

Chapter 1 Principles of Testing

1.1 OVERVIEW

In this module you will get an overview of the most fundamental principles of testing.

Firstly, we point out that although you will come across different terminology throughout your career, professional testers are all pretty much agreed on these basic ideas that we describe in this module.

Secondly, we take a look at the need for proper testing look at the cost of getting it wrong and we show why exhaustive testing is neither possible nor practical. What are errors and how do they get into the software?

Thirdly, we describe a fundamental test process, based on industry standards, and underline importance of planning) tests and determining expected results in advance of test execution.

We conclude with a look at re-testing and regression testing; why it is so important to go that extra mile and run a suite of tests again, just when you thought it was safe to hit the release date.

1.2 OBJECTIVES

After completing this module you will:

- » Understand basic testing terminology.
- » Understand why testing is necessary.
- » Be able to define error, fault and failure.
- » Appreciate why errors occur and how costly they can be.
- » Understand that you cannot test everything and that testing is therefore a risk management process.
- » Understand the fundamental test process.
 - » Understand that developers and testers have different mindsets.
 - » Learn how to communicate effectively with both developers and testers.
 - » Find out why you cannot test your own work.
 - » Understand the need for regression testing.
 - » Understand the importance of specifying your expected results in advance.
 - » Understand how and why tests should be prioritized.

1.3 TESTING TERMINOLOGY

There is no generally accepted set of testing definitions used by the worldwide testing community. The British Standard for Software Component Testing published in August 1998 provides a new source of testing definitions. When involved in a testing project it is important to ensure everyone understands terminology adopted; you may find different

ISTQB Certification Preparation Guide: Chapter 1 – Principles of Testing

terminology is used in your organization.

Exercise

The British Standard for Software Component Testing is known as BS_____

A useful glossary of terms used in software testing is called BS_____

Although it is a British Standard published by the BSI (British Standards Institute), the Specialist Interest Group in Software Testing (SIGIST) developed it over a period of several years.

ANSWERS: 7925-PART2, 7925-PART1

1.4 WHY IS TESTING NECESSARY?

This section explains why testing is necessary and closely at the cost and consequences of errors in computer software.

1.4.1 Definitions Errors, Faults and Failures

An error is a human action that produces an incorrect result. A fault is a manifestation of an error in software. Faults are also known colloquially as defaults or bugs. A fault, if encountered, may cause a failure, which is a deviation of the software from its existing delivery or service.

We can illustrate these points with the true story Mercury spacecraft. The computer program **aboa** spacecraft contained the following statement **wril** the FORTRAN programming language.

```
DO 100 i = 1.10
```

The programmer's intention was to execute a succeeding statements up to line 100 ten times then creating a loop where the integer variable I was using the loop counter, starting 1 and ending at 10.

Unfortunately, what this code actually does is writing variable *i* do to decimal value 1.1 and it does that once only. Therefore remaining code is executed once and not 10 times within the loop. As a result spacecraft went off course and mission was abort considerable cost!

The correct syntax for what the programmer intended is

```
DO 100 i =1,10
```

Exercise

What do you think was the error, fault and failure in this example?

The error is _____

ISTQB Certification Preparation Guide: Chapter 1 – Principles of Testing

The fault is _____

The failure is _____

1.4.2 Reliability

Reliability is the probability that software will not cause the failure of a system for a specified time under specified conditions. Measures of reliability include MTBF (mean time between failure), MTTF (mean time to failure) as well as service level agreements and other mechanisms.

1.4.3 Errors and how they occur

Why do we make *errors* that cause *faults* in computer software leading to potential *failure* of our systems? Well, firstly we are all prone to making simple human errors. This is an unavoidable fact of life. However, this is compounded by the fact that we all operate under real world pressures such as tight deadlines, budget restrictions, conflicting priorities and so on.

1.4.4 Cost of errors

The cost of an error can vary from nothing at all to large amounts of money and even loss of life. The aborted Mercury mission was obviously very costly but surely this is just an isolated example. Or is it? There are hundreds of stories about failures of computer systems that have been attributed to errors in the software. A few examples are shown below:

A nuclear **reactor was shut down** because a single line of code was coded as $X = Y$ instead of $X = \text{ABS}(Y)$ i.e. the absolute value of Y irrespective of whether Y was positive or negative.

Blue Cross of Wisconsin installed a new \$200m claims processing system - it sent out \$60 million in unwarranted and **duplicate payments**. Also, when a data entry clerk typed 'none' in the town field the system sent hundreds of checks to the non-existent town of 'NONE' in Wisconsin.

