S T S A R C E S

Standards for Safety Related Complex Electronic Systems

Annex 4

Tools for Software fault avoidance

Task 2: Guide for the construction of software tests

Final Report of WP1.2

Philippe Charpentier

INRS

European Project STSARCES
Contract SMT 4CT97-2191

Contents

SUMMARY:

This document covers the tests of embedded software. It is a guide to assist the assessor in constructing his own technique, and in adapting it to the specific context both upstream of and during the development process. This is a set of basic techniques and reference points that help the assessor both to collect and analyse the information provided by the development process.

It has been drawn up from a pedagogical point of view and not from the contractual viewpoint of a supplier / assessor organisation relationship.

It completes the document: “Software quality and safety requirements” which defines the quality and safety requirements adapted to the machinery, and the document: “Guide to evaluating software quality and safety requirements” which is intended to assist the analyst in assessing the respect of these requirements.

I. INTRODUCTION

I.1. AIM OF THE DOCUMENT

This document completes the document: “Software quality and safety requirements” [1] which defines the quality and safety requirements adapted to the machinery, and the document: “Guide to evaluating software quality and safety requirements” [2] which is intended to assist the analyst in assessing the respect of these requirements.

It is coherent with these two documents and with the EN 954-1 Standard: “Safety of machinery - Safety-related parts of control systems; Part 1: General principles for design” [3].

It includes certain (but not all) of the elements of the IEC 61508 Standard “ Functional safety of E / E / PE safety-related systems ” [4] by adapting them to the products in question (cf. § 1.3).

I.2. SCOPE OF THE DOCUMENT

This document is intended for the manufacturer's test team.

The activities described in this document are to be undertaken by this test team. **Under no circumstances are they the responsibility of the certifying body** which may, nevertheless, require that complementary testing be undertaken at each phase. The manufacturer's test team will, henceforward, be termed "the assessor".

This document has been drawn up from a pedagogical point of view and not from the contractual viewpoint of a supplier / assessor organisation relationship.

I.3. PRODUCTS CONCERNED

This document only covers embedded software tests.

Highly critical systems (aeronautics, nuclear, etc.) are excluded from the scope of this document, as are applications software such as application programmes for PLCs[1].

The scope of documents [1] and [2] is limited to small-size software. It is extended in this document to software of all sizes.

I.4. INSTRUCTIONS FOR USING THE DOCUMENT

The assessor must have basic knowledge regarding software quality and/or software development.

It is a set of basic techniques and reference points that help the assessor both to collect and analyse the information provided by the development process. **This is not a verification checklist, but in reality a guide to assist the assessor in constructing his own technique,**

[1] PLC : Programmable Logic Controler.

and in adapting it to the specific context both upstream of and during the development process.

I.5. PRESENTATION OF THE DOCUMENT

In addition to this introductory chapter, which presents the general aspects of the software test in particular, this document comprises the following chapters:

- Software life cycles and verification support documents (chapter II)
- Validation test phase (chapter III)
- Integration test phase (chapter IV)
- Unit test phase (chapter V)
- as well as one appendix presenting references for testing strategies.

I.6. SOFTWARE TESTING OBJECTIVES

Software testing is a dynamic verification method to check that the software indeed has the characteristics required for its context of use. This leads to a preliminary remark that the context of use (expected functions, environmental constraints, adverse conditions) must be described and indeed correspond to the real world.

The first aim of the tests is to detect possible divergences between the expected behaviour and the behaviour encountered during the tests. The testing techniques allow detection of a great many of the faults present in software.

The second aim of the tests is to obtain the necessary confidence before operational use. Unfortunately, the number of detected faults cannot be employed to calculate the number of faults remaining to be discovered. Field experience has shown that, at a constant technical and industrial complexity, a high number of errors detected compared to other “ standard ” projects can only be interpreted as an indicator of a software containing a great many faults and not as the achievement of a high rate of detection of the faults present. It is therefore very difficult to obtain the necessary confidence in software when a large number of faults have been detected.

Let us also bear in mind the risk attached to the same team defining the specification, the design, the testing strategy, and the test cases. The aim of the tests is rather to demonstrate that it has worked in an unsatisfactory way, which is not a natural mental process. In addition, common mode failure exists; what has been badly designed risks being badly tested by the same person through making the same error of reasoning.

Furthermore, correcting software errors is always delicate:

- not injecting new errors during correction cannot be guaranteed: the correction may not be satisfactory or perturb the correct parts already tested.
- correction can activate an inaccessible part of the software and therefore cause a great many new faults not yet noticeable before this correction.

Reliability growth models must therefore be employed with the utmost caution in the domain of software.

Every software must be considered as a complex system whose behaviour cannot be guaranteed in the real domain of use. The evaluation of a software cannot be stated on the single observation of a software under test. This test assessment can be completed by:

- analysing the architecture of the software and the programme code,
- continuous assessment throughout the development process.

These investigations answer the question: has the software been well made?

Indeed, to cover the subject thoroughly at system level from the safety point of view, the following question must also be able to be answered: **does the software produced indeed correspond to the problem to be resolved?**

Only system-level analyses allow this question to be studied. This is beyond the scope of this study which is limited to software tests.

To conclude, **it must be emphasised that it is an illusion to qualify a complex system on the single observation of its behaviour in a restricted number of test sets**. Tests are essential to detect errors and to improve confidence in the ability of the system to accomplish its task, but are insufficient to “ prove ” the behaviour of a system.

I.7. THE TEST PROCESS

Three test phases are involved during software development. These are:

- Validation tests (VT);
- Integration tests (IT);
- Unit tests (UT).

The activities linked to these three types of test are identical. Moreover, the test process must follow the same principles used in all other creation, construction or design processes if quality is to be ensured. This requires:

- initial definition,
- execution according to the previously defined procedures,
- verification according to fixed modalities.

I.7.1. Initial definition of tests to be undertaken

The tests are prepared immediately after validation of the corresponding specification/ design phase.

Test definition requires the drawing up of Test Plans for validation, integration and unit testing along with the associated sheets or procedures.

The diagram below supplies the test definition phases for a "V" life cycle.

