Chapter2: Testing throughout Lifecycle

"What is clear from the Inquiry Team's investigations is that neither the Computer Aided Dispatch (CAD) system itself, nor its users, were ready for full implementation on 26th October 1992. The CAD software was not complete, not properly tuned, and not fully tested. The resilience of the hardware under a full load had not been tested. The fall back option to the second file server had certainly not been tested."

Extract from the main conclusions of the official report into the failure of the London Ambulance Service's Computer Systems on October 26th and 27th 1992.

2.1 OVERVIEW

This module covers all of the different testing activities that take place throughout the project lifecycle. We introduce various models for testing, discuss the economics of testing and then describe in detail component, integration, system and acceptance testing. We conclude with a brief look at maintenance testing.

After completing this module you will:

2.2 OBJECTIVES

- Understand the difference between verification and validation testing activities
- Understand what benefits the V model offers over other models.
- Be aware of other models in order to compare and contrast.
- Understand the cost of fixing faults increases as you move the product towards live use.
- Understand what constitutes a master test plan.
- Understand the meaning of each testing stage.

2.3 Models for Testing

This section will discuss various models for testing. Definitions of these models will differ however the fundamental principles are agreed on by experts and practitioners alike. We cover verification and validation (V & V), the V-Model and briefly discuss a Rapid Application Development (RAD) approach to testing.

2.3.1 Verification and validation (V&V)

Verification is defined by B87925 as the process of evaluating a system or component to determine whether the products of the given development phase satisfy the conditions imposed at the start of that phase.

Validation is defined by B87925 as determination of the correctness of the products of software development with respect to the user needs and requirements.

Exercise

Validation and Verification

Complete the definition of testing: _____

Testing is defined by BS7925 as the process of exercising software to that it satisfies specified requirements and to_____

In simpler terms, verification answers the question "have we built the product right", i.e. is it correct and free from errors, does it meet the specification whereas Validation asks the question" is this the right product?" have the users got what they wanted. To help you remember this use the following:

Verification	Is it error free, does it do what was specified?
Validation	Is it valid, is this what you really, really want?

2.3.2 V-model

There are many models used to describe the sequence of activities that make a Systems Development Life Cycle (SDLC). SLDC is used to describe activities of both development and maintenance work. Three models are worth mentioning.

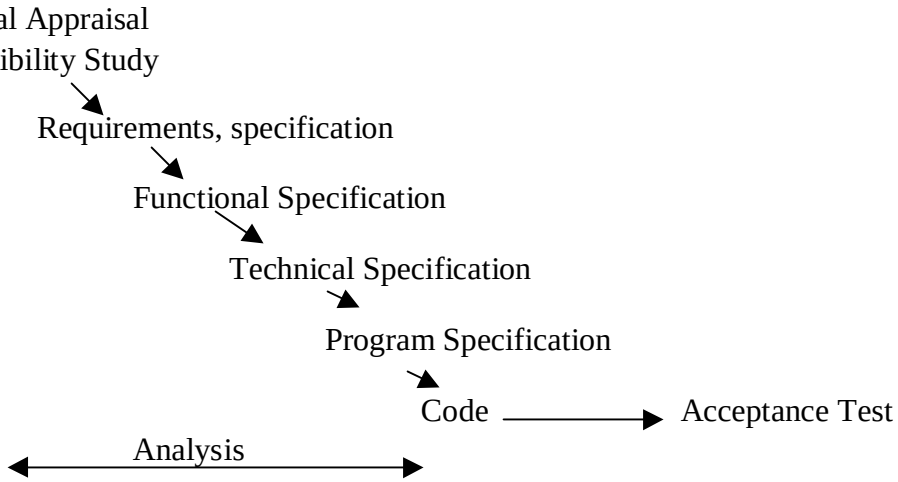
- **Sequential** (the traditional waterfall model).
- **Incremental** (the function by function incremental model).
- **Spiral** (the incremental, iterative, evolutionary, RAD, prototype model).

The three models would all benefit from earlier attention to the testing activity that has to be done at some time during the SDLC.

Any reasonable model for SDLC must allow for change and spiral approach allows for this with emphasis on slowly changing (evolving) design. We have to assume change is inevitable will have to design for change.