In May 1992, Pepsi ran a promotion in the Philippines. It told customers they could win a million pesos (approx. \$40,000) if they bought a bottle of Pepsi and found number 349 stamped on the underside of the bottle cap. Unfortunately, due to a software error, 800,000 bottle caps were produced with number 349 instead of one, which was an equivalent of \$42 billion in prize money. It cost the company dearly as some people pursued their claims through the courts and **Pepsi paid out millions of dollars in compensation**.

Another story was printed in the New York Times on 18th February 1994. Chemical Bank managed to allow \$15 million to be withdrawn incorrectly from 100,000 accounts - a **single line error** in the program caused every ATM on their network to process the transaction twice.

ISTQB Certification Preparation Guide: Chapter 1 – Principles of Testing

1.4.5 what happened on October 26th & 27th 1992?

The **London Ambulance Service (LAS)** covers an area of just over 600 square miles and is the largest ambulance service in the world. It covers a resident population of some 6.8 million, but its daytime population is larger, especially in central London. Since 1974 South West Thames Regional Health Authority has managed it.

The LAS carries over 5000 patients every day and receives between 2000 and 2500 calls daily (including between 1300 and 1600 emergency calls i.e. 999 calls). In terms of resources the LAS has some 2700 staff, 750 ambulances and a small number of various other vehicles including 1 helicopter. LAS make almost 2 million patient journeys each year. In 1992/1993 its budgeted income was £69.7 million.

On the 26th and 27th October 1992 the new system failed, ambulances failed to turn up and people lost their lives. Although no Coroners Court ever apportioned blame for any deaths directly to the computer systems failure, it was by any standards a major disaster and made the main evening news bulletins on several occasions.

1.4.6 London Ambulance Service

In summary the problems were:
Computer Aided Dispatch - 1

The system relied on near perfect information to propose optimum resource to be allocated to an emergency. However, there were imperfections in the information and changes in operational procedures made it difficult for staff to correct the system.

This was not a problem when it went live at 7 am on 26th October 1992 as the system load was light; however as the number of emergency calls increased throughout the-day it became increasingly difficult for staff to correct errors; this led to:

- Poor, duplicated and delayed allocations.
- Build-up of exception messages and awaiting attention list.
- Slow-down of system as messages and lists built up.
- Increased number of callbacks and hence delays in telephone answering.

The cost of these errors were ultimately that ambulances didn't turn up and people lost their lives although the official enquiry report did not attribute any fatalities to the system problems. The costs in terms of loss of confidence in the computer system, industrial relations and so on were probably also high.

1.4.7 Exhaustive testing why not test everything?

It is now widely accepted that you cannot test everything. Exhausted testers you will find, but exhaustive testing you will not. Complete testing is neither theoretically, nor practically possible. Consider a 10-character string that has 280 possible input streams and corresponding outputs. If you executed one test per microsecond it would take approx. 4

ISTQB Certification Preparation Guide: Chapter 1 – Principles of Testing

times the age of the Universe to test this completely. For a survey of the methodology and limitations of formal proofs of program correctness see [Manna 78]

1.4.8 Testing and risk

How much testing would you be willing to perform if the risk of failure were negligible? Alternatively, how much testing would you be willing to perform if a single defect could cost you your life's savings, or, even more significantly, your life? [Hetzel 88].

The amount of testing performed depends on the risks involved. Risk must be used as the basis for allocating the test time that is available and for selecting what to test and where to place emphasis. A priority must be assigned to each test.

Test Managers and Project Managers come up with different prioritization schemes but the basic principle is that you must focus the testing effort on those areas of the system that are likely to have the most defects. Another key principle is that you must execute the most important test first. Otherwise, if you run out of time, which is likely, you will not have exercised the tests that give you the best payback in terms of faults found.

1.4.9 Testing and quality

Testing identifies faults whose removal increases the software quality by increasing the software's potential reliability. Testing is the measurement of software quality. We measure how closely we have achieved quality by testing the relevant factors such as correctness, reliability, usability, maintainability, reusability, testability etc.

1.4.10 Testing and legal, contractual, regulatory or mandatory requirements

Other factors that may determine the testing performed may be legal, contractual requirements, normally defined in industry specific standards or based on agreed best practice (or more realistically non-negligent practice).

1.4.11 How much testing is enough?

It is difficult to determine how much testing is enough. Testing is always a matter of judging risks against cost of extra testing effort. Planning test effort thoroughly before you begin, and setting completion criteria will go some way towards ensuring the right amount of testing is attempted. Assigning priorities to tests will ensure that the most important tests have been done should you run out of time.