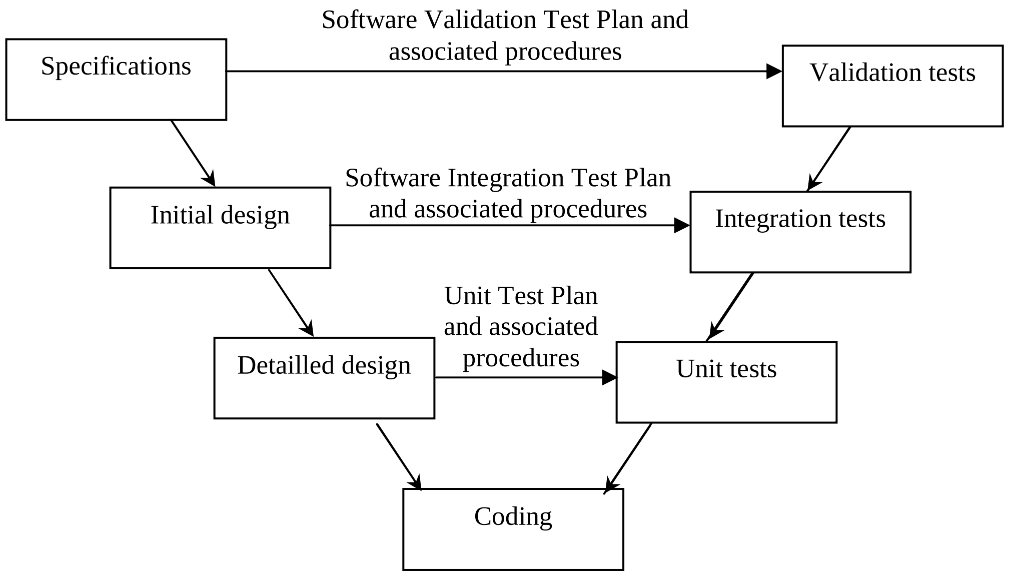
Software Validation Test Plan and associated procedures
Specifications
Validation tests
Software Integration Test Plan and associated procedures
Initial design
Integration tests
Unit Test Plan and associated procedures
Detailled design
Unit tests
Coding

Figure 1: Test Definition Phases for a "V" Development Cycle

Definition of the tests (unit, integration and validation) is undertaken in two stages:

1/ The first task to be undertaken in the preparation of the tests is to identify the **test objectives.**

This involves defining the objectives for each test phase, taking into account:

- dependability requirements, (robustness, maintainability, availability, safety, etc.),
- functional and performance requirements expressed in the software product definition/ design documents,
- all identified implementation constraints (use of a Data Base, use of real numbers, programming language, real-time software).

The **types of test** to be undertaken (functional paths, input / output data at limits , beyond limits , robustness, performance) and the justification of the choices made result from such definitions.

For each type of test defined, the **requirement concerning the test coverage rate** is 100% by default but this rate can be reduced under certain cases when there is sufficient justification.

Example

If the choice of testing functional paths is adopted, a coverage rate of 100% means that after tests have been undertaken, each of the functional paths of the software must have been tested at least once. It is, of course, necessary to be in possession of the means to be able to measure the real coverage rate by comparison with the objective coverage rate (here 100%).

<u>Case under which a reduction of the rate of coverage can be justified:</u>

The software environment represents a case under which the objective of coverage rate may be reduced. Indeed, during site validation tests, the software could be in an environment which might prevent testing of certain functions designed for a different environment. These functions are not, therefore, to be tested for this site and the coverage rate is not 100% of the functions described in the specifications, but all functions except those functions just mentioned above. The justification for this choice must nevertheless be set down in the test file.

Similarly, certain tests concerning real-time aspects cannot always be undertaken during unit testing. Appropriate justification will allow for the limiting of objective coverage of real-time test aspects during unit testing.

<u>Remark concerning the coverage rate for structural tests:</u>

The parameters necessary to measure the coverage rates are not independent and can lead to adverse effects. For example, to impose a coverage rate in branches or in function calls can lead the designer to degrade the code to reach the objectives more easily during the test phase: for example, duplicating the code (less sub-routines) or extending the average size of subroutines and procedures. Any objective approach to coverage rate must be accompanied by programming rules to prevent this mode of bypassing.

2/ The second task to be undertaken consists in defining t**he means** employed in testing and then in defining the linking together of test tasks which is to say the **planning procedure**.

The means and the planning procedure taken together constitute the **test strategy** adopted.

The definition of the tests leads to the drawing up of the **test sheets.**

The three following paragraphs present the aspects common to the three test phases (VT, IT, UT) concerning the definition of Test objectives, the test strategy and testing facilities.

I.7.1.1. Definition of test objectives

The effort put into testing software (VT, IT and UT) can be as much as is desired. The combinations of input data, utilisation scenarios, operating modes, operating parameters, the variety of verifiable aspects (conformity to specifications/ design, user and operating manuals, robustness requirements, performance, ergonomics, nominal behaviour, behaviour on failure, etc.) are such that an infinite number of tests can be carried out, without ever being absolutely certain of total conformity with the requirements.

In practice, the effort put into testing must therefore be adapted to what is at stake, and it is necessary to set objectives prior to giving thought to the tests.

Defining these objectives allows for maximum efficiency in undertaking tests whilst guaranteeing full coverage of the requirements defined and hence:

- avoids devoting useless effort to pointless activities,

- focuses on essential aspects.

I.7.1.2. Testing strategy

Whatever the requirement level, the test must be designed and developed on the basis of a strategy. This strategy reflects the consideration given to the tests by the assessor. It is necessary prior to developing tests and designing test cases. Its absence indicates a risk of improvisation in the development of the tests.

The strategy must define what the assessor is doing globally to ensure that the tests satisfy the previously defined objectives. It must organise the tests into a clear structure of units or stages, each with clear objectives or responding to precise environmental constraints (for example, availability of a specific tool).

I.7.1.3. Testing methods and techniques : basic techniques

This chapter concerns testing strategy. The basic techniques are divided into two categories, depending on whether or not they authorise the observation of the internal behaviour of the tested component:

- Black box: no visibility of the tested component. The input data and the expected results are expressed in terms of the behaviour of the system from an external point of view.
- White box: visibility of the tested component.

I.7.1.4. Testing facilities

Whatever the level of requirement, the definition of the testing strategy and methods must be accompanied by the definition of the facilities necessary to apply them. Absence of this information greatly reduces the credibility of the strategy and methods envisaged. It indicates a risk of improvisation in the development and conducting of tests.

As an example, the list below provides the facilities that should be considered in advance:

- Tools: simulators, comparators, signal measurement tools, testing tools, etc.
- Testing environments: different configurations of system, in factory or on site, etc.
- Pre-recorded sets of tests, pre-defined parameter tables, etc.

In so far as unit tests in particular are concerned, establishing strategy **imposes** a use of tools that is related to the size of the software. These tools allow both automation of redoing, test processing and measurement of the coverage rates.

