Understanding Software Test Cases

Techniques for better software testing

Josh Kounitz

Elementool

www.elementool.com

Understanding Software Test Cases

Techniques for building good test cases

"To err is human, but to really foul things up you need a computer."

- Paul Ehrlich

Building test cases is about doing your best to find the worst in a program in order to deliver a quality product to the customer. Without a well-designed and executed test plan, the results can be catastrophic and very costly. For example, because of an error in the software, the first flight of the Ariane 5 rocket resulted in its complete destruction and the loss of four satellites at a cost of more than $370 million[1].

In the U.S. alone, software bugs cost the economy $59 billion[2] back in 2002. Clearly, software defects must be found and corrected before unleashing new programs to the customer.

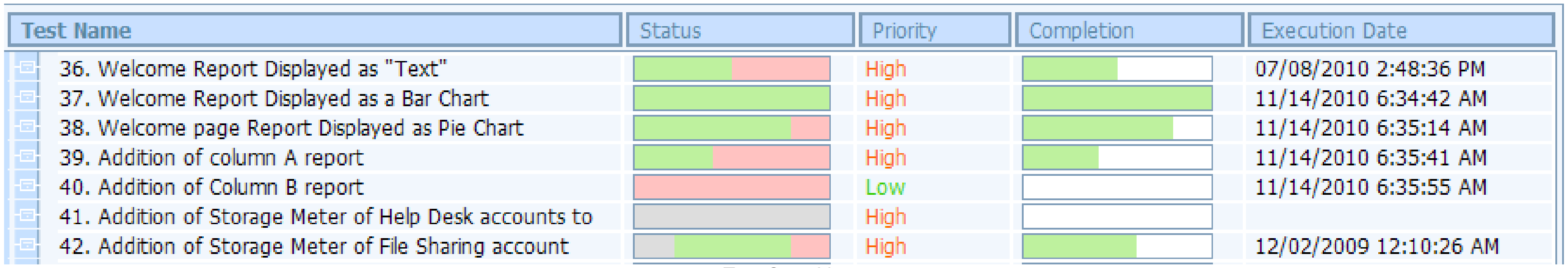
This eBook provides you with a basic understanding of software test cases. In addition, the eBook describes proven methods for building test cases that uncover the most defects with the minimum of time, effort and money.

What is a Test Case?

A test case has an input, an action and an expected result. In using test cases, the tester is trying to break the application. The whole point of using test cases is to find defects. Hopefully, serious defects that crash the system are found before your application is released to the customer. These show stoppers, or defects that may delay release of the application, must be found and fixed in order to save time and money and prevent customer dissatisfaction.

Test cases must exercise every feature of the application to prevent defects from being released. Each test case needs to contain a set of test steps of a feature or function. At the end of the test the expected results are compared to actual results to determine if the application is working as it should.

A test case can have information that includes the test case name, goal, environment, steps to take, input and expected results. The following is an example of a test case description from a test case management tool.



Test Name	Status	Priority	Completion	Execution Date
36. Welcome Report Displayed as "Text"		High		07/08/2010 2:48:36 PM
37. Welcome Report Displayed as a Bar Chart		High		11/14/2010 6:34:42 AM
38. Welcome page Report Displayed as Pie Chart		High		11/14/2010 6:35:14 AM
39. Addition of column A report		High		11/14/2010 6:35:41 AM
40. Addition of Column B report		Low		11/14/2010 6:35:55 AM
41. Addition of Storage Meter of Help Desk accounts to		High		
42. Addition of Storage Meter of File Sharing account		High		12/02/2009 12:10:26 AM

Test Case List

A good test case consists of several ingredients: its chance of finding an error is high, it doesn't copy other test cases, it's most likely to find error or defects, it's not too simple or too complex and it is obvious to a tester when an application failure has occurred.

Test Cases and the Attitude of the Software Testers

There is a profound connection between creating and using test cases and the attitude of the tester. Test cases are about finding errors and breaking software. However, most people are more interested in building something than in breaking something. For that reason, we must be careful with how we develop and use test cases.

The goal of using test cases should be to point out the errors in a program. If this goal is set, then there is a higher probability of finding defects. As Myers and Sandler point out in their book *The Art of Software Testing*, "This has implications for how test cases should be designed and who should and who should not test a given program." Furthermore, the authors write that testers should think of a program as a sick patient. In this manner, the goal becomes to test the "patient" in order to find what's wrong.