1.5 FUNDAMENTAL TEST PROCESS

1.5.1 Introduction

Testing must be planned. This is one of Bill Hetzel's 6 testing principles [Hetzel 88 p25] and he says we are all agreed on this one. However, he points out that the problem is that most of us do not discipline ourselves to act upon it. Good testing requires thinking out an overall

ISTQB Certification Preparation Guide: Chapter 1 – Principles of Testing

approach, designing tests and establishing expected results' for each of the test cases we choose.

You have seen already that we cannot test everything, we must make a selection, and the planning and care we expend on that selection accounts for much of the difference between good and poor testers.

The quality and effectiveness of software testing are primarily determined by the quality of the test processes used [Kit 95]. This is one of Ed Kit's 6 essentials of software testing. Test groups that operate within organizations that have an immature development process will feel more pain than those that do not. However, the test group should strive to improve its own internal testing processes. This section of the course shows a fundamental test process, based on the BS7925-2 standard for software component testing.

The fundamental test process comprises planning, specification, execution, recording and checking for completion. You will find organizations that have slightly different names for each stage of the process and you may find some processes that have just 4 stages, where 4 & 5 are combined, for example. However, you will find that all good test processes adhere to this fundamental structure.

1.5.2 Test process stages

See BS7925-2 for diagram of test process. Test planning involves producing a document that describes an overall approach and test objectives noting any assumptions you have made and stating any exceptions to your overall test strategy for your project. Test planning can be applied at all levels. Completion or exit criteria must be specified so that you know when testing (at any stage) is complete. Often a coverage target is set and used as test completion criteria.

Test specification (sometimes referred to as test design) involves designing test conditions and test cases using recognized **test techniques** identified at the planning stage. Here it is usual to produce a separate document or documents that fully describe the tests that you will carry out. It is important to determine the **expected results** prior to test execution.

Test execution involves actually running the specified test on a computer system either manually or by using an automated test tool.

Test recording involves keeping good records of the test activities that you have carried out. Versions of the software you have tested and the test specifications are software you have tested and the test specifications are recorded along with the actual outcomes of each test.

Checking for test completion involves looking at the previously specified test completion criteria to see if they have been met. If not, some test may need to be rerun and in some instances it may be appropriate to design some new test cases to meet a particular coverage target.

Note that BS7925-2 does not specify the format of any test documentation. However, The IEEE standard, known as 829, specifies in detail a standard for software test documentation.

ISTQB Certification Preparation Guide: Chapter 1 – Principles of Testing

BS7925-2 shows a diagram of a suggested hierarchy of test documentation.

HOME WORK

Exercise

Putting aside management problems. Read through test documentation examples in BS7925-2 and answer following questions:

What test techniques does component test strategy stipulate?

What percentage of decision coverage is required?

What should be done if errors are found?

The project component test plan is useful because the approach outlined allows:

- a) **Strict adherence to the component test strategy**
- b) **More faults to be identified in the LOG components**
- c) **A basic working systems to be established as early as possible**
- d) **Isolation of the components within the test strategy**

The component test plan must consist of a single document? TRUE/FALSE

The component test plan must specify test completion criteria? TRUE/FALSE

Why does component test plan specify 100% DC whereas strategy required 90%?

Which test case deals with non-numeric input?

List the expected outcome and the test condition

Why does the CTP have additional/altered test cases?

What action has been taken as a result of the test report?

1.5.3 Successful tests detect faults

As the objective of a test should be to detect faults, a successful test is one that does detect a fault. This is counter-intuitive, because faults delay progress; a successful test is one that may cause delay. The successful test reveals a fault which, if found later, may be many more times costly to correct so in the long run, is a good thing.

1.5.4 Meaning of completion or exit criteria

Completion or exit criteria are used to determine when testing (at any stage) is complete.

ISTQB Certification Preparation Guide: Chapter 1 – Principles of Testing

These criteria may be defined in terms of cost, time, faults found or coverage criteria.

1.5.5 Coverage criteria

Coverage criteria are defined in terms of items that are exercised by test suites, such as branches, user requirements, and most frequently used transactions etc.

1.6 THE PSYCHOLOGY OF TESTING

1.6.1 Purpose

The purpose of this section is to explore differences in perspective between tester and developer (buyer & builder) and explain some of the difficulties management and staff face when working together developing and testing computer software.

1.6.2 Different mindsets

We have already discussed that none of the primary purposes of testing is to find faults in software i.e., it can be perceived as a destructive process. The development process on the other hand is a naturally creative one and experience shows that staff that work in development have different mindsets to that of testers.