I.7.2. Test execution

Test execution must conform to the procedures and to the conditions previously defined in the test plans and test files (preparation activities, procedures used during testing, recording of anomalies and test termination criteria activities).

Any failure to respect these constraints must be well-argued. This will enable justification of these divergences during test verification.

Test stop criteria reflect the policy adopted with regard to the pursuit of anomaly detection tests. Any event which prevents the correct follow-through of the testing procedure must be considered in the elaboration of the criteria. This might be the detection of a bug or a major non-conformity, for instance. The detection of an event representing a test stop criterion will lead to stopping the group of tests currently being undertaken before its has reached its end and moving on to the following group of tests. The group of tests which has been stopped will have to be undertaken anew after correction.

Test stop criteria are to be defined beforehand in order to avoid undertaking pointless tests which will have to be undertaken anew in any case.

I.7.3. Test verification

Test verification must demonstrate that the objectives set have been met. It consists of:

- establishing the traceability of tests undertaken for each type of test defined given the objectives in hand. It is strongly recommended that traceability be established as tests execution in order to optimise the test verification phase for which such traceability is necessary,
- evaluating the coverage rate attained through the analysis of test results,
- establishing the acceptance or refusal of results in the phase in question by taking into account the justifications for divergence established during test execution.

The table below gives the format of a traceability table:

Elementary objective to be verified	**type of test**			
	TN	**HL**	**TR**	**…**
Objective 1	1,2		4,5	
Objective 2	6	3		
Objective 3	7,8	9		
Objective 4	…			
Objective …				
Objective n	m	m+1		

Table 1: Objective/ Type of test/ tests number traceability Table

TN : Nominal Test, HL : Outside limit ,TR : Robustness Test …

II. SOFTWARE LIFE CYCLES AND VERIFICATION SUPPORT DOCUMENTS

II.1. REFERENCE MODEL

This guide must have recourse to a model of the software development process that is as close as possible to the process normally employed by the designers. The “ V ” cycle mode, which serves as the basis of the definition of the requirements [1] and their assessment [2], has been employed again in the present guide to ensure coherence with the preceding documents and on account of its simplicity.

II.2. SUPPORT DOCUMENTS AND THEORETICAL DELIVERABLES

The verification technique defined in the present document is based on analysing the intermediate results of the development process of the designer.

For practical reasons, this document defines the basic verification to be conducted on each of the intermediate results of the process, in the order that they are normally delivered.

A deliverable can concern product-definition aspects (e.g. specification, architecture, source code), verification aspects (e.g. review reports, test plans, test reports, etc.) or support aspects (state of configuration management, project monitoring state).

The assessor will pay particular attention to verifying the presence of useful information and its relevance rather than its distribution in a “ theoretical ” documentary tree.

“ Theoretical ” deliverable	Contents
Validation planning	Objectives, strategy, techniques and methods, organisation and responsibilities, necessary facilities, identification of tests and demonstration of coverage for the validation tests.
Validation procedures	Detailed description of each validation test in terms of operational mode, input data, expected results, and stopping criteria.
Integration planning	Objectives, strategy, techniques and methods, organisation and responsibilities, necessary facilities, identification of tests and demonstration of coverage for the integration tests.
Integration procedures	Detailed description of each integration test in terms of operational mode, input data, expected results, and stopping criteria.
Unit test planning	Objectives, strategy, techniques and methods, organisation and responsibilities, necessary facilities, identification of tests and demonstration of coverage for the unit tests.
Unit test procedures	Detailed description of each unit test in terms of operational mode, input data, expected results and stopping criteria.
Unit test report	Conditions of carrying out the unit tests, elements tested, tests carried out, results obtained, interpretation of tests and report.
Integration report	Conditions of carrying out the integration tests, elements tested, tests carried out, results obtained, interpretation of tests and report.
Validation report	Conditions of carrying out the validation tests, elements tested, tests carried out, results obtained, interpretation of tests and report.

Table 2: Definition of the “theoretical” deliverables

II.2.1. Using the "theoretical deliverables"

A certain number of elements are necessary in order to undertake each test phase. These elements are to be found in the table "Input elements of the phase" situated at the beginning of the chapters dealing with each of the three test phases.

Use by the developer.

The table "Input elements of the Phase" allows the designer to ensure that all the documentation expected by the assessor is indeed available

The table can be used in the reviews preceding the production of the corresponding “theoretical deliverables ” to define, in conjunction with the designer and the assessor, the elements expected at the test phases.

Use by the assessor.

The assessor begins by checking the availability of the elements listed in the table before beginning the test phase.

For each element of the table, the assessor looks in the documentation available for the information corresponding to it. To do this:

- he notes the references of the designer’s documents in which the element is found and the location it is found (for example paragraph or page number).
- he indicates if the element cannot be found (completely or partially).

He then requests the missing elements from the designer.

It should be noted that the existence of documentation does not prejudge the fullness of the paragraphs designated to cover an entry of the test phase or the existence of any possible incoherence.

III. VALIDATION TEST PHASE

III.1. PHASE RESPONSIBILITIES

Those responsible for the validation test phase must be designated following the requirement 3.9 laid down in document [1]: “ … The verification strategy should be developed and implemented, and the test results should be evaluated independently (by an individual, department, or organisation) in relation to the size of the development team“. This requirement is not required for levels 1 and required for level 2.

Standard CEI 61508 [4] is more demanding and requires more independence. The table below gives the correspondence for levels 1 and 2 (cf. [2] for the definition of level).

Degree of independence required for the validation	**Level 1**	**Level 2**
Independent person	HR	HR
Independent department	-	HR
Independent organisation	-	-

Table 3: Degree of independence required for validation in relation to level

HR: Highly Recommended

III.2. INPUT ELEMENTS OF THE PHASE

The elements listed below are necessary to complete the phase correctly. The development team should supply them to the assessor.