Fact	1. Business is always changing 2. Finding a fault causes change
Result	Ease of fixing a fault defines ease of responding to change
Corollary	If we want systems that can be modified and hence maintained, the earlier we start testing and try the change process, the earlier we will find out how easy it is going to be to maintain the system

2.3.3 Sequential Model

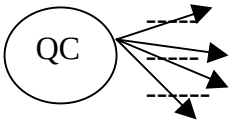


The sequential model often fails to bring satisfactory results because of the late attention to the testing activity. When earlier phases in the development cycle slip, it is the testing phase that gets squeezed. This can lead to a limited amount of testing being carried out with the associated production 'teething' problems.

2.3.4 Plan for wanted waterfall model development of system

The overall project plan for development of a system might be as shown below:

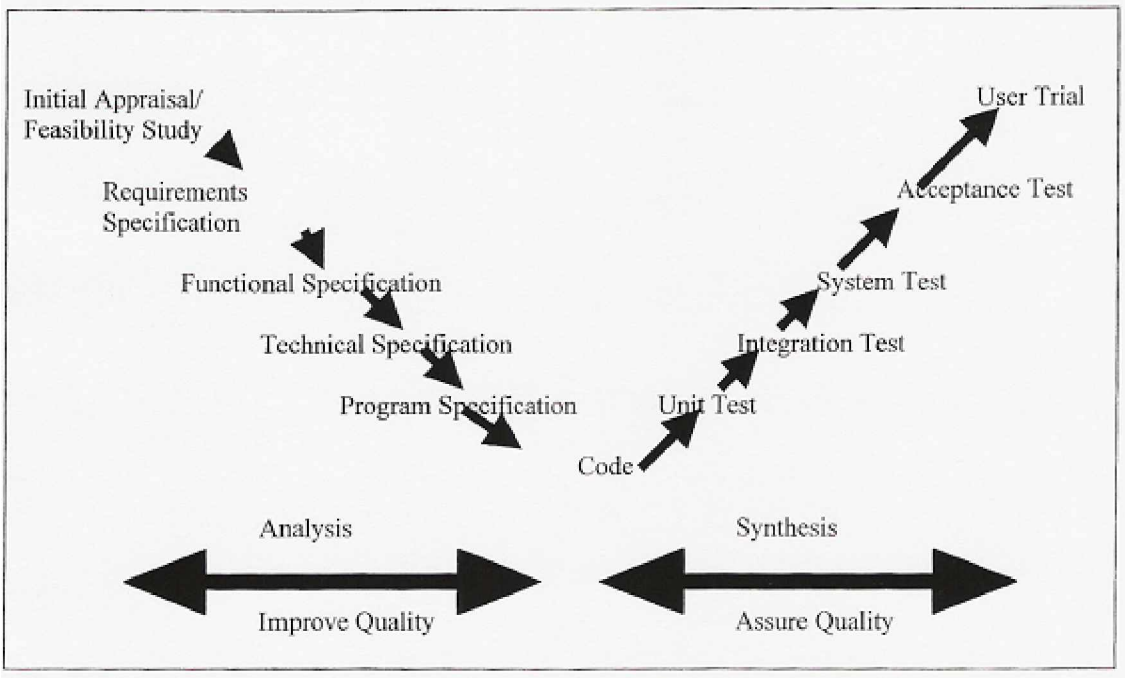
Activities	Time
Business study	----
Requirements analysis	----
User level design	----
Technical design	----
Program specification	----
Creation of code	----
Unit testing	----
Integration testing	-----
System testing	-----
Acceptance testing	-----
Implementation	-----



- This is a typical Specify, Design and Build project plan.
- All testing and quality control points come late in project and only done if there is time.
- When testing is done so late in project it can reveal costly errors.
- Project plan has testing done late because people think that only physical deliverables such as code can be tested. Clearly there has to be a better way.
- The challenge it to devise a better way of developing systems. There is a need to introduce quality control points earlier in the SDLC.

2.3.5 Sequential model plus testing gives 'V' diagram

The V diagram is another way of looking at the sequential development but this time from viewpoint of testing activities that need to be completed later in SDLC.



The 'V' diagram in this simple form has been around for a long time and is especially useful as it easily demonstrates how testing work done early in SDLC is used as input to assurance work later in development.

The V model of SDLC offers considerable benefits over others as it emphasizes building of test data and test scenarios during development and not as an after thought. The V model also allows for establishment of versions, incremental development and regression testing.