We would never argue that one group is intellectually superior to another, merely that they view systems development from another perspective. A developer is looking to build new and exciting software based on user's requirements and really wants it to work (first time if possible). He or she will work long hours and is usually highly motivated and very determined to do a good job.

A tester, however, is concerned that user really does get a system that does what they want, is reliable and doesn't do thing it shouldn't. He or she will also work long hours looking for faults in software but will often find the job frustrating as their destructive talents take their tool on the poor developers. At this point, there is often much friction between developer and tester. Developer wants to finish system but tester wants all faults in software fixed before their work is done.

In summary:

Developers:

- ü Are perceived as very creative - they write code without which there would be no system! .
- ü Are often highly valued within an organization.
- ü Are sent on relevant industry training courses to gain recognized qualifications.
- ü Are rarely good communicators (sorry guys)!
- ü Can often specialize in just one or two skills (e.g. VB, C++, JAVA, SQL).

Testers:

ISTQB Certification Preparation Guide: Chapter 1 – Principles of Testing

- Ü Are perceived as destructive - only happy when they are finding faults!
- Ü Are often not valued within the organization.
- Ü Usually do not have any industry recognized qualifications, until now
- Ü Usually require good communication skills, tact & diplomacy.
- Ü Normally need to be multi-talented (technical, testing, team skills).

1.6.3 Communication b/w developer and tester

It is vitally important that tester can explain and report fault to developer in professional manner to ensure fault gets fixed. Tester must not antagonize developer. Tact and diplomacy are essential, even if you've been up all night trying to test the wretched software.

1.6.4 How not to approach

Tester: "Hey Fred. Here's a fault report AR123. Look at this code. Who wrote this? Was it you? Why, you couldn't program your way out of a paper bag. We really want this fixed by 5 o'clock or else."

We were unable to print Fred's reply because of the language! Needless to say Fred did not fix the fault as requested.

Exercise

Your trainer will split you into small test teams. One of you will be the test team leader. You have found several faults in a program and the team leader must report these to the developer (your trainer). The background is that your team has tested this program twice before and there are still quite a lot of serious faults in the code. There are also several spelling mistakes and wrong colors on the screen layout. The test team is getting a bit fed up. However, you have to be as nice as possible to the developer.

1.6.6 Why can't we test our own work?

This seems to be a human problem in general not specifically related to software development. We find it difficult to spot errors in our own work products. Some of the reasons for this are:

- § We make assumptions
- § We are emotionally attached to the product (it's our baby and there's nothing wrong with it).
- § We are so familiar with the product we cannot easily see the obvious faults.
- § We're humans.
- § We see exactly what we want to see.
- § We have a vested interest in passing the product as ok and not finding faults.

Generally it is thought that objective independent testing is more effective. There are several levels of independence as follows:

- § Test cases are designed by the person(s) writing the software.

ISTQB Certification Preparation Guide: Chapter 1 – Principles of Testing

- § Test cases are designed by another person(s).
- § Test cases are designed by a person(s) from a different section.
- § Test cases are designed by a person(s) from a different organization.
- § Test cases are not chosen by a person.

The discussion of independence test groups and outsourcing is left to another section.

1.7 RE-TESTING AND REGRESSION TESTING

We find and report a fault in LOG 3, which is duly fixed by the developer and included in the latest release which we now have available for testing. What should we do now?

Examples of regression tests not carried out include:

- ü The day the phones stopped. .
- ü LAS failure on 4th November (perhaps)
- ü Ariane 5 failure.

Whenever a fault is detected and fixed then the software should be re-tested to ensure that the original fault has been successfully removed. You should also consider testing for similar and related faults. This is made easier if your tests are designed to be repeatable, whether they are manual or automated.

Regression testing attempts to verify that modifications have not caused unintended adverse side effects in the unchanged software (regression faults) and that the modified system still meets requirements. It is performed whenever the software, or its environment, is changed.

Most companies will build up a regression test suite or regression test pack over time and will add new tests, delete unwanted test and maintain tests as the system evolves. When a major software modification is made then the entire regression pack is likely to be run (albeit with some modification). For minor planned changes or emergency fixes then during the test planning phase the test manager must be selective and identify how many of the regression tests should be attempted. In order to react quickly to an emergency fix the test manager may create a subset of regression test pack for immediate execution in such situations.

Regression tests are often good candidates for automation provided you have designed and developed automated scripts properly (see automation section).

In order to have an effective regression test suite then good configuration management of your test assets is desirable if not essential. You must have version *control* of your *test Documentation* (test plans, scripts etc.) as well as your test data and baseline databases. An inventory of your test environment (hardware configuration, operating system version etc.) is also necessary.