Input elements of the phase
<u>Product definition documentation</u>
• Software specifications
• User documentation *All documents (manuals, procedures, etc.) defining the utilisation of the product.* *It is a useful addition but is not vital to understand the tests. It is a necessary element if the objective of the tests covers the conformity with this documentation.*
• Operation, parameter-setting documentation, etc. *All documents (manuals, procedures, etc.) defining all possible activities on the product beyond normal utilisation.* *It is a useful addition but is not vital to understand the tests. It is a necessary element if the objective of the tests covers the conformity with this documentation.*
<u>Configuration and modification management documentation</u>
• Versions presented in validation *Precise identification of the product tested with its version and the version of the system in which it is included*
• Requests for modifications stemming from the tests.
• Impact analysis reports, decisions made and justifications concerning these requests.
• Software configuration history *List of modifications carried out between successively presented versions.*
• Organisation and responsibilities *Persons involved in the validation, roles and responsibilities.*

Table 4: Input elements of the validation test phase

III.3. PHASE ACTIVITIES

III.3.1. Validation test planning

The aspects common to the three test phases (VT, IT, UT) concerning the definition of test objectives, the testing strategy and the testing facilities are to be found in Chapter I.7.1

III.3.1.1. Definition of validation test objectives

A priori, in validation, the objective of covering each element of the specification, including the safety mechanisms, should be set.

The objectives should cover as a minimum (cf. requirement 3.13 in document [1]):

- each specification element should be covered by a validation test, including safety mechanisms,
- it should be possible to verify the real-time behaviour of the software in any operational mode.

III.3.1.2. Validation testing strategy

As an example of a validation testing strategy, to show what a strategy can be (a specific strategy should be developed for each specific context and for a given context, there are n possible strategies), the list below provides examples of typical stages. These stages are linked to test type as well as to the constraints concerning the manner in which the tests are undertaken (test equipment, environment, etc.):

- a stage of detailed test of each function in nominal mode;
- a stage of detailed test of each function inside the limits;
- a stage of detailed test of each function outside the limits;
- a stage of performance tests ;
- a stage verifying the behaviour in the case of failure ;
- a stage dedicated to parameter-setting aspects ;
- stages with simulated inputs and stages with real inputs from the operational environment ;
- etc.

The strategy must be integrated into the overall product verification strategy. For example, if, in practice, the tests cannot fully cover the objectives set (for example because certain faults or certain failure modes cannot be simulated or because certain states cannot be reached in the environment of the target system), the strategy must show what additional verifications will be carried out to ensure that the risk of non conformity is set to a reasonable level, for example: reinforcement of certain tests at the unit or integration testing phase, theoretical checks, behaviour analysis on the basis of diagrams or models.

III.3.1.3.Testing methods and techniques: basic techniques

Whatever the level of requirement, black-box functional tests must be carried out in the validation phase. Here, as a general rule, there should be no white-box tests for the following reasons:

- The validation tests must verify the conformity with the specifications, which are normally a definition of the software from an external point of view.
- The validation must be carried out on the integrated system and, therefore, with little visibility of the internal behaviour of the software.

When the designer foresees employing white-box type methods, it is necessary to demonstrate that there is a real advantage to observing the internal behaviour of the software during this phase, and that these observations cannot be made in the preceding phases. Otherwise, he can tolerate the use of this technique, with the following provisions:

- This method is not the sole method: black-box tests must exist and cover the functional specifications (these tests can also possibly include “ white box ” aspects).
- The observation of the behaviour of the system from an internal point of view does not perturb this behaviour when it is observed from an external point of view. In particular, that there are no inhibited devices in operational behaviour (tools suppressed on the operational system, inhibition of execution traces, etc.).

The main **basic techniques** that can be used in validation are:

- the Boundary Value Analysis (limits of the domains of definition of the input or output data) ;

- the Equivalence Partitioning (definition of the classes of equivalence for the input or output data values).

Remark: Other techniques

Basic techniques can be added to by other techniques. However, either these techniques are not simple to implement (tests based on Cause-Effect Graphing or tests based on Finite State Machine) or else demonstration of the coverage obtained can only be established with difficulty (Error Guessing, Probabilistic Testing). Their use remains difficult.

These techniques are:

- less formal, such as Errors Guessing (looking for errors on the basis of experience) ;
- more formal, for high-requirement levels (tests based on Cause-Effect Graphing or tests based on Finite-State Machine ;
- or requiring specific tools for particular objectives (for example, primarily focused on reliability measurement, such as Probabilistic Testing).

III.3.2.Definition of validation test procedures

Requirement 3.11 in document [1] is : "Directives for drawing up test procedures should include : a description of the input data to be used (*value*), a description of the expected output (*value)*, criteria on which test results will be judged acceptable (*tolerance*)". This requirement is required for level 2 and recommended for level 1.

A correctly applied testing process must imply that the testing procedures have been drawn up before the beginning of the validation phase, and as early as possible in the development cycle.

This remark is particularly important in the context of assessment. Indeed, it is in the interests of the assessor to assess these procedures before rather than after they are applied. This avoids bringing the validation phase into question and designing additional tests during the final assessment.

It is necessary that these procedures contain all the information necessary for:

- their identification and their ability to be traced with respect to the objectives and/or the strategy ;
- carrying out the test during the phase. (preparation activities, procedures used during testing, anomaly recording and test termination criteria activities).

III.3.3.Coverage and relevance of the validation tests

The objective is to ensure adequate test coverage.

A traceability table (cf. chapter I.7.3) is to be established in order to demonstrate that:

- the matrix-referenced tests are indeed documented by a procedure ;
- no tests exist that are not matrix referenced.

If a traceability matrix does not exist, the content of each testing procedure must be analysed afterwards to establish the matrix. This action must be refined by in-depth examination of the procedures to ensure that:

- the tests are valid, relevant and can be reproduced ;
- the tests covering a given objective are indeed conclusive for the objective ;
- the testing methods specified when planning the tests are indeed being employed.

III.3.4.Relationship with the configuration and modification management

The Configuration and Software Modifications Management is of prime importance in this phase. Indeed, it must be verified that the tests have indeed been carried out:

- in conformity with that foreseen (in other words with respect to a referenced version of the test planning and test definition documents) ;
- on the version of the software that the test process covers.

III.4. PHASE PRODUCTS

The result of validation is made up of the following elements as established by the assessor:

- Test planning documentation
- Validation reports

The table below presents the main information which must figure in each product of the phase.

