

Software testing —

Part 2: Software component testing

ICS 35.080

NO COPYING WITHOUT BSI PERMISSION EXCEPT AS PERMITTED BY COPYRIGHT LAW

BSI
Documents

AVA

Document Sharing Hub

Committees responsible for this British Standard

The preparation of this British Standard was entrusted to Technical Committee IST/15, Software engineering, upon which the following bodies were represented:

- British Computer Society
- Defence Research Agency
- Institution of Electrical Engineers
- Lloyds Register of Shipping
- Ministry of Defence
- National Computing Centre Ltd.

This British Standard, having been prepared under the direction of the DISC Board, was published under the authority of the Standards Board and comes into effect on 15 August 1998

© BSI 1998

The following BSI references relate to the work on this standard:
Committee reference IST/15
Draft for Public Comment 97/644971 DC

ISBN 0 580 29556 7

Amendments issued since publication

Amd. No.	Date	Text affected

Contents

	Page
Committees responsible	Inside front cover
Foreword	iii
Introduction	1
1 Scope	1
2 Normative reference	1
3 Definitions	2
4 Process	2
5 Test case design techniques	3
6 Test measurement techniques	6
Annex A (informative) Process guidelines	10
Annex B (informative) Guidelines for testing techniques and test measurement	17
Annex C (informative) Test technique effectiveness	48
Bibliography	Inside back cover
Figure 1 — Generic component test process	2
Figure A.1 — Document hierarchy	10
Figure A.2 — Example of component test strategy	12
Figure A.3 — Example of project component test plan	14
Figure A.4 — Example of component test plan	14
Figure A.5 — Example of test specification	15
Figure A.6 — Example of component test report	16
Figure B.1 — Input “exam mark”	17
Figure B.2 — Input “coursework mark”	18
Figure B.3 — Equivalence partitions and boundaries for total mark	18
Figure B.4 — Input exam mark	23
Figure B.5 — Input coursework mark	23
Figure B.6 — Boundary values	23
Figure B.7 — Equivalence partitions and boundaries	24
Figure B.8 — State transition diagram	26
Figure B.9 — STD for <i>manage_display_changes</i>	27
Figure B.10 — Cause-effect graph	30
Figure B.11 — Control flow graph for binsearch	35
Figure C.1 — Partial ordering of structural test coverage criteria	49
Table B.1 — Input exam mark	19
Table B.2 — Input coursework mark	19
Table B.3 — Invalid inputs	20
Table B.4 — Valid outputs	20
Table B.5 — Invalid outputs	20
Table B.6 — Minimized test case suite	22
Table B.7 — Input exam mark	23
Table B.8 — Input coursework mark	23
Table B.9 — Valid outputs	24
Table B.10 — Additional test cases	25
Table B.11 — Additional test cases	25
Table B.12 — Possible transitions	27
Table B.13 — <i>manage_display_changes</i> component	28

	Page
Table B.14 — Notation for state tables	28
Table B.15 — State table for the <i>manage_display_changes</i> component	28
Table B.16 — State table	29
Table B.17 — Test cases	29
Table B.18 — Decision table	30
Table B.19 — Test cases	30
Table B.20 — Test cases	31
Table B.21 — Test cases to cover mutations	32
Table B.22 — Test cases	35
Table B.23 — Occurrence of variables and their categories	36
Table B.24 — du-pairs and their type	37
Table B.25 — All-c-uses	38
Table B.26 — All-c-uses	38
Table B.27 — Test inputs	39
Table B.28 — Alternative test inputs	39
Table B.29 — Test inputs	40
Table B.30 — Test cases	40
Table B.31 — Test cases	40
Table B.32 — Test cases	40
Table B.33 — Overall set of test cases	41
Table B.34 — Test cases	41
Table B.35 — Initial test case set	44
Table B.36 — LCSAJs	44
Table B.37 — Additional tests	45
Table B.38 — New test sets	45

Foreword

This part of BS 7925 was prepared by Technical Committee IST/15. It was developed by a working party of the British Computer Society Specialist Interest Group in Software Testing (BCS SIGIST).

A British Standard does not purport to include all the necessary provisions of a contract. Users of British Standards are responsible for their correct application.

Compliance with a British Standard does not of itself confer immunity from legal obligations.

Summary of pages

This document comprises a front cover, an inside front cover, pages i to iv, pages 1 to 50, an inside back cover and a back cover.

Introduction

This standard specifies a process for the testing of software components and techniques for the design and measurement of that testing. It enables users to improve the quality of their software testing and thus the quality of their software products.

The standard can be used in the context of both the “Inspection and testing” provisions of BS EN ISO 9001 and the “Software coding and testing” provisions of BS ISO/IEC 12207.

An important attribute of this standard is that it contains objective provisions which make it possible to determine whether or not it has been followed in a particular case. The standard includes the concept of measuring testing as well as assessment of whether the testing meets defined targets.

As there are many challenges in software testing, the standard is deliberately limited in scope to cover only the lowest level of independently testable software. Because the interpretation of, and name for, this lowest level is imprecise, the term “component” has been chosen rather than other common terms such as “unit”, “module”, or “program” to avoid confusion with them (see BS 7925-1).

1 Scope

This British Standard specifies a process for the testing of software components and techniques for the design and measurement of that testing.