This is an important distinction from trying to prove that the "patient" is OK. In the latter scenario, the tester's mindset is to test the program to prove that it works. Testers require the proper goal to direct their behavior. If the goal is to prove that the program works, then the tester will find fewer defects. If the goal is to find defects, then they will find them. In other words, don't test a program to prove that it works. Rather, test the program with the intent to find defects.

Test Cases and Money

Before we jump into the nuts and bolts of creating and using test cases, let's discuss the economics of test cases. Time is money, and running a multitude of test cases is costly and time-consuming. All but the smallest programs have a myriad of possible outcomes for each input value. For that reason the infinite possibilities must be pared down to be economically feasible. In short, test cases should cover the most possibilities with the fewest required test cases.

When Should You Start Writing Test Cases?

Writing test cases is one of the most important things to do at the beginning of the testing process. The task of writing the test cases makes you examine what you need to test. Also, it can help point out problems and errors in the requirement and design specifications. Writing a test case also makes you think how each component of the application works, its integration with the application and the function of the application as a whole. This is due to the fact that in order to write the test cases, you must first understand the required input and output of each feature and how each component is integrated with the application.

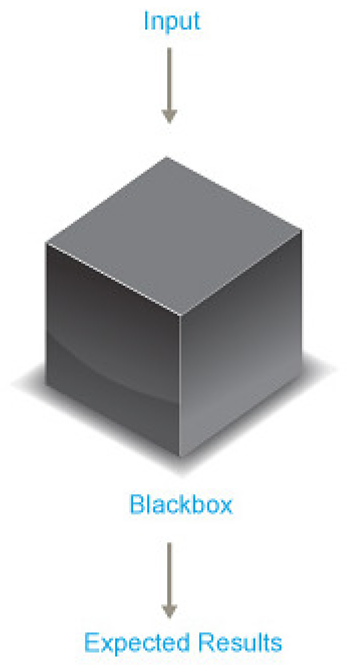
In more traditional development testing, and the writing of test cases, requirements specifications are finished and the project is code complete before testing begins. More recent software development methods require testing and test cases to be defined as the developers complete each part of the application. For example, upon adding a new feature to a GUI, the tester would then write and execute test cases to see if it's possible to break the application by using the new feature. This may involve something as straightforward as examining expected results after inputting simple data. A more complex example might involve a test case that checks the operation of servers over the Web while trying to conduct multiple financial transactions and deal with thousands of hits on the website. This is called Load Testing, and we'll discuss more about it in a different eBook.

Techniques for Developing Test Cases

There are many different ideas and techniques used for creating test cases. In the real world, however, you are likely to use several different techniques in any one project. To begin, let's discuss the two broadest categories of test cases: white-box test cases and black-box test cases.

Black-box testing and test cases

Black-box testing uses test cases that check known inputs against expected results. It’s as if the application code was in a black box and you do not care and cannot obtain knowledge about the internal logic and structure of the application. Rather, your goal is to write test cases that find the circumstances where the application does not respond as required, revealing a bug or defect. Following is a diagram illustrating the process of black-box testing.



Some of the most popular methods of black-box testing and test cases are discussed in the following sections.

Boundary-value analysis (BVA)

This black-box technique is based on the assumption that if the software functions correctly for the boundary values, it will function as well for the values that lie in between. It helps us design test cases to focus on the edges of the equivalence classes, which are described in the next section.

Mistakes can occur easily at a boundary of equivalence classes. Fixing these defects requires that you focus testing and test cases on these boundaries. In addition, you must create test cases at the extremes, or edges, of the equivalence classes. A boundary value can contain the smallest or largest input, internal value or result for a system or part of a system.

Equivalence partitioning

QA departments need to reduce costs. As a result, one of the goals of testing is to try not to write redundant test cases. Equivalence partitioning can reduce the number of test cases that are needed. First, separate the input of a program into classes. Each equivalence class should result in an identical answer. Then design test cases so program input is within these equivalence classes.

Equivalence partitioning tackles the problem of having to make extensive and costly black-box testing. This method assumes similar input results in similar output. Rather than test each input, the input field is divided into equivalence classes with each input belonging to one class only. Then, one or more test cases are created for testing each class.

Decision tables

Decision tables record complex business rules in the program. The tables represent possible input conditions. Events trigger actions associated with the business rules. Each column in the decision table is a combination of rules. A unique combination of inputs triggers action associated with the rules. Finally, every rule or column should be a test case.

