Software testing Level 2

Notes

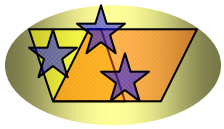
for

City & Guilds

7540 Unit 012

Tench Computing Ltd
Pines
Glendale Road
Burgess Hill
West Sussex
RH15 0EJ

Web address: www.tenchcomputing.co.uk
Email address: jtench@globalnet.co.uk



About the author: *Jackie Tench MSc, ACIB, Cert Ed*

Jackie started her working career in branch banking with the Midland Bank (now HSBC) and was transferred to their Computing Department after achieving 100% in their ability test for programmers. She then worked for more than a decade in this department and was one of the first women to achieve a junior management grade at the age of 21. She attended a significant number of IBM programming training courses during her time there.

Jackie was the first woman to pass the ACIB (Associate Chartered Institute of Bankers) examinations in the Midland Bank (HSBC) and the youngest person at 21 years of age.

Jackie then left to raise a family but still found time to teach part-time at a college in Sheffield and to obtain a MSc in Computing and a Cert Ed in teaching.

When her children were old enough Jackie returned to work full-time and was a Senior Lecturer in Software Engineering and Computer Studies at a college in Brighton for nearly 10 years teaching all levels up to and including HND.

Therefore, Jackie has considerable business knowledge and qualifications plus wide experience in practical computing and training – covering areas such as structured design, analysis, coding, testing and implementing software applications plus training students to fulfil an important role in the computer industry.

Jackie has worked as a consultant for several blue chip companies and examination boards using her software engineering and educational training skills and is now one of the foremost experts in computing with an extensive knowledge of programming languages and applications.

Contents Page

Testing

Purpose of testing

The purpose of testing is to prove that the software works under normal and exceptional conditions and that it complies with the software design specification.

In practice most software is not 100% free of errors and it is difficult to test every situation that may occur while a program is running. But, it is important to find and eliminate as many errors as possible.

The testing should be done in a structured way and not haphazardly. If testing is done in a structured way then any errors in the software should be found. Also if the testing is structured then when an error is found it will be known what caused the error because the testing procedures have been documented.

Testing done using a standardised and rigorous approach will produce software that is more reliable.

Types of testing

Logical testing

Logical testing is done to prove that the logic of a software component is correct. Access to the program code is required to be able to plan the test data required to test the logic of the component. This type of testing is also known as ‘white box testing’ because the logic of the program must be seen in order to plan the testing.

To be able to undertake logical testing a tester needs to understand the programming language in which the software is written. There are numerous programming languages used for writing software. Programming languages all have similar functionality but the syntax of the language can be very different.

Functional testing

Functional testing is carried out independently of the code used in the program. The software design specification is used to construct test data that covers all the types of input and software functions. This type of testing is also know as ‘black box testing’ because the internal code of the software component does not need to be seen.

To undertake functional testing a tester has to examine the software design specification to find out the function of the software, the input data that is used and the output that is required for the given input. The input data could be entered via the keyboard or read from a file created by other software. The output could be displayed on screen, printed or written to a file.

This unit deals with functional testing.

Recovery testing

Where data is being manipulated by a software component, backup procedures should be in place to backup the data at regular intervals in case of failure of hardware or software. Using backup procedures to backup data means that at some future date if there is a failure the data will need to be restored. Recovery procedures should be in place for software so that in the event of the failure of a software component or hardware, the data can be restored if it was corrupted during the failure. These recovery procedures should be tested before software becomes live to ensure that they work correctly and that data can be restored. It is too late after a failure occurs to find that recovery procedures do not work.

Performance testing

In large systems for instance a database enquiry system the response times for an enquiry must be kept to a minimum. Performance testing involves generating large volumes of data to ensure that a system can cope with the volumes created and also respond within a reasonable time. This may involve writing software to generate the data with which to test the system. This also tests the capacity of the system i.e. how much data can it cope with without crashing or locking up.

Test strategies

The two most common testing strategies are top-down and bottom-up. Both of these strategies can be used for small programs or for a large-scale project with a team of software developers.

The order in which the components are coded needs to be planned so that the testing proves the software works, for example it is no good coding all the input modules first because without any output it is difficult to prove that a component works.

Logical testing can be done on each individual software component before it is combined with other components for functional testing.