This standard is fully applicable to the testing of components written in procedural languages. The functional test case design techniques and test measurement techniques specified are applicable to all languages. The structural test case design techniques and test measurement techniques specified may not be applicable to non-procedural languages.

This standard prescribes the generation of test cases using only valid input data values, i.e. values for which the behaviour of the component under test is defined by its specification.

NOTE Good practice in component testing would also include test cases that use invalid input data values as a means of testing the adequacy of the component’s specification. Formalization of such “error guessing” is excluded from this standard.

This standard does not cover the following:

— *some types of testing*

This standard excludes a number of areas of software testing, e.g. integration testing, system testing, user acceptance testing, statistical testing, testing of non-functional attributes such as performance, testing of real-time aspects, testing of concurrency, static analysis such as data flow or control flow analysis, reviews and inspections (even as applied to components and their tests).

— *test completion criteria*

This standard does not prescribe test completion criteria as it is designed to be used in a variety of software development environments and application domains. Test completion criteria will vary according to the business risks and benefits of the application under test.

— *selection of test case design techniques*

This standard does not prescribe which test case design techniques are to be used. Only appropriate techniques should be chosen and these will vary according to the software development environments and application domains.

— *selection of test measurement techniques*

This standard does not prescribe which test measurement techniques are to be used. Only appropriate techniques should be chosen and these will vary according to the software development environments and application domains.

— *implementation*

This standard does not prescribe how required attributes of the test process are to be achieved, for example, by manual or automated methods.

— *fault removal*

This standard does not address fault removal. Fault removal is a separate process from fault detection.

2 Normative reference

The following normative document contains provisions which, through reference in this text, constitute provisions of this British Standard. For dated references, subsequent amendments to, or revisions of, any of these publications do not apply. For undated references, the latest edition of the publication referred to applies.

BS 7925-1, *Software component testing — Vocabulary*.

3 Definitions

For the purposes of this part of BS 7925, the terms and definitions given in BS 7925-1 apply.

4 Process

4.1 General

Before any component testing may begin there shall be a project component test strategy (4.2), and a project component test plan (4.3). Every component shall have a component specification which shall allow the output of the component to be determined for any sequence of inputs given an initial state and a defined environment for the component.

NOTE See annex A for general guidance on the process.

4.2 Project component test strategy

4.2.1 The project component test strategy shall specify:

- the techniques to be employed in the design of test cases, and the rationale for their choice; these techniques shall be selected from those listed in clause 5;
- criteria for test completion and the rationale for their choice; these test completion criteria shall be test coverage levels, the measurement of which shall be achieved by using test measurement techniques chosen from those listed in clause 6;
- the degree of independence from the design process required of personnel who design test cases;

NOTE Typical arrangements include the following:

- the test cases are designed by the person(s) who writes the component under test;
 - the test cases are designed by another person(s);
 - the test cases are designed by (a) person(s) from a different section;
 - the test cases are designed by (a) person(s) from a different organization;
 - the test cases are not chosen by a person.
- d) whether the component testing is performed using isolation, bottom-up or top-down approaches, or some mixture of these;
- e) the environment in which component tests will be executed, including a description of the hardware and software environment in which all component tests will be run;
- f) the test process that shall be used for component testing; for any given component, the specified test process shall start with component test planning and finish with verification of component test completion; each test case for a component shall first be specified, then executed and then recorded in the component test record.

The component test is illustrated in Figure 1.

NOTE The sequence of component test specification, component test execution, and component test recording may be iterated and, on any one iteration, may be performed for a subset of the test cases associated with a component. Later activities for one test case may occur before earlier activities for another.

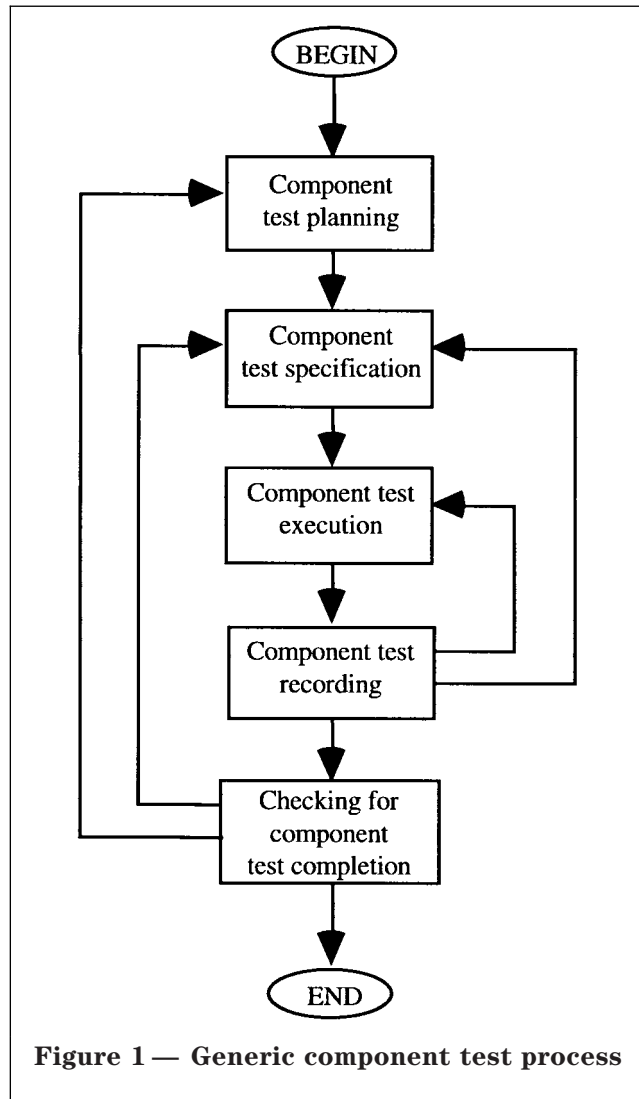


Figure 1 — Generic component test process

4.3 Project component test planning

The project component test plan shall specify the dependencies between component tests and their sequence. Their derivation shall include consideration of the chosen approach to component testing [see 4.2.1d)].