Management needs to rename activities, referred to variously as systems testing or acceptance testing. There has always been a phase of development traditionally thought of as the testing phase. This is an historical perception. Testing is not a phase but rather an activity that must be carried out all through development, giving rise to the principle of Total Quality.

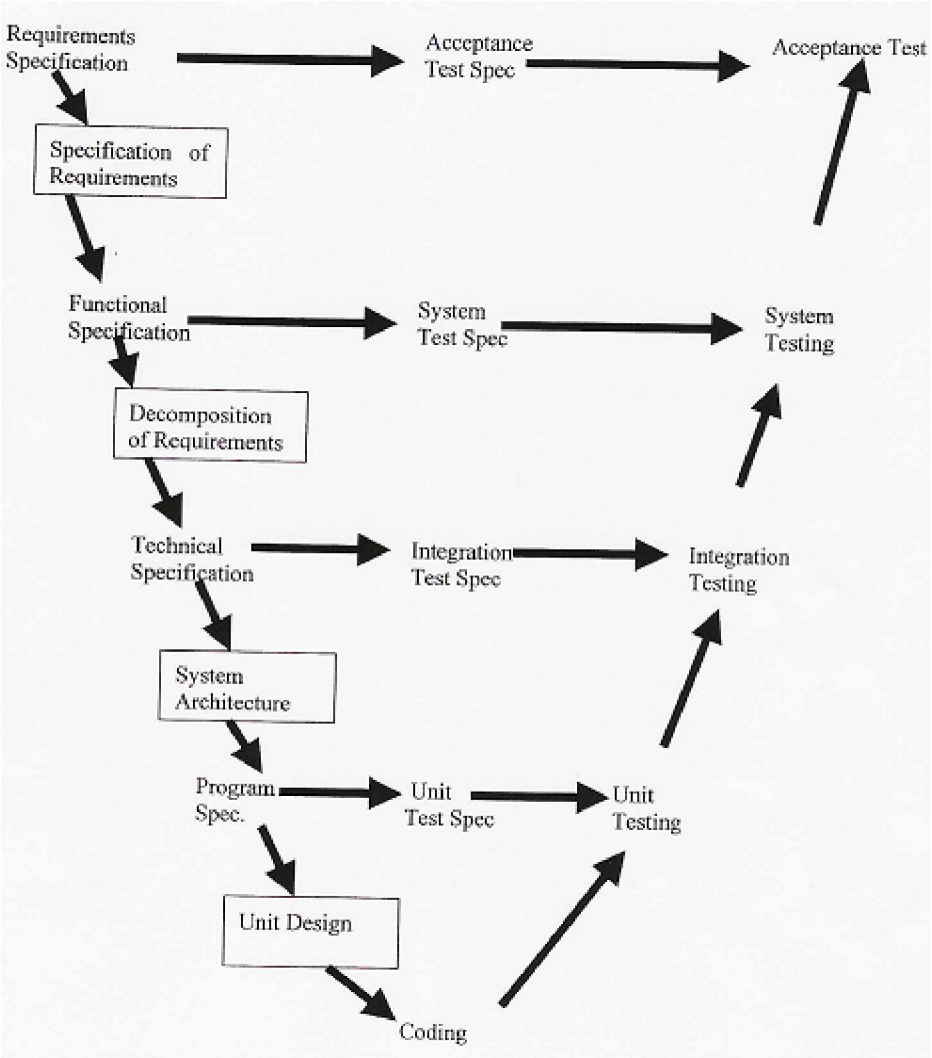
In the past, system testing was the only type of testing carried out. Testing was checking that programs, when linked together, met the systems specification. Whether design itself was correct was another matter. The concept of "testing" design before programs were coded was given only the most perfunctory attention. This was for two reasons:

1. By the time physical design had been done the system was very difficult to alter; modifications caused by design reviews were therefore very unwelcome.

2. The design was documented in terms of physical file layouts and program specifications, neither of which the user could comprehend. The question of whether physical design was correct could be reviewed, but the more important question: "Did the system do what the user wanted?" Was largely neglected.