Phase products
Validation test planning documentation
• Organisation, responsibilities *Persons involved in the validation. Organisation. Responsibilities.*
• Generic objectives of the tests *For example:* *- Coverage of all the functions; of all the operating modes; of safety requirements in the case of a fault; of the performance requirements;* *- Conformity with the user and operation documentation;* *- Verification of endurance, etc.*
• Testing strategies *Approach retained, with scheduling in stages aimed at specific objectives or with recourse to specific environments or techniques, for example: Test of certain groups of functions; Test of an operating mode; Test of logical sequences; Parameter-setting tests; Endurance tests; Simulation tests in certain environments, etc.*
• Testing methods and techniques *Methods and techniques employed: manual or automatic techniques, analytical or statistical tests, Equivalence Partitioning, boundary tests, etc.*

Phase products
• Testing tools and environments *Identification of the testing tools employed (simulators, measurements tools, comparators, etc.).* *If the software environment is not the same at the different stages, description of each of these environments.*
Validation reports *There must be as many reports as successive versions supplied.* • Objective of the tests *Must be defined with respect to a defined reference (requirement identifier, paragraph of a document, etc.) and form part of one of the generic objectives of the test planning.* • Testing environment and tools employed *Possibly with reference to previously defined environments.* *To be supplied by reference to the test definition documentation. To be completed by information concerning the calibration of measurement apparatus (calibration data), where appropriate, and by divergence from the test planning.* • Input data and/or signals *To be supplied with their sequencing and their value. If the tests are automated, this information can take the form of digital recordings as long as the rule and means to interpret it are available.* • Output data and/or signals. *As above* • Operating method *Following the activity required to carry out the test.* • Success criteria *Obtaining the expected data and/or signals is implicitly one of the criteria. This criterion must be completed whenever necessary by other criteria : response time, tolerance of output values, absence or occurrence of a certain event before a certain lapse of time.* • Tests carried out *Can be supplied by reference to the test procedures. To be completed by divergence from the test planning, where appropriate (tests not carried out).* • Chronology of the tests *Effective dates of the different tests, test scheduling.* • Striking facts *Striking facts (divergence from the foreseen scheduling, abandon of certain tests, etc.). Can be the subject of a separate document termed “ test log ”.* • Detail of the results obtained *For each test, output signals and/or data observed (values and sequencing).* *These results can take the form of digital recordings as long as the rules and facilities for interpreting them are available.* • Links with the modification management *For each test having failed, a link towards the corresponding modification request.* • Report of non conformities *Synthetic list of non conformities detected, with their impact on safety.* • Proof of test coverage *A list of elementary tests and a cross-referenced between the requirements to be tested and the tests identified.*

Table 5: Products at the output of the validation test phase

IV. INTEGRATION TEST PHASE

IV.1. PHASE RESPONSIBILITIES

Those responsible for the integration test phase must be designated following the same requirement with regard to independence as applied during the validation tests (cf. chapter IV.1).

IV.2. INPUT ELEMENTS OF THE PHASE

The elements listed below are necessary to the undertaking of the phase. The development team must supply the assessor with them.

Input elements of the phase
Product definition documentation • Software specifications • Software design (software architecture alone) · User documentation *All documents (manuals, procedures, etc.) defining the utilisation of the product.* *This is a useful but not indispensable addition to understand the tests. It is a necessary element if the objectives of the tests is to cover the conformity of this documentation.* • Operation and parameter-setting documentation, etc. *All documents (manuals, procedures, etc.) defining possible activity on the product beyond normal use.* *This is a useful but not indispensable addition to understand the tests. It is a necessary element if the Objectives of the tests is to cover the conformity to this documentation.*
Configuration and modification management documentation • Versions presented in integration *Precise identification of the product tested with its version and the version of the system in which it is included.* • Requests for design modifications stemming from the tests. • Impact analysis reports, decisions made and justifications concerning these requests. • Software configuration history *List of modifications carried out between successively presented versions.* • Organisation and responsibilities *Persons involved in the validation, roles and responsibilities.*

Table 6: Input elements of integration test phase

IV.3. PHASE ACTIVITIES

IV.3.1. Integration test planning

The aspects common to the three test phases (VT, IT, UT) concerning the definition of test objectives, the testing strategy, and the testing facilities are to be found in chapter I.7.1

IV.3.1.1. Definition of integration test objectives

Defining these objectives allows for the preparation of the future validation of the product by ensuring that all features have been white-box verified during the integration.

A priori, the objective of the integration tests is to cover each element of the design, including the safety mechanisms that can be verified at this stage of the tests.

IV.3.1.2.Integration testing strategy

As an example of integration testing strategy, to show what a strategy can be, the list below provides the typical stages:

- a detailed test stage for each hardware peripheral and each functional blocks ;
- a stage (or several) testing the interaction between the different functions ;
- a stage verifying the behaviour in the case of a failure ;
- an endurance testing stage. This allows verification of aspects linked to memory fragmentation (dynamical memory allocation is strongly advised against), changing day/month/year/century, fault accumulation ;
- a stage dedicated to parameter setting ;
- stages with simulated inputs and stages with real input in an operational environment ;
- etc.

As for the validation tests, the strategy must be integrated into the overall product-verification strategy. The following chapter deals with the test strategy.

IV.3.1.3.Testing methods and techniques : basic techniques

As a general rule, the integration tests are primarily white-box tests. Indeed, the integration tests must verify the conformity of the design, and are normally a definition of the software from an internal point of view. To integrate large-size software, several integration stages should be planned. At each stage of the integration, lower-level components are considered as black box, with only the exchanges between components being examined in white box.

The possibilities of instrumenting the architecture (using the development environment, installing software “ stubs ” replacing part of the software) must be taken into account at the design stage when defining the architecture. Otherwise, there is a real risk of being confronted with an architecture where all the components are accessible at the same time or where it is difficult to observe each part of the software. This type of integration must be reserved to smaller-size software (for example less than 10,000 lines) with a low level of interaction with external peripherals.

Remark:

The concept of black-box / white-box tests is recursive the further the integration stages progress. For example, with the architecture of figure 2:

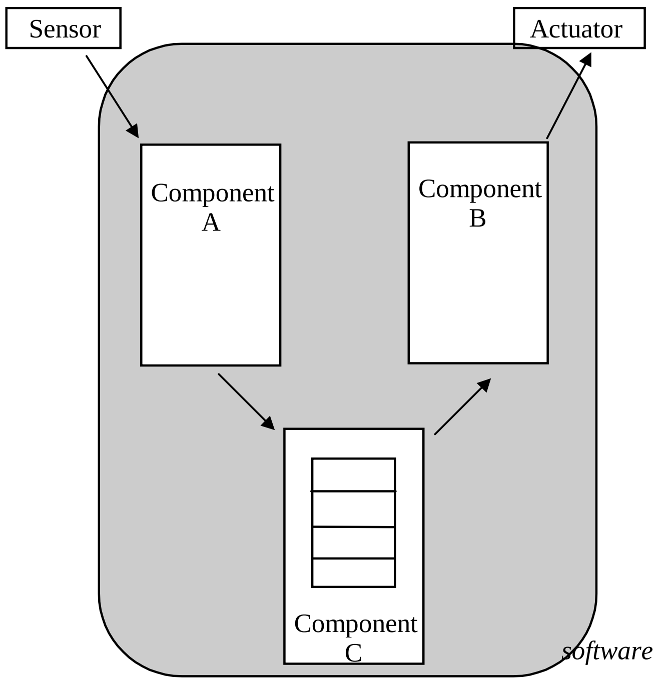

Figure2: Recursive tests

The first integration stage is carried out in white box at each architecture component level (A, B or C) or with a grouping described in the integration strategy. The internal data of the components are observed to ensure the behaviour of the software.