1.8 EXPECTED RESULTS

The specification of expected results in advance of test execution is perhaps one of the most fundamental principles of testing computer software. If this step is omitted then human

ISTQB Certification Preparation Guide: Chapter 1 – Principles of Testing

subconscious desire for tests to pass will be overwhelming and tester may perhaps interpret a plausible, yet erroneous result, as correct outcome. .

As you will see when designing test using black box and white box techniques there is ample room within the test specification to write down you expected results and therefore no real excuse for not doing it. If you are unable to determine expected results for a particular test that you had in mind then it its not a good test as you will not be able to (a) determine whether it has passed or not and (b) you will never be able to repeat it.

Even with a quick and dirty ad-hoc test it is advisable to write down beforehand what you expect to happen. This may all sound pretty obvious but many test efforts have floundered by ignoring this basic principle.

" The major difference between a thing that might go wrong and a thing that cannot possibly go wrong is that when a thing that cannot possibly go wrong does go wrong it usually turns out to be impossible to get at or repair."

--*Douglas Adams*

SOME TESTING TERMINOLOGY

Faults - a mistake in the code that causes the software to not behave as expected (*causes*)

Failures - the act of a product not behaving as expected - the manifestation of a fault (*symptoms*)

Validation - establishing the correspondence between the software and its specification - "*are we building the right product?*"

Test care - the collection of inputs, predicted results and execution conditions for a single test

Ad-lib/ad-hoc test care - a test executed without prior planning - especially if the expected behaviors is not known prior to running the test.

Pass/fail criteria - decision rules used to determine whether a product passes or fails a given test

Coincidental correctness - when behavior appears to be what is expected, but it is just a coincidence

Test suite - a collection of test cases necessary to "adequately" test a product

Test plan - a document describing the scope, approach, resources and schedule of intended testing activity - identifies features to be tested, the testing tasks, who will do each task, and any risks requiring contingency planning.

Oracle - a procedure, process or magical phenomenon that can determine if the actual

ISTQB Certification Preparation Guide: Chapter 1 – Principles of Testing

behavior matches the expected behavior

Incident - when a test produces an unexpected outcome, - further effort is necessary to classify the incident as a software error, a design error, a specification error, a testing error, etc.

Bug report - a method of transmitting the occurrence of a discrepancy between actual and expected output to someone who cares for "follow-up" - also known as discrepancy report, defect report, problem report, etc.

Work-around - a procedure by which an error in the product can be "by-passed" and the desired function achieved.

Why Write Programs?

Concept: If you want your computer to do exactly what you want it to do, you must write a program.

A computer does nothing on its own. In fact, a computer is a dumb machine with no intelligence whatsoever. Despite what you might read in science fiction stories, a computer does nothing more than blindly follow instructions supplied by a programmer. Computers cannot think.

Definition: A program is a set of instructions that tells the computer exactly what to do.

When someone buys a computer today, the computer sits on the desk doing nothing until he loads a program into the computer's internal memory and starts running the program. Just as a VCR does not record shows on its own without being programmed to do so, a computer requires detailed instructions found only in programs.

Suppose that you own rental properties and want to use your computer to track your tenant records. Your computer will not help you out in any way until you load and run a rental property program. Where do you find such a program? There are two ways to obtain programs for computers. You can

(1) Buy one and hope that the program does exactly what you want it to do.

(2) Write your own program.

It's much easier and faster to buy a program that you need. Thousands of programs are on the market today. In fact, there are so many programs out there that you might not see the need for writing your own programs.

If you can find a program that does exactly what you want, you are ahead of the game. If you find a program that meets your exact requirements, you should buy that program because purchasing a program is often less expensive and much quicker than writing the same program yourself or hiring programmers to write it for you.

ISTQB Certification Preparation Guide: Chapter 1 – Principles of Testing

Think about this for a moment, though: If there are so many programs sold today that do virtually everything, why are programming languages such as Visual Basic continuing to break previous sales records each year? The answer is simple: People buy a computer so that the computer will do jobs that they need done. Firms cannot adapt their business to a computer program. They must find programs, or write their own programs, so that the computer processes information according to the business procedures already in place. The only way to ensure that a program exactly fits the needs of a firm is for the firm to develop its own programs.

Business people are not the only ones who need custom-designed programs. No two people manage their finances exactly the same way; no two scientists need computers for exactly the same kinds of computations; and no two graphic artists need the same kinds of computer drawing tools. Although people buy spreadsheets and word processors for their general-purpose computing needs, many people require specialized programs for specific jobs.