NOTE Dependencies may also be influenced by overall project management and work scheduling considerations.

4.4 Component test planning

For each component a component test plan shall specify how the project component test strategy (4.2) and project component test plan (4.3) apply to the given component. This shall include specific identification of all exceptions to the component test strategy and all software with which the component under test will interact during test execution, such as drivers and stubs.

4.5 Component test specification

4.5.1 For each component test cases shall be designed using the test case design techniques identified in the component test plan (4.4).

4.5.2 Each test case shall be specified by defining its objective, the initial state of the component, its input, and the expected outcome. The objective shall be stated in terms of the test case design technique being used.

4.5.3 The execution of each test case shall be repeatable.

4.6 Component test execution

Each test case defined in the component test specification shall be executed.

4.7 Component test recording

4.7.1 The test records for each test case shall unambiguously record the identities and versions of the component under test and the test specification. The actual outcome shall be recorded. It shall be possible, by reference to the test records, to establish whether all the specified test cases have been performed.

4.7.2 The actual outcome shall be compared against the expected outcome. Any discrepancy found shall be logged and analysed in order to establish:

- a) where the fault lies; and
- b) the earliest test activity that should be repeated in order to remove the discrepancy in the test specification or verify the removal of the fault in the component.

4.7.3 Whenever a fault is corrected by making a change or changes to test materials or the component under test, the affected activities shall be repeated.

4.7.4 A record shall be made of the test coverage levels achieved for those measures that are specified as test completion criteria in the project component test strategy.

4.8 Verification of component test completion

For each component the component test records shall be checked against the previously specified test completion criteria. If these criteria are not met, the earliest test activity that has to be repeated in order to meet the criteria shall be identified and the test process shall be restarted from that point.

NOTE It may be necessary to repeat the component test specification activity to design further test cases in order to meet a test coverage target.

5 Test case design techniques

5.1 Equivalence partitioning

5.1.1 Component analysis

Equivalence partitioning shall use a model of the component that partitions the input and output values of the component. The input and output values shall be derived from the specification of the component's behaviour.

The model shall comprise partitions of input and output values. Each partition shall contain a set or range of values, chosen such that all the values can reasonably be expected to be treated by the component in the same way (i.e. they may be considered "equivalent").

5.1.2 Test case design

Test cases shall be designed to exercise partitions of input and output values.

NOTE A single test case may exercise any number of partitions.

A test case shall comprise the following:

- the input(s) to the component;
- the partition(s) exercised;
- the expected outcome of the test case.

5.2 Boundary value analysis

5.2.1 Component analysis

Boundary value analysis shall use a model of the component that partitions the input and output values of the component into a number of ordered sets with identifiable boundaries. These input and output values shall be derived from the specification of the component's behaviour.

The model shall comprise bounded partitions of ordered input and output values. Each partition shall contain a set or range of values, chosen such that all the values can reasonably be expected to be treated by the component in the same way (i.e. they may be considered "equivalent"). The boundaries of a partition are normally defined by the values of the boundaries between partitions, however where partitions are disjoint the minimum and maximum values in the range which makes up the partition shall be used.

5.2.2 Test case design

For each identified boundary, three test cases shall be produced corresponding to values on the boundary and an incremental distance either side of it. This incremental distance shall be defined as the smallest significant value for the data type under consideration.

A test case shall comprise the following:

- the input(s) to the component;
- the partition boundaries exercised;
- the expected outcome of the test case.

5.3 State transition testing

5.3.1 Component analysis

State transition testing shall use a model of the states the component may occupy, the transitions between those states, the events which cause those transitions, and the actions which may result from those transitions.

The model shall comprise states, transitions, events and actions, where:

- inputs to the component cause events;
- events cause actions and transitions between states (transitions can return to the same state where they began);
- actions may cause outputs from the component.

The states of the model shall be disjoint, identifiable and finite in number.

NOTE The model will typically be represented as a state transition diagram, state transition model, or a state table.

5.3.2 Test case design

Test cases shall be designed to exercise transitions between states.

NOTE A test case may exercise any number of transitions.

For each test case, the following shall be specified:

- the starting state of the component;
- the input(s) to the component;
- the expected outputs from the component;
- the expected final state.

For each expected transition within a test case, the following shall be specified:

- the starting state;
- the event which causes transition to the next state;
- the expected action caused by the transition;
- the expected next state.

5.4 Cause-effect graphing

5.4.1 Component analysis

Cause-effect graphing shall use a model of the logical relationships between causes and effects for the component, where:

- each cause is expressed as a condition, which is either true or false (i.e. a Boolean) for an input, or combination of inputs, to the component; and
- each effect is expressed as a Boolean expression representing an outcome, or a combination of outcomes, for the component having occurred.