The integration tests, after this first integration stage, can be carried out considering each component as a black box. The variables observed, for example, for the Equivalence Partitioning are the files, messages, and data exchanged between the components. This can correspond to representations of data external to the software but also to data appearing during the design of the software.

The validation tests are carried out in black box : only the sensor data and the actuator instructions are taken into account, for example during the domain tests.

The main **basic techniques** that can be employed in integration are:

- Boundary Value Analysis (limits of the domain of definition of input or output data) ;
- Equivalence Partitioning (definition of classes of equivalence for input or output values).

Remark: Other techniques

The “Errors Guessing”, “Cause-Effect Graphing" techniques or tests based on "Finite-State Machine”and “Probabilistic Testing” can be used with the same restrictions as applied during validation testing.

IV.3.2. Definition of the integration test procedures

Requirement 3.11 in document [1] is : " Directives for drawing up test procedures should include : a description of the input data to be used (*value*), a description of the expected output (*value)*, criteria on which test results will be judged acceptable (*tolerance*)". This requirement is required for level 2 and recommended for level 1.

In principle, the method is similar to that presented for validation (Refer to chapter III.3.2: "Definition of validation test procedures").

IV.3.3. Coverage and relevance of the integration tests

In principle, the method is similar to that presented for validation (Refer to chapter III.3.3: "Coverage and relevance of the validation tests").

IV.3.4. Relationship with the configuration and modification management

As for validation, the control of configuration and of modifications is of vital importance in this part of the integration test phase verification.

In principle, the method is identical to that presented for validation in chapter III.3.4.

The additional difficulty is to link up the integration test tools and simulators employed with the configuration management. Frequently, there is no longer a written trace of the tests actually carried out. The assessor must therefore examine the version of the software used to conduct the tests and compare with the modification files (date) to ensure that the non-regression problems have indeed been taken into account.

IV.4. PHASE PRODUCTS

The result of the integration phase is made up of the following elements established by those responsible for validation:

- Test planning documentation
- Integration reports

The following tables present the principal information that must appear in each of the products of the phase.

Phase products
<u>Integration test planning documentation</u>
• Organisation, responsibilities *Persons involved in the integration. Organisation. Responsibilities.* • Generic objectives of the tests • Testing strategy *Approach retained, with scheduling in stages targeting specific objectives or having recourse to specific environments or techniques.* • Testing methods and techniques *Methods and techniques employed, for example: manual or automatic techniques, analytic or statistical techniques ; Equivalence Partitioning ; Boundary Value Analysis, etc.* • Testing environments and tools *Identification of the testing tools employed (simulators, measurement tools, comparator, etc.).* *If the software environment is not the same in the different stages, description of each of these environments.* • Proof of test coverage *For example, a list of elementary tests and a cross reference between the requirements to be tested and the tests identified.*

Phase products
Integration reports
There must be as many reports as successive versions supplied.
• Objective of the test *Must be defined with respect to a determined reference (requirement identifier, paragraph of a document, etc.) and form part of one of the generic objectives of the test planning.*
• Testing environment and tools employed *Possibly with reference to previously defined environments.*
• Input data and/or signals *To be supplied with their sequencing and their value. If the tests are automated, this information can take the form of digital recordings as long as the rules and means to interpret it are available.*
• Output data and/or signals *As above.*
• Operating mode *Activities required to carry out the test.*
• Success criteria *Obtaining the expected data and/or signals is implicitly one of the criteria. This criterion must be completed whenever necessary by other criteria : response time, tolerance of output values, absence or occurrence of a certain event before a certain lapse of time.*
• Tests carried out *Can be supplied by reference to the test definition documentation. To be completed by divergence from the test planning, where appropriate (tests not carried out).*
• Chronology of the tests *Effective dates of the different tests, test scheduling.*
• Striking facts *Striking facts (divergence from the foreseen scheduling, abandon of certain tests, etc.). Can be the subject of a separate document termed “ test log ”.*
• Detail of the results obtained *For each test, output signals and/or data observed (values and sequencing).* *These results can take the form of digital recordings as long as the rules and facilities for interpreting them are available.*
• Links with the modification management *For each test having failed, a link towards the corresponding modification request.*
• Report of non conformities *Synthetic list of non conformities detected, with their impact on safety.*
• Proof of test coverage *For example, a list of elementary tests and a cross-referenced between the requirements to be tested and the tests identified.*

Table 7: Products at the end of the integration tests

V. UNIT TEST PHASE

At the outset, it should be noted that the concept of the unit associated to the unit tests has not yet been defined as regards the programming languages employed. This can therefore be interpreted as an item of code of varying size:

- a procedure or a function,
- a procedure or a function visible from the exterior of a compilation unit (for example, an " operator ", " a method " for oriented object),
- a unit of compilation,
- a software component (tree structure of n compilation unit levels with n≥1.

V.1. PHASE RESPONSIBILITIES

The person responsible for the unit test must be designated as early as the outset of the detailed conception phase.

As much independence from the developer as possible should be established. However, in the case of small-scale projects (software team made up of one or two people, for instance) this is rarely possible.

In such cases, those responsible for software development and those responsible for software validation will make every possible effort to ensure that an external body (for example, the assessor) can satisfactorily establish that the phase has been completed in line with software safety requirements.

V.2. INPUT ELEMENT OF THE PHASE

The elements listed below are necessary to the undertaking of the phase. The development team must supply the assessor with them.