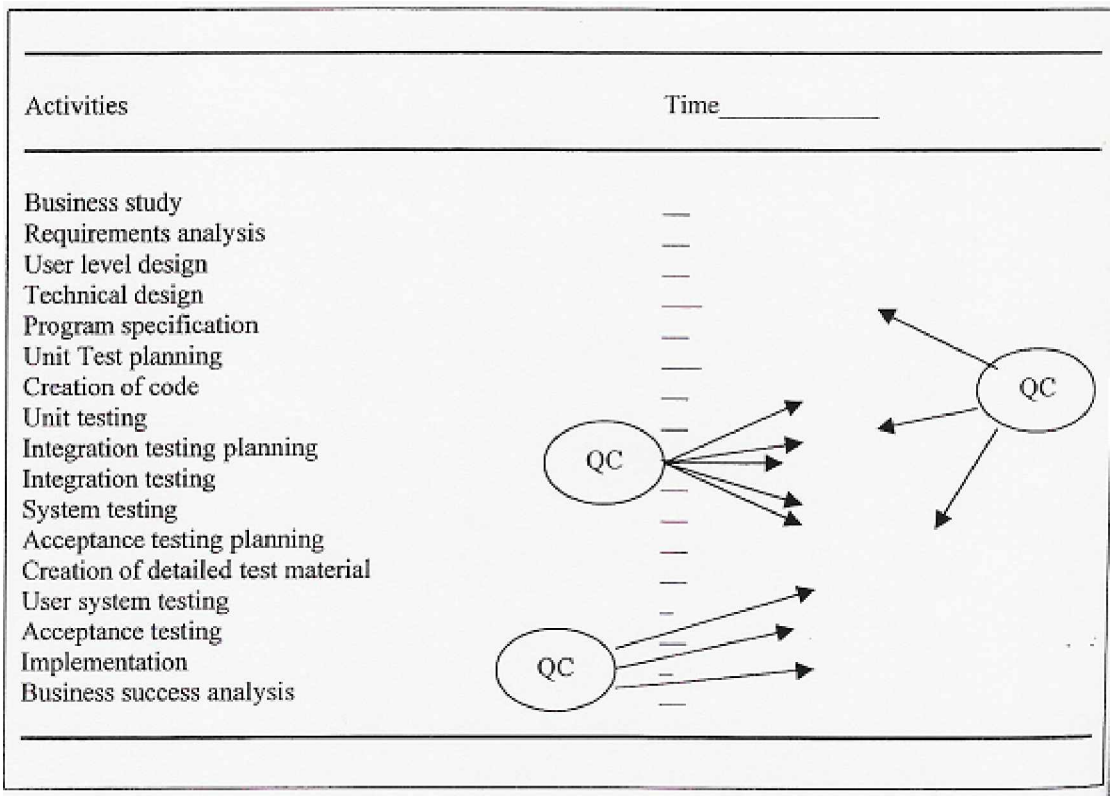
2.3.6 'V' with test recognized deliverable

The activity of Business Analysis has, as deliverables, the Specification of Requirements from which Acceptance Test Plan is constructed. To have created, for example, System Architecture without integration Test Specification is to do only *half the job!*



2.3.7 Revised plan for system development

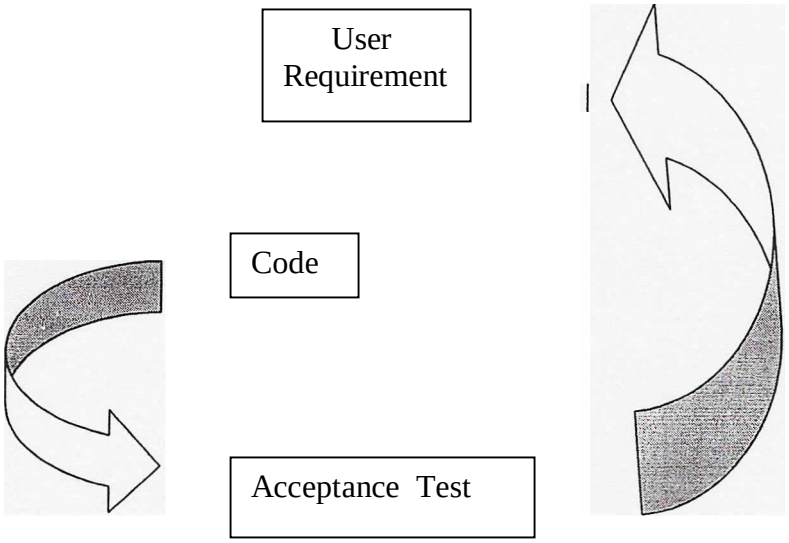
The overall project plan for development of a system might be as shown below. Note the new early quality control points.