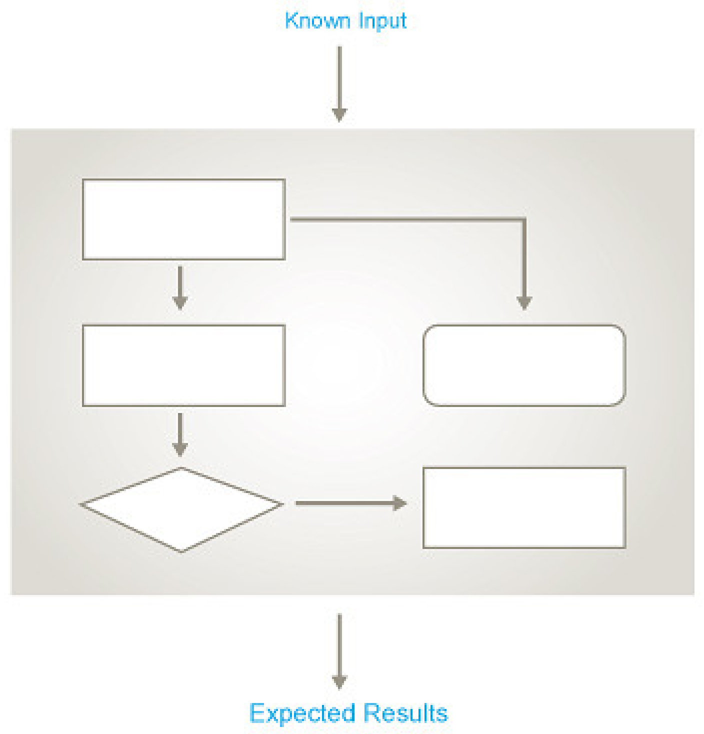
Diabolical test cases

Just as the name suggests, diabolical test cases seek to break the system by using data with extreme values to crash the system.

White-box testing and test cases

White-box testing and test cases are primarily concerned with achieving the broadest possible coverage of the source code. Therefore, the testers must be familiar with the logic of the application and use this knowledge to create test cases that execute as much of the code as possible. Again, the goal is to break the system.

White box test cases test different control flow paths in a program. The test cases also test decision points (true and false), execute loops, and check internal data structures of the application. Basis path testing, equivalence partitioning, and boundary value analysis are all used to create white box tests. Equivalence partitioning divides the set of possible input values into equivalence classes. Only a value from each of the equivalence classes needs to be tested. Boundary value analysis looks at testing around a set boundary. The following diagram illustrates the process of white-box testing.



Error guessing

This technique relies on the experience and intuition of the tester to foresee the defects that may exist in the program. Then test cases are designed to reveal these defects.