Input elements of the phase
<u>Product definition documentation</u>
• Software specifications
• Software design (preliminary and detailed)
· User documentation *All documents (manuals, procedures, etc.) defining the utilisation of the product.* *This is a useful but not indispensable addition to understand the tests. It is a necessary element if the objectives of the tests is to cover the conformity of this documentation.*
• Operation and parameter-setting documentation, etc. *All documents (manuals, procedures, etc.) defining possible activity on the product beyond normal use.* *This is a useful but not indispensable addition to understand the tests. It is a necessary element if the Objectives of the tests is to cover the conformity to this documentation.*

<u>Configuration and modification management documentation</u>
• Versions presented *Precise identification of the product tested with its version and the version of the system in which it is included.*
• Requests for design modifications stemming from the tests.
• Impact analysis reports, decisions made and justifications concerning these requests.
• Software configuration history *List of modifications carried out between successively presented versions.*
• Organisation and responsibilities *Persons involved in the validation, roles and responsibilities.*

Table 8: Input elements of the unit test phase

V.3. PHASE ACTIVITIES

V.3.1. Unit test planning

The aspects common to each of the three phases (VT, IT, UT) concerning the definition of test objective, the testing strategy, and the testing facilities are to be found in chapter I.7.1

V.3.1.1.Definition of unit test objectives

A priori, the objective of the unit tests is to cover each element of the detailed design, including the safety mechanisms that can be verified at this stage of the tests.

Each software module must be subject to tests that verify, by means of data supplied at the input, that the module fulfils the functions required in the detailed design.

V.3.1.2.Unit testing strategy

Important: The unit tests must be distinguished from the simple development debugging carried out by the designer. The difference between unit tests and the development debugging lies in the following points:

- unit tests are planned,
- a strategy exists (coverage of the control or data flow of the unit),
- the result expected is described before the tests are carried out,
- a trace of execution and of the results obtained must be recorded in a report comprising, as a minimum, the software version, the inputs applied and the results both expected and obtained.

<u>Remark:</u>

It should be borne in mind that it is preferable to insist on the choice of programming environment rather than on the effort put into the unit test. These choices concern:

- the programming rules, which also allow the complexity of the items of code to be limited. It is an illusion to carry out unit tests on models that are too complex, poorly structured or that manipulate digital data with no precautions.

- the programming language: all the integrity checks offered by the language limit the explosion of the number of tests.

It is already too late at the start of the unit test campaign to go back on possible poor choices.

The effort put into the unit tests must be adapted to the criticity level of the software. Several methods are employed to limit the effort required for these tests:

- limitation of the complexity of the data flow (no global variables, few variables common to a component) and of the control flow (number of decisions taken by the software (if, as long as, etc.),
- focusing on the units where the integration test does not provide correct coverage of the structure.

The following chapter concerns test strategy.

V.3.1.3.Testing methods and techniques: basic techniques

The unit test techniques are often complementary or redundant. For example, to test the control flow of the following item of code:

```
Until condition_1 and condition_2 and (loop counter < n)
        <instructions to be carried out>
        increment the loop counter
end of Until
```

In this example, we have:

- *branch testing* :

  Two test cases to cover the branches: the loop is entered into or not.

- *decision testing* :

  Eight test cases (2^3) to cover the combination of three under decision conditions. These tests are particularly important when the decisions are used to change the state of the software.

- *path testing* :

  n+1 test cases are necessary to cover the different paths if the loop counter is a positive integer.

Conclusion:

- Branch coverage represents a minimum objective (is a minimum objective) to be reached for the software important for safety.
- The path testing (and the decision testing) soon becomes impractical for the unit test. However, this type of test provides the best representation of software behaviour and of the real complexity of the code. As a result, if the number of unit tests of code paths is considerable, it is no doubt preferable to attempt simplification of the code.
- The presence of loops in sections of the code important for safety is unacceptable without justification.

Remark: An unattainable code can be tolerated as long as it consists of processing for fault tolerance allowing verification that the integrity constraints are still satisfied. Such processing is only used for protecting against possible regression on unintentional modification of the source.

V.3.2. Definition of the unit test procedures

Requirement 3.11 in document [1] is required for level 2 and recommended for level 1.

In principle, the method is similar to that presented for validation (Refer to chapter III.3.2: "Definition of validation test procedures").

V.3.3. Coverage and relevance of the validation tests

In principle, the method is similar to that presented for validation (Refer to chapter III.3.3: "Coverage and relevance of the validation tests").

V.3.4. Relationship with the configuration and modification management

As in the other test phases, the control of configuration and of modifications is of vital importance in this part of the unit test phase verification.

In principle, the method is identical to that presented for validation in chapter III.3.4. It must be verified that the tests have indeed been carried out:

- in conformity with that foreseen (in other words with respect to a referenced version of the test planning and test definition documents) ;
- on the software version that the test process covers.

V.4. PHASE PRODUCTS

The result of the unit tests is made up of the following elements established by the assessor:

- Test planning documentation
- Unit test reports

The table below presents the principal information that must appear in each of the products of the phase.

Phase products
Unit tests planning documentation
• Organisation, responsibilities *Persons involved in the integration. Organisation. Responsibilities.* • Generic objectives of the tests • Testing strategy *Approach retained, with scheduling in stages targeting specific objectives or having recourse to specific environments or techniques.* • Testing methods and techniques *Methods and techniques employed, for example : manual or automatic techniques, analytic or statistical techniques ; Equivalence Partitioning ; Boundary Value Analysis, etc.*

Phase products
• Testing environments and tools *Identification of the testing tools employed (simulators, measurement tools, comparator, etc.).* *If the software environment is not the same in the different stages, description of each of these environments.* • Proof of test coverage *For example, a list of elementary tests and a cross reference between the requirements to be tested and the tests identified.*
Unit Test reports *There must be as many reports as successive versions supplied.* • Objective of the test *Must be defined with respect to a determined reference (requirement identifier, paragraph of a document, etc.) and form part of one of the generic objectives of the test planning.* • Testing environment and tools employed *Possibly with reference to previously defined environments.* • Input data and/or signals *To be supplied with their sequencing and their value. If the tests are automated, this information can take the form of digital recordings as long as the rules and means to interpret it are available.* • Output data and/or signals *As above.* • Operating mode *Activities required to carry out the test.* • Success criteria *Obtaining the expected data and/or signals is implicitly one of the criteria. This criterion must be completed whenever necessary by other criteria : response time, tolerance of output values, absence or occurrence of a certain event before a certain lapse of time.* • Tests carried out *Can be supplied by reference to the test definition documentation. To be completed by divergence from the test planning, where appropriate (tests not carried out).* • Chronology of the tests *Effective dates of the different tests, test scheduling.* • Striking facts *Striking facts (divergence from the foreseen scheduling, abandon of certain tests, etc.). Can be the subject of a separate document termed “ test log ”.* • Detail of the results obtained *For each test, output signals and/or data observed (values and sequencing).* *These results can take the form of digital recordings as long as the rules and facilities for interpreting them are available.* • Links with the modification management *For each test having failed, a link towards the corresponding modification request.* • Report of non conformities *Synthetic list of non conformities detected, with their impact on safety.* • Proof of test coverage *For example, a list of elementary tests and a cross-referenced between the requirements to be tested and the tests identified.*

Table 9: Products at the end of the Unit Tests phase

VI. CONCLUSION

This document has been drawn up for the manufacturer's test team from a pedagogical point of view. It is intended to assist the assessor in constructing his own technique, and in adapting it to the specific context both upstream of and during the development process.