The art of programming computers is rewarding not only from a requirements standpoint, but also on a more personal level. Programming computers is fun! Just as a sculptor looks on a finished work of clay, programmers are often proud of the programs that they write. By the time you finish this book, you will have written programs that were not available before you wrote them. When you want your computer to do something specific and you cannot find a program that does the job exactly the way you want, you will be able to design and write the program yourself.

Some Programs are Changeable: There is a third method for getting exactly the program that you need if you want to computerize your company's accounting records. Accounting software firms often sell not only accounting programs but also the *source code* for those programs. The source code is a listing of the program's instructions. By having access to the source code, you can take what the software company wrote and modify the behavior of the program to suit your own requirements.

By starting with a functional program instead of starting from scratch, you save programming time and money. Sadly, most non-accounting software firms do not supply the source code. Most programs sold today have been compiled. After compiling, the source code is translated into a locked-in executable program. The bottom line is that you cannot easily change the behavior of compiled programs. For most programs, therefore, you have the choice of buying them or writing them yourself from scratch.

Definition: Code is another name for program.

Review: No single program pleases everyone. When a company sells a program, it must be general enough to please most purchasers. Some people need programs to behave in a specific manner in order to fulfill a specific need. They must resort to writing their own programs. Luckily, Visual Basic takes a lot of the pain out of writing programs.

A Brief History of Textual Programming

ISTQB Certification Preparation Guide: Chapter 1 – Principles of Testing

Concept: Computers cannot understand just any language. You must learn a language that your computer knows before you can write programs for your computer.

Definition: An application is yet another name for program.

Many people use computers all day long for word processing, database storage, and spreadsheet analysis, without realizing what is going on behind the scenes. You must always keep in mind that computers cannot think. Your computer does not know how to be a word processor. If you want your computer to do word processing, you must supply detailed instructions in the form of a program. Only by following the detailed instructions of a word processor program that you load can your computer perform word processing.

It would be nice if writing a program is as easy as telling the computer what you want done. Many people can handle ambiguous instructions, but computers are not smart enough to understand vague requirements. Computers can only follow orders given to them, and you must supply those orders in the form of a program. Therefore, you must supply the programs that you write. Writing programs, especially complex programs, takes time and several procedural steps. Visual Basic speeds the process of creating programs, but even with Visual Basic some programs take time to write and perfect.

Definition: A bug is a program error.

These are the typical steps that most programmers go through when writing programs. First, you have an idea for a program. Next, you use a program-development system, such as Visual Basic, to write the program. Errors, or bugs, often appear in programs because of the details needed for even the simplest of programs. Therefore, you must test the program thoroughly and fix the errors. Fixing errors is called debugging. Once all the errors are out of the program, you have a finished application.

PROGRAMMING

- A computer is an information-processing machine. [Executes given commands.
- Uses memory.
- Information = data. There are various types of data e.g. numerical, text, graphics or pictures, sound signals etc.
- A computer program is a sequence or set of instructions in a programming language.
- All data and instructions for a program are stored in the computer's memory in coded form. This coded form is similar to the morse code (-0---0-00-). The coding or translation is now done automatically. I.e. by commercial or computer programming.
- Programs will be written in the programming languages of which there are many. Like Pascal, Cobalt, (used for records), Visual Basic, etc.
- For the purpose of this module, Visual Basic (VB) programming language will be used. Hence the VB Integrated Development Environment (IDE) will do the translation into coded form, which is a software program.

LOGGING INTO VISUAL BASIC

ISTQB Certification Preparation Guide: Chapter 1 – Principles of Testing

VBS (adds topics), enter Open, enter

When in the form use - *Label 1. Caption = "Hello Class of 2000"*

To delete an object from the form Select Edit menu.

ERRORS IN PROGRAMMING

There are three common errors

1 Syntax errors: are due to structure or grammar of the language (rules) applied, e.g. more or less spacing, full stops, commas etc.

2 Routine errors: are due to non-existing situations like 1 divided by 0, is impossibility. These are errors where the program has instructed the computer to perform an impossible operation e.g. as shown above.

