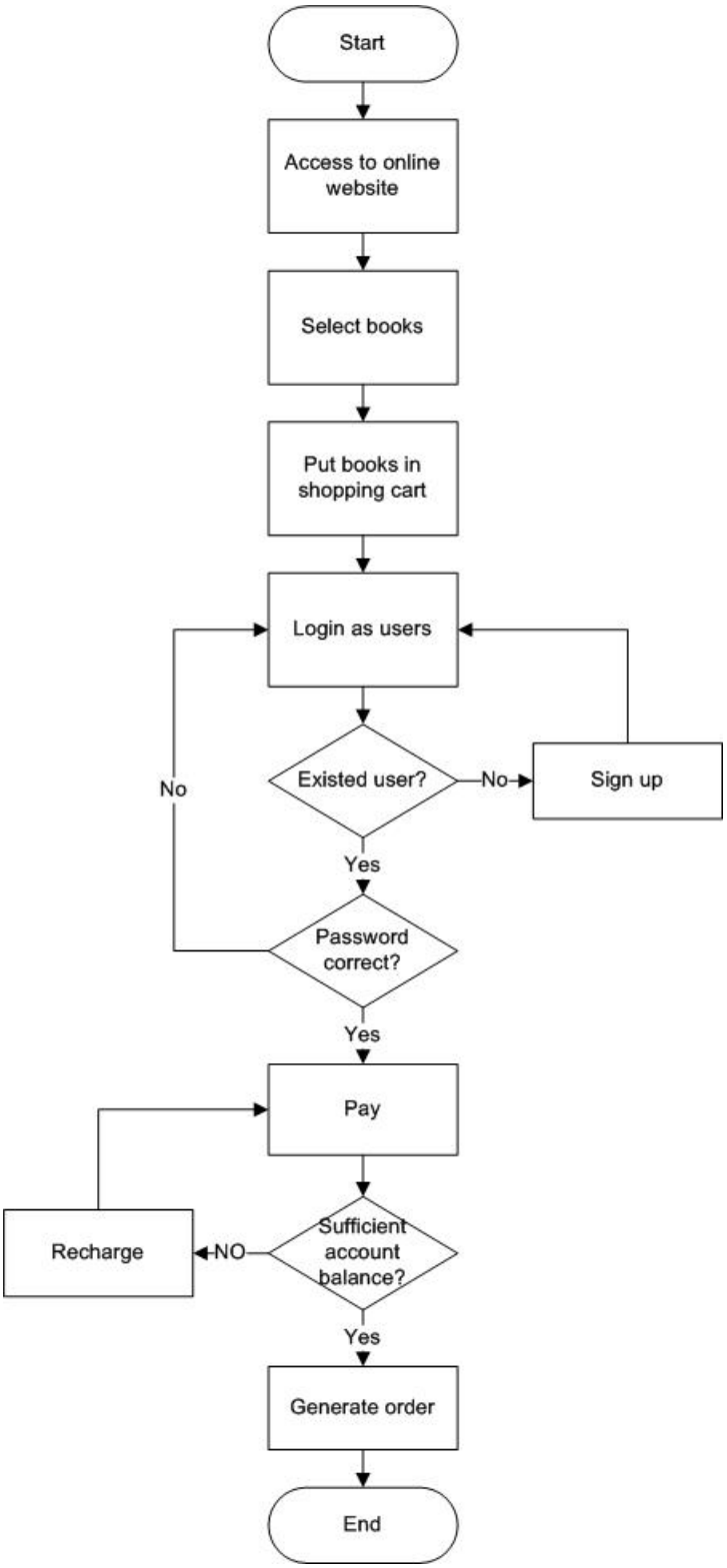
2	Basic Flow	Alternate Flow 1		
3	Basic Flow	Alternate Flow 1	Alternate Flow 2	
4	Basic Flow	Alternate Flow 3		
5	Basic Flow	Alternate Flow 3	Alternate Flow 1	
6	Basic Flow	Alternate Flow 3	Alternate Flow 1	Alternate Flow 2
7	Basic Flow	Alternate Flow 4		
8	Basic Flow	Alternate Flow 3	Alternate Flow 4	

3.8.1 Example

The production is an application of online shopping for books.

Test cases design steps:

1). Draw flow chart according to requirement document.



2). Determine the basic flow and alternative flow according to the flow chart.

Basic Flow	Access to website, Select books, Put books in cart, Login, Pay, Generate order

Alternate Flow 1	Login user doesn't exist
Alternate Flow 2	Username or Password error
Alternate Flow 3	Insufficient account balance

3). Determine the scenario based on the basic and alternative flows.

1	Successful shopping	Basic Flow	
2	User doesn't exist	Basic Flow	Alternate Flow 1
3	Incorrect username or password	Basic Flow	Alternate Flow 2
4	Insufficient account balance	Basic Flow	Alternate Flow 3

4). Generate test cases for each scenario.

Legend: V (Valid), I (Invalid), NA (Not Available)					
#	**Scenarios**	**User name**	**Password**	**Account balance**	**Expected result**
1	Successful shopping	V	V	V	Shop successfully, reduced account balance
2	User doesn't exist	I	NA	NA	Message: The account does not exist
3	Incorrect username	V	I	NA	Message: Incorrect username or password
4	Incorrect password	I	V	NA	Message: Incorrect username or password
5	Insufficient account balance	V	V	I	Message: Insufficient account balance, please recharge

5). Review all test cases generated, remove the redundant, and determine test data values for each test case.

#	**Scenarios**	**User name**	**Password**	**Account balance**	**Expected result**
1	Successful shopping	Kim.Wilson	1s2#plkj	220	Shop successfully, reduced account balance $200
2	User doesn't exist	Pat.Black	Pt123!	NA	Message: The account does not exist, please sign up first
3	Incorrect username	Sue.Green	1V2$	NA	Message: Incorrect username or password
4	Incorrect password	Tom.Wong	Shop666	NA	Message: Incorrect username or password
5	Insufficient account balance	Kim.Wilson	1S2#ERTB	20	Message: Insufficient account balance, please recharge

3.8.2 Strengths and Weaknesses

Strengths

Business processes can be overridden.

Weaknesses

Only validate business processes, not single points of functionality.

In general, the methods of equivalence class, boundary value, error guessing, decision table and orthogonal array are adopted to verify the single point of function, then the scene method is adopted to verify the business process.

4 General principles

Here are some general principles for when to use which test design techniques.

1. If the variables are independent, you can divide the domain and use equivalence partitioning. Turn an infinite number of tests into a finite number of tests.

2. Boundary value analysis is a method that used in any software test.

3. Error guessing helps us to supplement the original test cases with additional ones.

4. If the combination of input conditions needs to be considered, the causal graph method and the decision table method can be used

5. If the relationships between inputs and outputs are large, orthogonal array method can be used to reduce test cases.

6. If the functionalities of a system are complicated, functional structure can be used to breakdown functionalities.

7. The system with clear business process can choose scene method to design test cases.

8. Check the designed test case logic coverage against the program logic, and add enough test cases if the required coverage standards are not met.

References

Books:

Glenford J. Myers, Tom Badgett, Corey Sandler. The Art of Software Testing (Third Edition)

GuXiang. 2017, Software testing technology in action, China post and telecommunications press

Cem Kaner, Jack Falk and Hung Quoc Nguyen. 2006 Testing Computer Software (Second Edition)