A set of basic techniques and reference points is given that help the assessor both to collect and analyse the information provided by the different phases of the development process: validation, integration and unit test phases

The presented test process follows the same principles used in all other creation, construction or design processes if quality is to be ensured, that are:

- initial definition
- execution according to the previously defined procedures
- verification according to fixed modalities.

The use of such a process allow to construct efficient software tests that will detect possible divergences between the expected behaviour and the behaviour encountered during the tests and will give the necessary confidence before operational use. However, these software tests are essential to detect errors and to improve confidence in the ability of the system to accomplish its task, but are insufficient to “ prove ” the behaviour of a system.

VII. REFERENCE DOCUMENTS

[1] **SOFTWARE QUALITY AND SAFETY REQUIREMENTS**
STSARCES Project - WP 1.2 / Aspect 1 – Final Report - INRS – Feb 2000

[2] **GUIDE TO EVALUATING SOFTWARE QUALITY AND SAFETY REQUIREMENTS**
STSARCES Project - WP 1.2 / Aspect 2 – Final Report - INRS – Feb 2000

[3] **SAFETY OF MACHINERY - Safety-related parts of control systems; Part 1: General principles for design**
EN 954-1 - December 1996

[4] **CEI 61508: SURETE FONCTIONNELLE : systèmes relatifs à la sûreté - Version 1998**
Partie 1 : Prescriptions générales
Partie 2 : Exigences pour les systèmes électriques / électroniques / électroniques programmables
Partie 3 : Prescriptions concernant les logiciels
Partie 4 : Définitions et abréviations
Partie 5 : Exemples de méthodes pour la détermination des niveaux d'intégrité de sécurité
Partie 6 : Guide pour l'application des parties 2 et 3
Partie 7 : Bibliographie des techniques et des mesures

APPENDIX : REFERENCES FOR TESTING STRATEGIES

The following works can be useful if one wishes to obtain more precise information on the construction of a software testing method. The list presented below contains some relevant work. The references are presented by order of interest, from the point of view of software testing, and a brief summary is presented for each one.

Joe ABBOT — **Unit and integration testing manual, NCC Publications, 1988.**

This work describes a module and integration testing method, which is presented in the form of procedures and steps to follow. For each action to be taken, a clear and precise description is given. Following this, a basis of some experience acquired in the area of software testing is presented.

Joe ABBOT — **Software testing techniques,
NCC Publications, 1988.**

This work deals with how to ensure that a system functions as specified.

It introduces the concept of software testing with respect to the quality assurance portion lifecycle and the review of the documentation.

The author then describes the methods and techniques currently used to this end.

Stephen J. ANDRIOLE — **Software Validation Verification, Testing and Documentation
Petrocelli books, 1986.**

This work presents a variety of structural testing methods. It defines and develops a complex measurement programme, and then describes the criteria and procedures for implementing the structural testing process, as well as the manner in which these techniques can be used during the maintenance phase.

The author also presents a reference guide to the techniques and tools that can be used for validation, verification and testing. This guide contains approximately thirty case descriptions, outlining for each one which input and output data are necessary, an example and an estimation of the learning time and necessary resources.

Martin L. SHOWMAN — **Software engineering, design, reliability and management,
McGraw-Hill, 1983**

A single chapter deals with software testing. It assembles the techniques and observations dealing with tests by presenting the use of statistical data in the testing process, the types of tests, and a comparison of these tests through a display of the results in different arrays, sorted according to qualitative criteria and efficiency.

It also presents different debugging procedures for models of control graphs, and a classification of tests in five different groups. In fact, the goal of the author is to reduce testing to a collection of software techniques and to elevate the art of the evaluator to the level of a scientific discipline.

G.J. MYERS — **The art of software testing
John Wiley and Sons, 1979.**

This publication brings together all methods and techniques used in the area of software testing.

It offers an open and very wide view of software testing, which allows it to be used as a practical reference by anyone obliged to deal with this issue.

Additionally, the author tries to instil those involved in testing with the correct attitude.

Martin A. OULD — **Testing in software development.**

Charles UNWIN — **Cambridge University Press, 1986.**

This work is entirely consecrated to software tests, but seen from a different angle. The test is first of all presented from the test management point of view - i.e. the test is placed in relation to the overall structure of the software project - and then the organisation of tests and personnel, the planning and the configuration management are described.

Following this, the author treats testing from different points of view :

- the user : through requirements and specifications tests, and the acceptation of the test plan.
- the designer : through design, integration and system tests.
- the programmer : the testing activity focuses on module specifications, planning of module tests and module tests themselves.

Norman PARRINGTON
Marc ROPER

Understanding software testing.
Ellis Horwood ltd., 1989.

This is a work that tries to be practical, and does not become lost in the description of methods or techniques that are difficult to implement.

The authors bring the testing activity into play very early in the software lifecycle. The originality of this work resides in the fact that every description of a technique or method is applied to a case study throughout the work. Both tests based on requirements and design tests are presented. The work finishes with tests on the system integration and validation in its final environment.

W.J. QUIRK

Verification and validation of real time software.
Springer-Verlag, 1985.

In the first part of this book, a description of software testing is given from a systematic point of view. It mentions the methods that use internal and external information (with respect to the software), and then compares and evaluates the different methods and techniques presented.

Following this, statistical tests that complete the systematic verification of real time software are presented. Several equations that can be used to estimate the risk of errors are also given.

David J. SMITH
Kenneth B. WOOD.

Engineering Quality Software.
Elsevier Applied Science, 1989.

This work gives an overview of different testing strategies, and then presents statistical analysers with a precise description of the MALPAS, SPADE and TESTBED tools. Dynamic tests are then described by enumerating different levels of tests and tools. The work finishes its approach to testing by presenting test management as going hand in hand with the activity of writing the test itself.

Roger S. PRESSMAN

Software engineering : a practitioner's approach,
McGraw Hill, 1982.

Software testing is presented in one of the chapters by a description of the testing phase, including the test objectives, the test in a flow of information, black and white box testing. After which the author develops the module test, the steps in which testing intervenes, the integration test, and finishes by giving a list of computer tools.