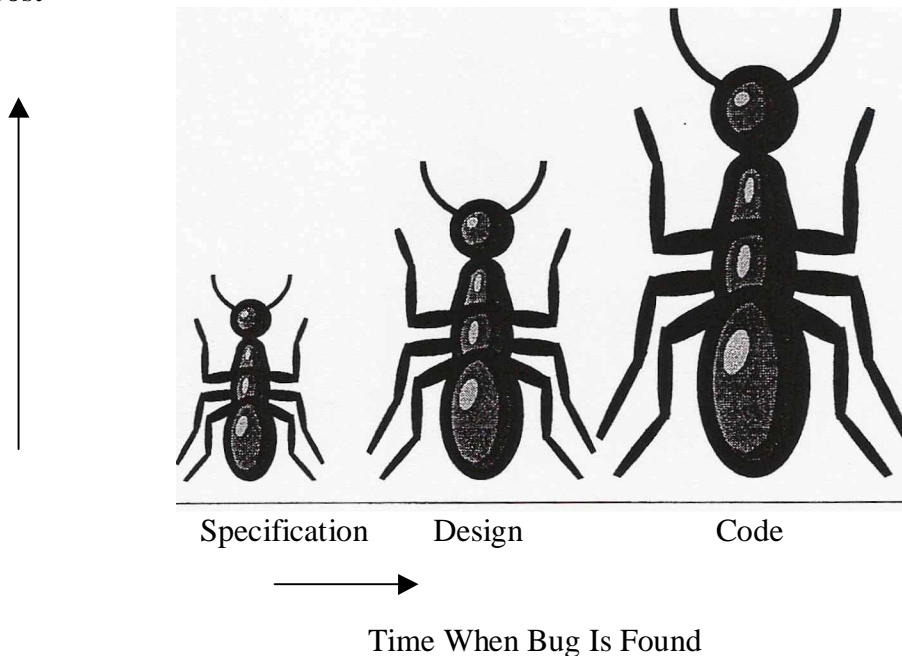
3 Logical errors: are those in the meaning of the program.

To save, always **SAVE FORM AS**, followed by **SAVE PROJECT AS**.

The Cost of Bugs

Programming, and testing, to its use by the public, there's the potential for bugs to be found. The Figure below shows how the cost of fixing these grows over time.

Cost



The cost to fix bugs increased dramatically over time.

The cost are logarithmic - that is, they increase tenfold as time increases. A bug found and fixed during the early stages when the specification is being written might cost next to nothing or 10 pence in our example. The same bug, if not found until the software is coded and tested, might cost £1 to £10. If a customer finds it, the cost could easily top £100.

ISTQB Certification Preparation Guide: Chapter 1 – Principles of Testing

WHY DOES SOFTWARE HAVE BUGS?

- Miscommunication or no communication - as to specifics of what an application should or shouldn't do (the application's requirements).
- Software complexity - the complexity of current software applications can be difficult to comprehend for anyone without experience in modern-day software development. Windows-type interfaces, client-server and distributed applications, data communications, enormous relational databases, and sheer size of applications have all contributed to the exponential growth in software/system complexity. And the use of object-oriented techniques can complicate instead of simplify a project unless it is well engineered.
- Programming errors - programmers, like anyone else, can make mistakes.
- Changing requirements - the customer may not understand the effects of changes, or may understand and request them anyway - redesign, rescheduling of engineers, effects on other projects, work already completed that may have to be redone or thrown out, hardware requirements that may be affected, etc. If there are many minor changes or any major changes, known and unknown dependencies among parts of the project are likely to interact and cause problems, and the complexity of keeping track of changes may result in errors. Enthusiasm of engineering staff may be affected. In some fast-changing business environments, continuously modified requirements may be a fact of life. In this case, management must understand the resulting risks, and QA and test engineers must adapt and plan for continuous extensive testing to keep the inevitable bugs from running out of control
- Time pressures - scheduling of software projects is difficult at best, often requiring a lot of guesswork. When deadlines loom and the crunch comes, mistakes will be made.
- Egos - people prefer to say things like: 'no problem', 'piece of cake', 'I can whip that out in a few hours' 'it should be easy to update that old code'
Instead of: 'that adds a lot of complexity and we could end up making a lot of mistakes' or 'we have no idea if we can do that; we'll wing it', 'I can't estimate how long it will take, until I take a close look at it', 'we can't figure out what that old spaghetti code did in the first place'

If there are too many unrealistic 'no problems', the result is bugs.

- Poorly documented code - it's tough to maintain and modify code that is badly written or poorly documented; the result is bugs. In many organizations management provides no incentive for programmers to document their code or write clear, understandable code. In fact, it's usually the opposite: they get points mostly for quickly turning out code, and there's job security if nobody else can understand it ('if it was hard to write, it should be hard to read').
- Software development tools - visual tools, class libraries, compilers, scripting tools,

ISTQB Certification Preparation Guide: Chapter 1 – Principles of Testing

etc. often introduce their own bugs or are poorly documented, resulting in added bugs.