This plan shows that the creation and running of actual tests are separated. The creation of test material (acceptance test plans, user system test scripts, technical system tests such as integration, link, recovery restart, etc., and unit test data) is done as the relevant design is done. The potential for automation is very good and the use of tools to capture the test cases, scripts, etc. will play a big part in making the running tests efficient. The early creation of test material will make the process of developing a system effective. The emphasis must be on first being **effective** then being **efficient.**

2.3.8 Rapid Application Development

The spiral, Rapid Application Development (RAD) model has the benefit of the evolutionary approach. This is an incremental process of build a little then test a little, which has the benefit of attempting to produce a usable but limited version early.



The RAD approach relies upon the quality of the RAD team.

The management issues to address are:

- Ø Have I got knowledgeable user input in my team?
- Ø Have I got experienced designers and developers in my team?
- Ø Am I leaving a good audit trail to support future Maintenance?

2.4 ECONOMICS OF TESTING

This section looks at some of the economic factors involved in test activities. Although some research has been done to put forward the ideas discussed, few organizations have yet to provide accurate figures to confirm these theories.

Major retailers and car manufacturers often issue product recall notices when they realize that there is a serious fault in one of their products. Perhaps you can think of other examples. The fixing of the so-called millennium but is probably one of the greatest product recall notices in history.

Boehm's research suggests that cost of fixing faults increases dramatically as we move software product towards field use. If fault is detected at an early stage of design, it may be only design documentation that has to change resulting in perhaps just a few hours work. However, as project progresses and other components are built based on faulty design, more work is obviously needed to correct fault once it has been found. This is because design work, coding and testing will have to be repeated for components of the system that were previously thought to have been completed.

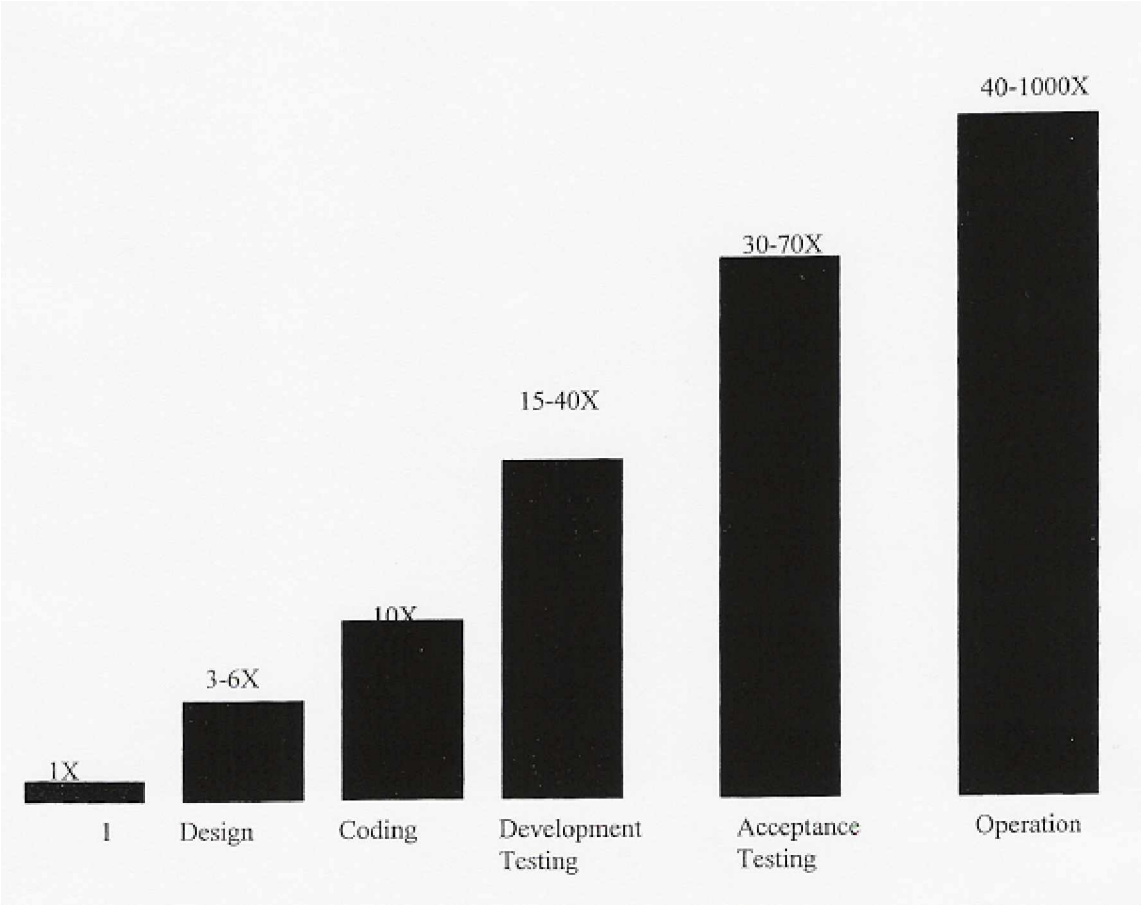
If faults are found in documentation, then development based on that documentation might generate many related faults, which multiply the effect of the original fault.