NOTE The model is typically represented as a Boolean graph relating the derived input and output Boolean expressions using the Boolean operators: AND, OR, NAND, NOR, NOT. From this graph, or otherwise, a decision (binary truth) table representing the logical relationships between causes and effects is produced.

5.4.2 Test case design

Test cases shall be designed to exercise rules, which define the relationship between the component's inputs and outputs, where each rule corresponds to a unique possible combination of inputs to the component that have been expressed as Booleans.

For each test case the following shall be identified:

- Boolean state (i.e. true or false) for each cause;
- Boolean state for each effect.

5.5 Syntax testing

5.5.1 Component analysis

Syntax testing shall use a model of the formally defined syntax of the inputs to a component, where the syntax is represented as a number of rules each of which defines the possible means of production of a symbol in terms of sequences of, iterations of, or selections between other symbols.

5.5.2 Test case design

Test cases shall be designed from the formally defined syntax of the inputs to the component.

Test cases shall be designed to execute options which are derived from rules including the following (although additional rules may also be applied where appropriate):

- whenever a selection is used, an option is derived for each alternative by replacing the selection with that alternative;
- whenever an iteration is used, at least two options are derived, one with the minimum number of iterated symbols and the other with more than the minimum number of repetitions.

For each test case the following shall be identified:

- the input(s) to the component;
- option(s) exercised;

NOTE A test case may exercise any number of options.

- the expected outcome of the test case.

5.6 Statement testing

5.6.1 Component analysis

Statement testing shall use a model of the source code which identifies statements as either executable or non-executable.

5.6.2 Test case design

Test cases shall be designed to exercise executable statements.

For each test case, the following shall be specified:

- the input(s) to the component;
- identification of statement(s) to be executed by the test case;
- the expected outcome of the test case.

5.7 Branch/decision testing

5.7.1 Component analysis

Branch testing shall use a model of the source code which identifies decisions and decision outcomes (i.e. possible transfer of control), where a decision is an executable statement which may transfer control to another statement depending upon the logic of the decision statement.

NOTE Typical decisions are found in loops and selections. Each possible transfer of control is a decision outcome.

5.7.2 Test case design

Test cases shall be designed to exercise decision outcomes.

For each test case, the following shall be specified:

- the input(s) to the component;
- identification of decision outcome(s) to be executed by the test case;
- the expected outcome of the test case.

5.8 Data flow testing

5.8.1 Component analysis

Data flow testing shall use a model of the interactions between parts of a component connected by the flow of data as well as the flow of control.

Categories shall be assigned to occurrences of variables in the component, where:

- a category identifies the “definition” or the “use” of the variable at that point;
- “definitions” are occurrences of variable occurrences where a variable is given a new value. The entry point of a component shall be considered to have a definition of each parameter, each non-local variable, and the input buffer (which may implicitly appear in calls to procedures or functions);
- “uses” are variable occurrences where a variable is not given a new value; “uses” can be further distinguished as either “data definition p-uses” or “data definition c-uses”. A data definition p-use shall be considered to occur for each decision outcome of the predicate portion of a decision statement such as `while .. do`, `if .. then .. else`, etc. Data definition c-uses are all others, including variable occurrences in the right hand side of an assignment statement, or an output statement. The exit point of the component is considered to have a c-use of each parameter, each non-local variable, and the input buffer.

The control flow model for the component shall be derived and the location and category of variable occurrences on it shall be identified.

5.8.2 Test case design

Test cases shall be designed to execute control flow paths between definitions and uses of variables in the component.

Each test case shall include:

- the input(s) to the component;
- locations of relevant variable definition and use pair(s);
- control flow sub-path(s) to be exercised;
- the expected outcome of the test case.

5.9 Branch condition testing

5.9.1 Component analysis

Branch condition testing shall use a model of the source code which identifies decisions and the individual Boolean operands within the decision conditions, where:

- a decision is an executable statement which may transfer control to another statement depending upon the logic of the decision statement; and
- a decision condition is a Boolean expression which is evaluated to determine the outcome of a decision.

NOTE Typical decisions are found in loops and selections.

5.9.2 Test case design

Test cases shall be designed to exercise individual Boolean operand values within decision conditions.

For each test case, the following shall be specified:

- the input(s) to the component;
- for each decision evaluated by the test case, identification of the Boolean operand to be exercised by the test case and its value;
- the expected outcome of the test case.

5.10 Branch condition combination testing

5.10.1 Component analysis

Branch condition combination testing shall use a model of the source code which identifies decisions and the individual Boolean operands within the decision conditions, where:

- a decision is an executable statement which may transfer control to another statement depending upon the logic of the decision statement; and
- a decision condition is a Boolean expression which is evaluated to determine the outcome of a decision.

NOTE Typical decisions are found in loops and selections.

5.10.2 Test case design

Test cases shall be designed to exercise combinations of Boolean operand values within decision conditions.

For each test case, the following shall be specified:

- the input(s) to the component;
- for each decision evaluated by the test case, identification of the combination of Boolean operands to be exercised by the test case and their values;
- the expected outcome of the test case.