Example - Microsoft Word

TEST NO	INPUTS	EXPECTED RESULTS
1	Click on File hold down and Drag to Open	Open dialog appears
2	Right click on paragraph	Menu options appear - font, paragraph etc. Cut Copy Paste are grayed out
3	Right click within a table	12 menu options appear related to table Actions e.g. insert row, delete row
4	Spelling	Spelling checking starts and displays first Error but check grammar off (Depends on option)
5	File Print	Displays printer dialog
6	Click on print icon	Will attempt to print to default printer

Driving License Exercise

Consider the following simple program:

```
IF age > 16 and age <= 65 THEN
    Issue drivers License
    IF age < 26 THEN
        Set Insurance Premium to HIGH
    ELSE
        Set Insurance Premium to STANDARD END IF
ELSE
    Error message DLOO1 -" Sorry, unable to issue drivers license"
END IF
```

TEST NO	INPUTS	EXPECTED RESULTS
1	10	
2	16	
3	17	
4	25	
5	26	
6	65	
7	70	

ATM Exercise

Some simplified rules for an ATM:

- a) Card must be valid for this bank
- b) Correct PIN must be entered
- c) Withdrawal must not take account overdrawn unless there is an overdraft agreed
- d) Notes available are £ 1 0 and £20 only

TEST NO	INPUTS	EXPECTED RESULTS
1	Valid Card Incorrect PIN £50 requested	'''
2	Valid Card Correct PIN £200 requested Balance £50 No overdraft	'i-
3	Valid Card Correct PIN £50 requested Balance £50 No overdraft	/
4	Valid Card Correct PIN £50 requested Balance £400 Withdrawn £160 earlier	1-

Telephones Exercise

TEST NO	INPUTS	EXPECTED RESULTS
1	Pick UD receiver	
2	Dial 100	
3	Dial 1142 or 192	
4	Dial 999	
5	Dial Busy Number	
6	Press 5 After Hearing Engaged tone	
7	Dial 11471	
8	Dial 11571	

--	--	--

1. 9 Prioritization of Tests

Since we have discussed already that there is never enough time to do all of the testing that you would like an obvious approach is to prioritize the tests This will then mean that whenever you stop testing you will have done the best testing possible in the time available.

The test manager should identify the criteria to be used when prioritizing tests. These will include but are not limited to:

- Ø Severity.
- Ø Probability.
- Ø Visibility of failure.
- Ø Business priority assigned to the requirement.
- Ø Customer wants.
- Ø How much change the function has undergone.
- Ø How error prone is this module.
- Ø How critical is it to the business.
- Ø How technically complex is it.
- Ø How difficult is it to test.
- Ø How costly is it to test.

Keep the test priority with the test at all times. This will prevent you from developing and designing low priority tests that never get run. Use a scheme of high, medium, low or a numeric scheme or 1,2, 3, 4 but try not to have more than about 4 categories.

NO	TEST CASE DESCRIPTION	PRI
1	A manual block on a bank is introduced (using the HOLD function) Causing payments to be moved between priority classes by the Scheduler	
2	Failure of the primary scheduler process	
3	Internal authentication failures	
4	Normal close of day	
5	Normal system start up	
6	One of each type of request to the scheduler	
7	Payments are released correctly from the USER class when limits Exceeded but funds received from another bank causing position to Improve	
8	Payments can be routed to the scheduler queue from several sources	

	with the correct class and user defined priority (including payments that have been referred or repaired)	
9	Payments cancelled manually be central control	
10	Regression testing of existing telecommunications links	
11	Regression testing of recompiled interface with mainframe system	
12	Remote bank changes their limit on us causing payments to be Released from the USER class (if held due to exceeding limit)	
13	Remote bank issues temporary raise	
14	Route all payments to the scheduler with parameters set for immediate release	
15	Special processing when internal cut-off time reached	
16	Suspend/activate limits	
17	Suspend/activate scheduler	

1.10 Summary

You should now be familiar with some of the fundamental principles of testing and you will recognize much of the terminology. The horror stories that you will have heard are not just sensationalist to motivate the course; they are true stories and there are (unfortunately) many more stories about the costs of not testing systems properly.

In particular you can now:

- Ø Recognize and use basic testing terminology.
- Ø Understand why testing is necessary.
- Ø Define error, fault and failure.
- Ø Explain why errors occur.
- Ø Give examples of the cost of errors.
- Ø Understand why exhaustive testing is unachievable.
- Ø Appreciate why testing is a risk management process.
- Ø Understand the fundamental test process.
- Ø Understand the difference between the mindsets of developers and testers.
- Ø Understand why you cannot test your own work.
- Ø Understand the need for regression testing.
- Ø Understand the importance of specifying your expected result in advance.
- Ø List some criteria for prioritizing your tests.