Analysis of specifications during test preparation (early test design) often brings faults in specifications to light. This will also prevent faults from multiplying i.e. if removed earlier they will not propagate into other design documents.

In summary we suggest that it is generally cost effective to use resources on testing throughout the project lifecycle starting as soon as possible. The alternative is to potentially incur much larger costs associated with the effort required to correct and re-test major faults. Remember that the amount of resources allocated to testing is a management decision based on an assessment of the associated risks. However, few organizations are able to accurately compare the relative costs of testing and the costs associated with re-work.

2.5 RELATIVE COSTS TO FIX ERROR

The cost of fixing errors escalates as we move the project towards field use. From an analysis of sixty-three projects cited in Boehm: Software Engineering Economics (Boehm, 1981).



2.6 High level test planning

You should be aware that many people use the term 'test plan' to describe a document detailing individual tests for a component of a system. We are introducing the concept of high level test plans to show that there are a lot more activities involved in effective testing than just writing test cases.

The IEEE standard for test documentation (IEEE/ ANSI, 1983 [Std 829-1983 D, affectionately known as 829, defines a master validation test plan as follows:

Purpose of master test plan is to prescribe scope, approach, resources and schedule of testing activities. A master test plan should include the following:

1. Test Plan Identifier
2. References
3. Introduction
4. Test Items
5. Software Risk Issues
6. Features to be tested
7. Features not to be tested
8. Approach
9. Item Pass/Fail Criteria
10. Suspension Criteria
11. Resumption Requirements
12. Test Deliverables
13. Remaining Testing Tasks
14. Environmental Needs
15. Staffing and Training Needs
16. Responsibilities
17. Schedule
18. Planning Risks and Contingencies
19. Approvals
20. Glossary

2.6.1 SPACE - the final frontier (of testing?)

This may be useful acronym to help you remember what you should include in any high level test plan document. The actual format is up to you and will depend upon your own company standards and other internal considerations. Space stands for Scope, People, Approach, Criteria and Environment and we have tried to map this onto the IEEE 829 headings as follows:

Scope	People	Approach	Criteria	Environment
I. Test plan identifier 2. References 3. Introduction	15. Staffing and training needs	8. Approach	9. Item pass/fail criteria	14. Environmental needs
4. Test Items 6. Features to be tested 7. Features not to be tested	16. Responsibilities	12. Test deliverables 13. Remaining test tasks	10. Suspension criteria	Test lab Test network
5. Software risk issues	17. Schedule	19. Approvals	11. Assumption requirements	Test data Use of live data
18. Planning risks and contingencies	Office requirements	Use of testing tools	Number of cycles es	
20. Glossary	Motivation/rewards	Configuration management		

Exercise

High Level Planning

We will use the case study for the London Ambulance Service's new computer aided Dispatch system to discuss the test planning issues. Take a few moments to think about how you would test such a system and we will complete the outline plan as a group exercise.