5.11 Modified condition decision testing

5.11.1 Component analysis

Modified condition decision testing shall use a model of the source code which identifies decisions, outcomes, and the individual Boolean operands within the decision conditions, where:

- a decision is an executable statement which may transfer control to another statement depending upon the logic of the decision statement; and
- a decision condition is a Boolean expression which is evaluated to determine the outcome of a decision.

NOTE Typical decisions are found in loops and selections.

5.11.2 Test case design

Test cases shall be designed to demonstrate that Boolean operands within a decision condition can independently affect the outcome of the decision.

For each test case, the following shall be specified:

- the input(s) to the component;
- for each decision evaluated by the test case, identification of the combination of Boolean operands to be exercised by the test case, their values, and the outcome of the decision;
- the expected outcome of the test case.

5.12 Linear code sequence and jump (LCSAJ) testing

5.12.1 Component analysis

LCSAJ testing shall use a model of the source code which identifies control flow jumps (i.e. where control flow does not pass to a sequential statement), where an LCSAJ is a triple: the start of the linear code sequence, the end of the linear code sequence, and the target line to which control flow is transferred.

NOTE LCSAJs are conventionally identified by line numbers in a source code listing.

5.12.2 Test case design

Test cases shall be designed to exercise LCSAJs.

For each test case, the following shall be specified:

- the input(s) to the component;
- identification of the LCSAJ(s) to be executed by the test case;
- the expected outcome of the test case.

5.13 Random testing

5.13.1 Component analysis

Random testing shall use a model of the input domain of the component that defines the set of all possible input values. The input distribution (normal, uniform, etc.) to be used in the generation of random input values shall be based on the expected operational distribution of inputs. Where no knowledge of this operational distribution is available, a uniform input distribution shall be used.

5.13.2 Test case design

Test cases shall be chosen randomly from the input domain of the component according to the input distribution.

A test case shall comprise the following:

- the input(s) to the component;
- the expected outcome of the test case.

The input distribution used for the test case suite shall also be recorded.

5.14 Other testing techniques

Any alternative techniques used shall satisfy the following criteria.

- a) The technique shall be freely available in the public domain.
- b) A source reference shall be provided.
- c) The test case design technique shall be documented in the same manner as the other test case design techniques in clause 5.
- d) Associated test measurement techniques shall be in accordance with 6.15 if technically feasible.

6 Test measurement techniques

6.1 General

In each coverage calculation, a number of coverage items may be infeasible. A coverage item shall be defined to be infeasible if it can be demonstrated to be not executable. The coverage calculation shall be defined as either counting or discounting infeasible items; this choice shall be documented in the test plan. If a coverage item is discounted, justification for its infeasibility shall be documented in the test records.

In each coverage calculation, if there are no coverage items in the component under test, 100 % coverage shall be defined as being achieved by one test case.

6.2 Equivalence partition coverage

6.2.1 Coverage items

Coverage items shall be the partitions described by the model (see 5.1.1).

6.2.2 Coverage calculation

Coverage shall be calculated as follows:

$$C_{ep} = \frac{P_c}{P_t} \times 100 \%$$

where:

- C_{ep} is the equivalence partition coverage;
- P_c is the number of covered partitions;
- P_t is the total number of partitions.

6.3 Boundary value coverage

6.3.1 Coverage items

The coverage items shall be the boundaries of partitions described by the model (see 5.2.1).

NOTE Some partitions may not have an identified boundary, for example, if a numerical partition has a lower but not an upper bound.

6.3.2 Coverage calculation

Coverage shall be calculated as follows:

$$C_{bo} = \frac{B_e}{B_t} \times 100 \%$$

where:

- C_{bo} is the boundary value coverage;
- B_e is the number of distinct boundary values executed;
- B_t is the total number of boundary values;

and where a boundary value corresponds to a test case on a boundary or an incremental distance either side of it (see 5.2.2).

6.4 State transition coverage

6.4.1 Coverage items

Coverage items shall be sequences of one or more transitions between states on the model (see 5.3.1).

6.4.2 Coverage calculation

For single transitions, the coverage measure shall be the percentage of all valid transitions exercised during test (known as 0-switch coverage). For n transitions, the coverage measure shall be the percentage of all valid sequences of n transitions exercised during test [known as $(n - 1)$ switch coverage].

6.5 Cause-effect coverage

6.5.1 Coverage items

Coverage items shall be rules, where each rule represents a unique possible combination of inputs to the component that have been expressed as Booleans.

6.5.2 Coverage calculation

Coverage shall be calculated as follows:

$$C_{c/e} = \frac{R_e}{R_t} \times 100 \%$$

where:

- $C_{c/e}$ is the cause/effect coverage;
- R_e is the number of rules exercised;
- R_t is the total number of rules.

6.6 Syntax coverage

No coverage measure is defined for syntax testing.

6.7 Statement coverage

6.7.1 Coverage items

Coverage items shall be executable statements in the source code.

6.7.2 Coverage calculation

Coverage shall be calculated as follows:

$$C_s = \frac{S_e}{S_t} \times 100 \%$$

where:

- C_s is the statement coverage;
- S_e is the number of executable statements executed;
- S_t is the total number of executable statements.

6.8 Branch and decision coverage

6.8.1 Branch coverage items

A branch shall be:

- a conditional transfer of control from any statement to any other statement in the component; or
- an unconditional transfer of control from any statement to any other statement in the component except the next statement; or
- when a component has more than one entry point, a transfer of control to an entry point of the component.