Top-down

Top down testing is where the skeleton of the whole program is tested. A ‘stub’ replaces each individual software component. Each stub is a skeleton of a software component that contains no code but enough information to be compiled without errors. The stub may contain a line to display a message with the component name to prove that the stub was entered and would be executed if it contained code. As individual components are coded the code replaces the stub and that component is tested. So, the software is built from the top down until all the components have been written and tested.

This method of testing means that software is tested incrementally and helps to isolate the errors. When a stub is replaced with code a test is done and if any errors are found the first place to look for the errors is in the last stub to be replaced.

Bottom-up

Bottom-up testing is where individual components are tested as standalone units. As individual components are tested they are combined together until the whole program can be tested.

This method of testing means that software is tested incrementally and helps to isolate errors. The last component added is the first place to look for any errors.

Test plan

A test plan should be completed to show what testing is to be done.

The test plan should include the

- software component name
- version number
- tester name
- page numbers
- date
- test number
- purpose of the test
- input test data
- expected result from that input.

If the input test data and the expected result are small they can actually be written on the test plan. If the input test data and expected results are too large to include on the test plan then a number to cross reference them should be inserted on the test plan and on the prepared test data.

A well thought out test plan, with test data, once prepared can be used throughout testing even for repeated tests and later maintenance

Test data

Tests should be done to test the software under normal operating conditions, under exceptional conditions and boundaries should be tested for input data.

Exceptional conditions are tested to make sure that if a user does something wrong the software will not crash or lose data or the data does not become corrupted. An example of an exceptional condition is if a user tries to open a file that does not exist. The software should not allow this to happen because it will cause the software to crash; instead an error message should appear on the screen to tell the user that they are trying to open a non-existent file.

Boundaries around input data must be tested. If a field can have only values between 100 and 1000 (inclusive) as input values then validation should be applied to the field. Testing should be done to ensure that only these values are allowed. Testing the boundaries means that the values 99, 100, 101, 999, 1000 and 1001 should be entered. These values will test the boundaries of the input data. It is the boundary input that is most likely to cause an error. The values 100, 101, 999 and 1000 should be accepted as valid data and the values 99 and 1001 should be rejected as invalid data with an error message.

Expected results are the results that are expected as output given the stated input. The expected results have to be worked out manually. So, if the input is an action such as a click on a command button to produce a report, the result for the report should be worked out and would be the expected results. The expected results would then be compared with the actual results to confirm that the software works correctly.

Test data must be prepared which will test the conditions required and fulfil the tests shown in the test plan.

Enough test data must be used to test the software component. Never enter too much test data because if a test results in incorrect output the data may have to be entered again or amended.

An example of a blank test plan page is shown on the next page. For a large software application a test plan can run to many pages.

TEST PLAN

SOFTWARE COMPONENT NAME: **DATE:**		**VERSION NO:** **TESTER NAME:**	**PAGE NO:**
Test No	**Purpose/Type of Test**	**Input**	**Expected Result**

Test log

A test log is used to track the testing as it is being done.

A test log should include the

- software component name
- version number
- tester name
- page numbers
- test number
- date
- actual results
- comments on any discrepancies between the expected results and the actual results.

If any discrepancies are found then the cause of the discrepancy must be found and corrected by the developer and the test must be done again to check the results. Sometimes a test may have to be done several times before the problem is resolved.

Output from a test e.g. report, file listing, screen print, must have the test number and date written on it so that it can be cross-referenced to the test log.

A new test log is used for each batch of testing. A test log provides a complete record of all the testing done on a system.

An example of a blank test log page is shown on the next page. For a large software application a test log can run to many pages.

Test plan and test log forms

These forms can be created in a word processor using tables and filled in using the word processor. The rows can be made as large or as small as required.

Target environment

Software can initially be tested in a test environment. Eventually it must be tested in the target environment. This is to ensure that it works in the target environment, which could be different, in terms of either software or hardware, to the test environment. The fact that the target environment is different may not be realised until a test is done using it. For example, the target environment may have the same operating system but a different (later) version of the operating system.

TEST LOG

SOFTWARE COMPONENT NAME:			VERSION NO: TESTER NAME:	PAGE NO:
Test No	**Date**	**Actual Output**	**Comments on discrepancies**	