Scope

People

Approach

Criteria

Environment

2.7 Component testing

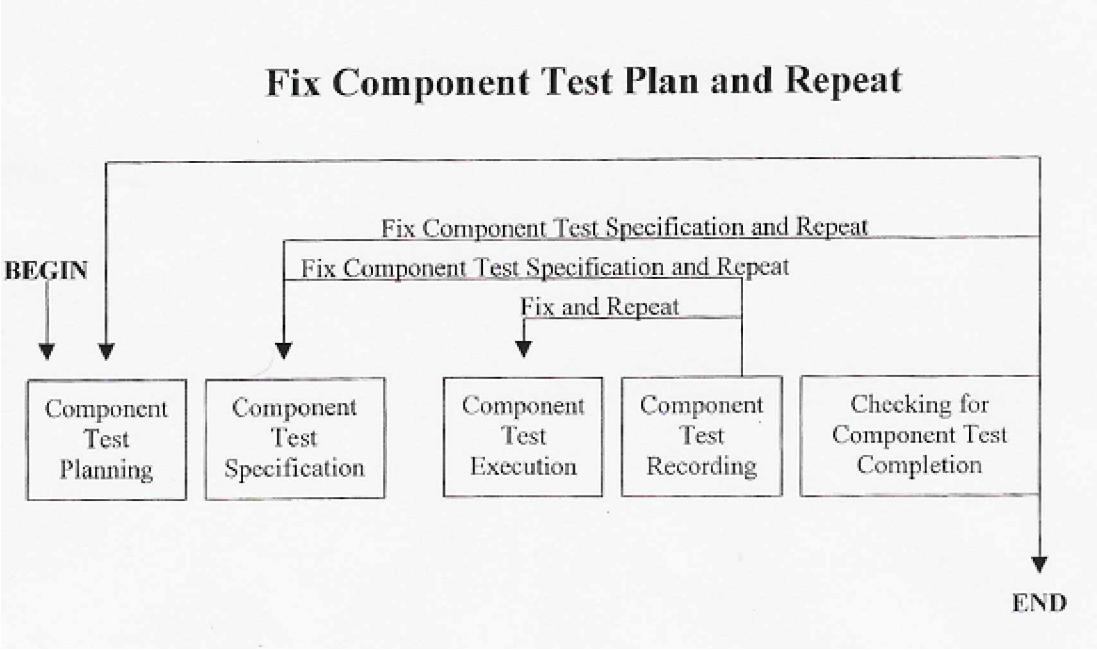
Component testing is described fully in BS-7925 and should be aware that component testing is also known as unit testing, module testing or Program Testing. The definition from BS7925 is simply **the testing of individual software components.**

> Traditionally, the programmer carries out component testing. This has proved to be less effective than if someone else designs and funs the tests for the component.

"Buddy" testing, where two developers test each other's work is more independent and often more effective. However, the **component test strategy** should describe what level of independence is applicable to a particular component.

Usually white box (structural) testing techniques are used to design test cases for component tests but some black box tests can be effective as well.

We have already covered a generic test process in this course. The component test process is shown in the following diagram:



2.8 INTEGRATION TESTING

Integration is the process of combining components into larger assemblies. From the standard BS-7925 **integration testing** is defined as "testing performed to expose faults in the interfaces and in the interaction between integrated components", However, in this section we look at two interpretations of integration testing known as integration testing in the large and integration testing in the small.

By Integration Testing in the Large we mean testing the integration of the new system or software package with other (complete) systems. This would include the identification of, and risk associated with, all interfaces to these other systems. Also included is testing of any interfaces to external organizations (e.g. EDI - electronic data interchange, Internet) but not testing of the processing or operation of those external systems.

Exercise

Compare the definition of integration testing in the large with B87925 (50115) definition of interface testing. Any comments?

We use Integration Testing in the Small in the more traditional sense of integration testing where components are assembled into sub-systems and subsystems are linked together to form complete systems. Integration strategies may be incremental or non-incremental and include:

- Ø bin-ban
- Ø Top-down
- Ø Bottom-up
- Ø Sandwich

Testing approach is directly related to integration strategy chosen.