An entry point shall be either the first statement of the component or any other statement which may be branched to from outside the component.

6.8.2 Branch coverage calculation

Coverage shall be calculated as follows:

$$C_b = \frac{F_e}{F_t} \times 100 \%$$

where:

- C_b is the branch coverage;
- F_e is the number of executed branches;
- F_t is the total number of branches.

6.8.3 Decision coverage items

Decision coverage shall use the model of the component specified for branch testing in 5.7.1. Coverage items shall be decision outcomes.

NOTE Decision coverage is defined only for components with one entry point.

6.8.4 Decision coverage calculation

Coverage shall be calculated as follows:

$$C_d = \frac{D_e}{D_t} \times 100 \%$$

where:

- C_d is the decision coverage;
- D_e is the number of executed decision outcomes;
- D_t is the total number of decision outcomes.

6.9 Data flow coverage**6.9.1 Coverage items**

The coverage items shall be the control flow sub-paths from a variable definition to the variable's corresponding p-uses, c-uses, or their combination.

6.9.2 Coverage calculation**6.9.2.1 General**

For the purposes of the coverage calculations given in **6.9.2.2** to **6.9.2.6**, a definition-use pair shall be defined as a *simple sub-path* between a definition of a variable and a use of that variable. Coverage shall be calculated using the formula:

$$\text{Coverage} = (N/T) \times 100 \%$$

where:

- N and T are as defined in **6.9.2.2** to **6.9.2.6**.

NOTE A *simple sub-path* is a sub-path through a component's control flow graph where no parts of the sub-path are visited more than necessary.

6.9.2.2 All-definitions

This measure shall be defined with respect to the traversal of the set of sub-paths from each variable definition to some use (either p-use or c-use) of that definition.

For the purposes of the formula given in **6.9.2.1**, the following shall apply:

- N is the number of exercised definition-use pairs from distinct variable definitions;
- T is the number of definition-use pairs.

6.9.2.3 All-c-uses

This measure shall be defined with respect to the traversal of the set of sub-paths from each variable definition to every c-use of that definition.

For the purposes of the formula given in **6.9.2.1**, the following shall apply:

- N is the number of exercised definition-c-use pairs;
- T is the number of definition-c-use pairs.

6.9.2.4 All-p-uses

This measure shall be defined with respect to the traversal of the set of sub-paths from each variable definition to every p-use of that definition.

For the purposes of the formula given in **6.9.2.1**, the following shall apply:

- N is the number of exercised definition-p-use pairs;
- T is the number of definition-p-use pairs.

6.9.2.5 All-uses

This measure shall be defined with respect to the traversal of the set of sub-paths from each variable definition to every use (both p-use and c-use) of that definition.

For the purposes of the formula given in **6.9.2.1**, the following shall apply:

- N is the number of exercised definition-use pairs;
- T is the number of definition-use pairs.

6.9.2.6 All-du-paths

This measure shall be defined with respect to the traversal of the set of sub-paths from each variable definition to every use (both p-use and c-use) of that definition.

For the purposes of the formula given in **6.9.2.1**, the following shall apply:

- N is the number of exercised simple sub-paths between definition-use pairs;
- T is the number of simple sub-paths between definition-use pairs.

6.10 Branch condition coverage**6.10.1 Coverage items**

Branch condition coverage shall use a model of the component specified in **5.9.1**. Coverage items shall be Boolean operand values within decision conditions.

NOTE Branch condition coverage is defined only for components with one entry point.

6.10.2 Coverage calculation

Coverage shall be calculated as follows:

$$C_{br} = \frac{G_e}{G_t} \times 100 \%$$

where:

- C_{br} is the branch condition coverage;
- G_e is the number of Boolean operand values executed;
- G_t is the total number of Boolean operand values.

6.11 Branch condition combination coverage

6.11.1 Coverage items

Branch condition combination coverage shall use a model of the component described in 5.10.1. Coverage items shall be unique combinations of the set of Boolean operand values within each decision condition.

NOTE Branch condition combination coverage is defined only for components with one entry point.

6.11.2 Coverage calculation

Coverage shall be calculated as follows:

$$C_c = \frac{J_{oe}}{J_{ot}} \times 100 \%$$

where:

- C_c is the branch condition combination coverage;
- J_{oe} is the number of Boolean operand value combinations executed;
- J_{ot} is the total number of Boolean operand value combinations.

6.12 Modified condition decision coverage

6.12.1 Coverage items

Modified condition decision coverage shall use a model of the component specified in 5.11.1. Coverage items shall be Boolean operand values within decision conditions.

NOTE Modified condition decision coverage is defined only for components with one entry point.

6.12.2 Coverage calculation

Coverage shall be calculated as follows:

$$C_m = \frac{J_i}{J_t} \times 100 \%$$

where:

- C_m is the modified condition decision coverage;
- J_i is the number of Boolean operand values shown to independently affect the decision outcome;
- J_t is the total number of Boolean operand values.

6.13 LCSAJ coverage

6.13.1 Coverage items

Coverage items shall be LCSAJs for the component (see 5.12.1).