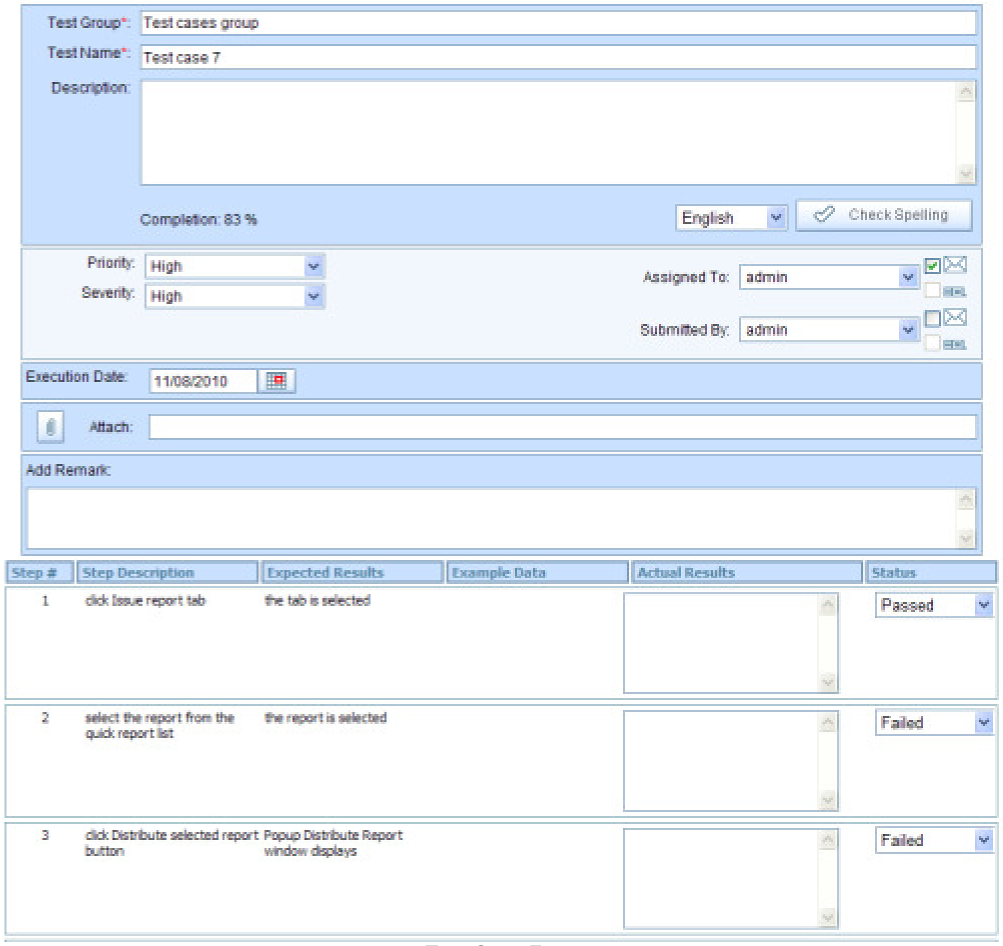
Writing test cases

Written specifications and user documentation can provide you with excellent information for making test cases. Later, you can write more test cases based on the function and flow of the application. At this point, you are ready to group test cases together to form a test procedure. Finally, you can automate the running of test cases for regression testing. This way the testers and others in QA can work on checking new functionality.

Following is a simple example of fields that you commonly see in test cases.

- Test case ID
- Unit to test
- Assumptions
- Test data
- Steps to be executed
- Expected result
- Actual result
- Pass/Fail
- Comments

Additional fields may be created, including those for verification and identification. Finally, good test cases do not try to find all the bugs, but rather focus on what is good for your purposes. You want to answer the question, “Did this program do what it’s supposed to do without error?”


Test Group*: Test cases group

Test Name*: Test case 7

Description:

Completion: 83 %

English

Priority: High

Severity: High

Assigned To: admin

Submitted By: admin

Execution Date: 11/08/2010

Attach:

Add Remark:

Step #	Step Description	Expected Results	Example Data	Actual Results	Status
1	click Issue report tab	the tab is selected			Passed
2	select the report from the quick report list	the report is selected			Failed
3	click Distribute selected report button	Popup Distribute Report window displays			Failed

Test Case Form

Testing Cycle

The testing cycle consists of several stages, each of which is described in the numbered list below. The use of test cases in the testing cycle is fundamental.

First, a comprehensive test plan is created by developers and QA. Next, the manager assigns test case groups to testers according to the test plan. The testers follow the test plan and run the test cases. The results of the tests (passed, failed) are submitted for scrutiny by developers and QA. Any failed tests are reported as bugs to the bug tracking system. The developers fix these problems and send the improved system to the testers for defect retesting. Finally, regression testing is conducted to see if the software is still working after the latest modifications.

The software testing cycle varies between organizations. However, the testing cycle and the use of test cases usually progresses through the following stages :

1. Requirements analysis : During this design stage, testers and developers work together to see how well the design can be tested and within what parameters.
2. Test planning : The testing cycle requires a plan to organize testing efforts, including the creation of test cases.
3. Test development : Test cases are created to test the software.
4. Test execution : Testers run the software, including various test cases. Defects are reported.
5. Test reporting : Testers write their reports regarding the readiness of the software for release.
6. Test result analysis : Testers and customers determine which defects to fix and which can be delayed.
7. Retesting Defects : After a defect has been identified and corrected, test cases are run again by testers.
8. Regression testing : Test cases are run to determine if the software is still working after changes have been made to the code.
9. Closure : Documents, results, test plans, etc. are archived once the software passes all the tests necessary for release.

Managing Test Cases

Managing testing and test cases is important to the success of QA. Poorly managed tests can cost your company time and money. You need to organize, manage, track and perform several other tasks as well, such as assigning test cases to colleagues, print test reports, etc. A well-managed QA department is crucial to the success of your software and your company.

Conclusion

Well-designed test cases are the most important tools for discovering defects in software. These tools give you the ability to prevent serious defects, and even minor ones, from being shipped to customers. Good test cases save you time and money and maybe even the reputation of your organization.

---

[1] http://www.cse.unsw.edu.au/~se4921/PDF/ariane5-article.pdf

[2] http://www.qualitydigest.com/aug02/news.shtml#6