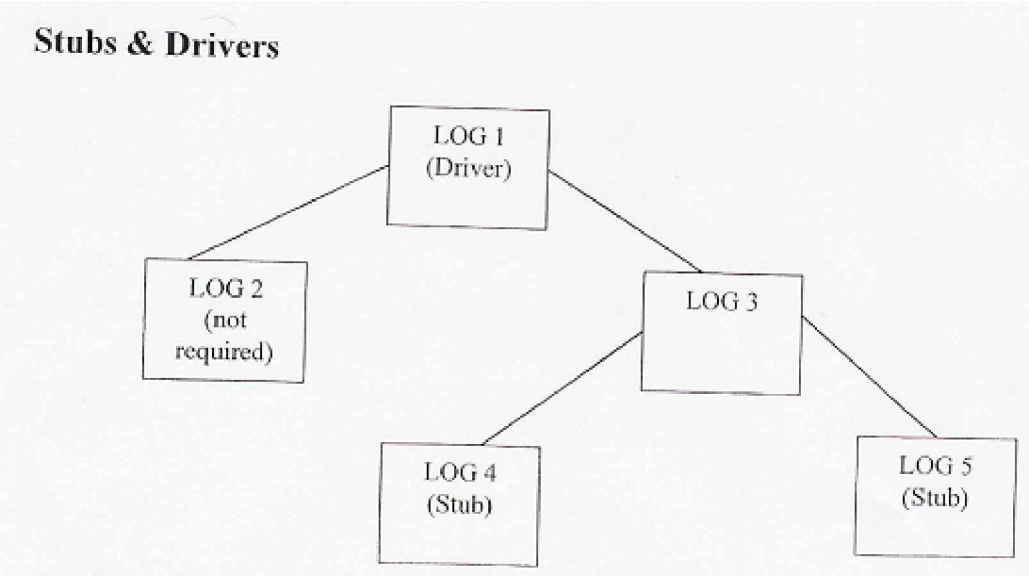
Stubs & Drivers

A stub is a skeletal or special purpose implementation of a software module, used to develop or test a component that calls or is otherwise dependent on it.

A test driver is a program or test tool used to execute software against a test case suite.

TRAINER'S NOTES:

Why not refer to the project EX example in BS7925; in the component test plan for the component LOG3 it has a good diagram showing stubs and drivers.



2.9 SYSTEM TESTING

System testing is defined as the process of testing an integrated system to verify that it meets specified requirements.

You will come across two very different types of system testing, Functional system testing and non-functional system testing. In plain English functional system testing is focuses on testing the system based on what it is supposed to do. Non-functional system testing looks at those aspects that are important yet not directly related to what functions the system performs. For example if the functional requirement is to issue an airline ticket, the non-functional requirement might be to issue it within 30 seconds.

A functional requirement is " a requirement that specifies a function that a system or system component must perform". Requirements-based testing means that the user requirements specification and the system requirements specification (as used for contracts) are used to derive test cases. Business process-based testing based on expected user profiles (e.g. scenarios, use cases).

Non-functional requirements cover the following areas:

1. Load
2. Performance
3. Stress
4. Security
5. Usability
6. Storage
7. Volume
8. Install ability
9. Documentation
10. Recovery

Non-functional requirements are just as important as functional requirement

Exercise

List some non-functional requirements for LAS CAD system.

2.10 ACCEPTANCE TESTING

The definition of acceptance testing in BS7925 states that *"acceptance testing* is formal testing conducted to enable a user, customer, or other authorized entity to determine whether to accept a system or component". Acceptance testing may be the only form of testing conducted by and visible to a customer when applied to a software package. The most common usage of the term relates to the user acceptance testing (VAT) but you should be aware that there are several other uses of acceptance testing, which we briefly describe here.

User acceptance testing - the final stage of validation. Customer should perform or be closely involved in this. Customers may choose to do any test they wish, normally based on their usual business processes. A common approach is to set up a model office where systems are tested in an environment as close to field use as is achievable.

Contract acceptance testing - a demonstration of the acceptance criteria, which would have been defined in the contract, being met.

Alpha & beta testing - In alpha and beta tests, when the software seems stable, people who represent your market use the product in the same way (s) that they would if they bought the finished version and give you their comments. Alpha tests are performed at the developer's site, while beta tests are performed at the user's sites.

Exercise

What problems would LAS have avoided had they set some acceptance criteria?

2.11 MAINTENANCE TESTING

Maintenance testing is not specifically defined in BS7925 but it is all about testing changes to the software after it has gone into productions.

There are several problems with maintenance testing. If you are testing old code the original specifications and design documentation may be poor or in some cases non-existent. When defining the scope of the maintenance testing it has to be judged in relation to the changed code; how much has changed, what areas does it really affect etc. This kind of impact analysis is difficult and so there is a higher risk when making changes - it is difficult to decide how much regression testing to do.