6.13.2 Coverage calculation

Coverage shall be calculated as follows:

$$C_L = \frac{L_e}{L_t} \times 100 \%$$

where:

- C_L is the LCSAJ coverage;
- L_e is the number of executed LCSAJs;
- L_t is the total number of LCSAJs.

6.14 Random testing

No coverage measure is defined for random testing.

6.15 Other test measurement techniques

Any alternative techniques used shall satisfy the following criteria.

- a) The technique shall be freely available in the public domain.
- b) A source reference shall be provided.
- c) The test measurement technique shall be documented in the same manner as the other test measurement techniques specified in clause 6.

Annex A (informative)

Process guidelines

A.1 General

This annex gives guidance on the requirements specified in clause 4 of this standard.

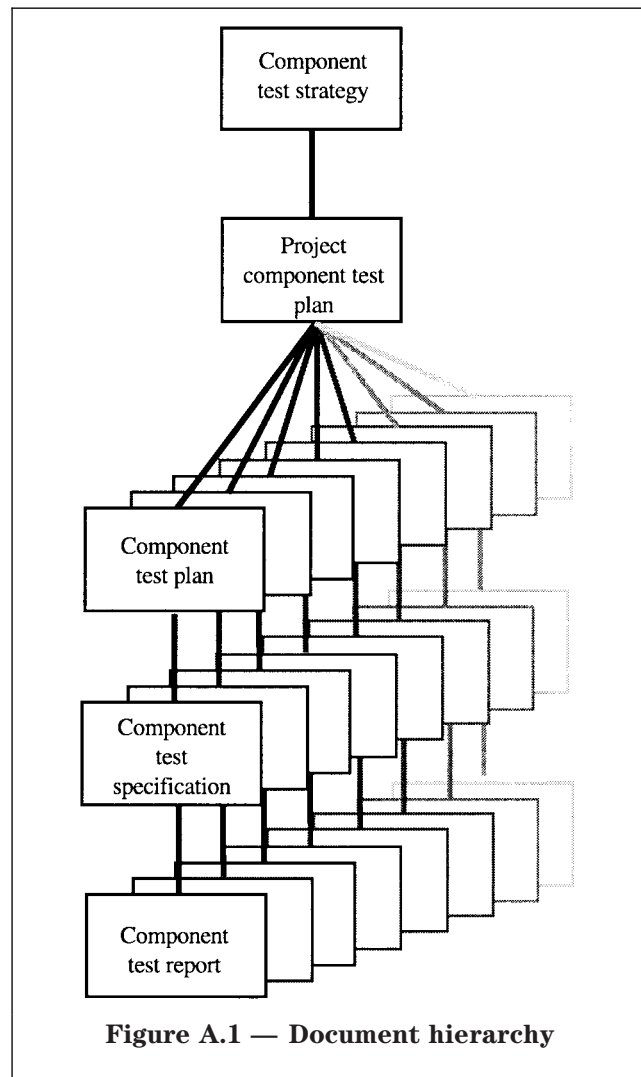
These guidelines are supported by the use of example documentation from a fictitious project, named EX. The example documentation provides a basic sample of the full set of documentation which conforms to this standard. The technical content of the example documentation merely serves to illustrate one interpretation of this standard and does not suggest that this interpretation is to be preferred over any other. Any set of documentation which meets the requirements of clause 4 of this standard is acceptable.

The test documentation examples included are:

- component test strategy;
- project component test plan;
- component test plan;
- component test specification;
- component test report.

The relationships between the different forms of example test documentation can be seen in the document hierarchy illustrated in Figure A.1.

The document hierarchy shows that there is a single component test strategy for project EX (in this example documented within the project's quality plan) and a single project component test plan for the project. For each component there is a component test plan and corresponding component test specification and component test report. Clause 4 of the standard specifies requirements for the overall test process including component testing. The prerequisites for component testing are an overall component test strategy and a project component test plan which is project specific.



A.2 Component test strategy

A.2.1 In order to conform to this standard, component testing has to be performed within a pre-defined documented strategy. Many organizations will achieve this through a corporate or divisional testing manual or quality manual. In other cases, the component test strategy could be defined for one project only and may be presented as part of a quality plan relating to a specific project.

A.2.2 Whilst there is no requirement in this standard for the component test strategy documentation to be a single document or a whole document, in the case of a component test strategy that is tailored to a specific project, the component test strategy documentation could be incorporated into the project component test planning documentation.

A.2.3 Although this standard requires that the component test strategy is defined in advance, it does not prescribe the format of component test strategy documentation.

A.2.4 As part of the test strategy, this standard defines a generic test process comprising a series of test activities and requires this sequence of activities to be followed for a particular test case. This allows, for instance, incremental approaches such as planning, executing and recording the tests for one set of components before another; or planning, executing and recording the application of one test case design technique before another.

A.2.5 Further requirements on the management of the testing are given in BS EN ISO 9001. Guidance on setting up a test process is available in ANSI/IEEE 1008.

A.2.6 The form of the test completion criterion will typically be a target for test coverage.

A.2.7 Figure A.2 gives an example of a component test strategy for project EX, developing the EX system. For the purposes of the example, the architectural design process decomposes the system into functional areas, each containing a number of components. The example component test strategy is provided as a section in the project's quality plan.

A.3 Project component test planning

A.3.1 This standard makes no requirement for the project component test plan documentation to be a single document or a whole document.

A.3.2 As a minimum, the project component test plan should identify any specific adaptations of the component test strategy and specify any dependencies between component tests.

A.3.3 Further guidance on test planning is available in ANSI/IEEE 829.

A.3.4 Figure A.3 gives an example of a project component test plan for project EX. There are no dependencies between component tests because all components are tested in isolation. The hypothetical project EX wants to begin the integration of tested components before all component testing is complete. Thus the sequence of component testing is driven by the requirements of integration testing.

A.4 Component test planning

A.4.1 This standard makes no requirement for the component test plan documentation to be a single document or a whole document.

A.4.2 The form of the test completion criterion is not prescribed. It will typically be a test coverage target which has been specified in the component test strategy.

A.4.3 Further guidance on test planning is available in ANSI/IEEE 829.

A.4.4 An example test plan for component LOG 3 of project EX is given in Figure A.4. This may be contained in the same physical document as the project component test plan for all components (see Figure A.3).

A.5 Test specification

A.5.1 The test specification for a component may be in machine-readable form; some test automation tools employ scripting languages that may meet the requirements of this standard.

A.5.2 An example test specification for component LOG 3 in project EX is given in Figure A.5. Note that the specification has been extended (version 2) to cater for gaps in the coverage of the initial specification and faults as recorded in the test report (A.7).

A.6 Test execution

Execution and coverage measurement are often performed by tools. This standard does not prescribe how such tools work.

A.7 Test report

A.7.1 In the example test report given in Figure A.6, component LOG 3 of project EX fails the first execution of tests 1 to 9. Test cases 2 and 9 fail due to faults in the test specification. These are corrected and a further test case is specified to complete decision coverage.

Project EX Quality plan
Section 6
Component test strategy

This section provides the component test strategy for Project EX. Any exceptions to this strategy (for example: omitted or additional design or measurement techniques) shall be documented in the component test plan.

Exceptions to this strategy shall be approved by the Project Manager and the Quality Manager. The use of any additional design or measurement techniques, where appropriate, is encouraged and can be done without specific approval.

Design techniques: Component tests shall be designed using the following techniques:

- Equivalence partitioning (EP, Std 3.1);
- Boundary value analysis (BVA, Std 3.2);
- Decision testing (DT, Std 3.7).

Rationale: EP and BVA have proven cost-effective on previous releases. DT shall be used to complete coverage should EP and BVA fail to meet the coverage completion criteria.

Completion criteria: A minimum of 100 % equivalence partition coverage (EPC, Std 4.1), 100 % boundary value coverage (BVC, Std 4.2) and 90 % decision coverage (DC, Std 4.7, 4.7.3 & 4.7.4) shall be achieved for each component test.

Rationale: Equivalence partition and boundary value coverage are used to ensure full application of the corresponding test case design techniques, while decision coverage provides a useful white box check on the overall set of test cases.

Independence: No independence is required in the production of test plans and test specifications, or in the implementation of tests. All test documentation including test records shall be reviewed by the member of the project to whom the originator of the work reports.

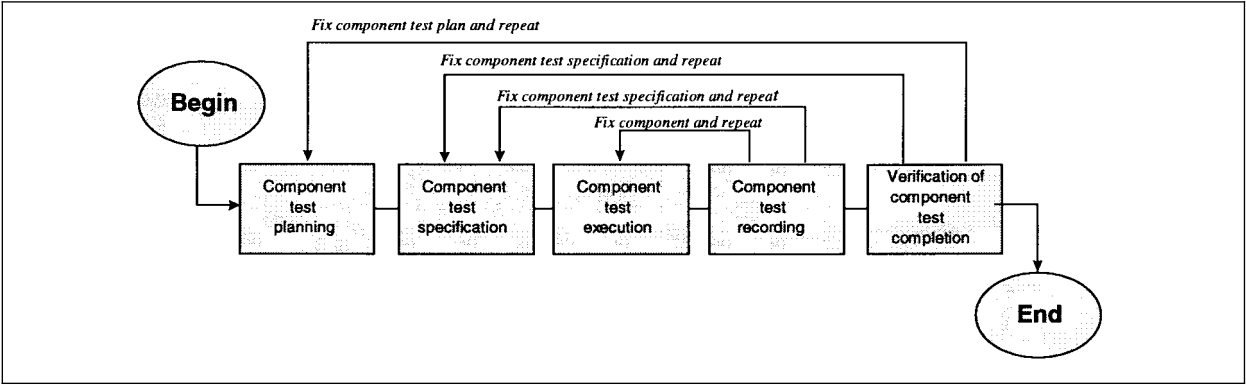
Approach: Components shall be tested in isolation using stubs and drivers in place of interfacing components.

Environment: Component tests shall be run in the developer's own workspace. All output shall be logged to a file. The coverage tool¹⁾ shall be used to analyse the test coverage of each test run.

Fault correction: Wherever possible, test cases for a component shall be executed together in a single test run. The outcome of all test cases shall then be analysed. If corrections are needed, the entire component test shall be repeated.

Project component test plan: A project component test plan shall be produced following completion of the architectural design of the EX system (Std 2.1.2).

Component test process: Items 10 to 14 below detail the component test process.



¹⁾ For the purposes of this example, a generic test coverage measurement tool has been used called "coverage tool". In practice, a test strategy would explicitly give the name of any tool used.

Figure A.2 — Example of component test strategy