Testability Analysis of Co-designed Systems

Yves Le Traon and Chantal Robach

LGI-IMAG - 46, avenue Félix Viallet - 38031 Grenoble Cedex 1- FRANCE

Abstract

This paper focus on the testability analysis of co-designed data-flow specifications. The co-designed specification level implies a high level testability analysis, independent of the implementation choices. With respect to testability, the difficulties of generating test sets, detecting and diagnosing faults are discussed and estimates are proposed. A hardware modelling, based on information transfers and called the Information Transfer Graph, is adapted to the specifications. A real case study supplied by Aérospatiale illustrates all the evaluations.

1: Introduction

Testing is an important phase in the validation and maintenance processes because of high system testing costs and since it has to guarantee the system reliability. The problem is crucial for embedded systems which are well-known to be safety-critical. During testing and, of course, maintenance phases, it is already too late to improve testability. Besides, the way by which specifications are designed influences the following testing and maintenance efforts. Consequently, it becomes essential to predict the difficulty of testing software as early as during the specification-design phase. In this paper, we focus on the testability of data flow co-designed specifications, which are encountered in most avionics systems.

Testability concepts which have been developed for hardware are appropriate to data-flow software, and more generally co-designed systems. In fact, data flow modelling is adapted to a functional analysis, for which hardware testability measurements are existing. We have thus studied the application of hardware design testability on executable co-designed data-flow specifications. We restrict ourselves to structural testing using some specific coverage strategies.

The concept of testability of a software component is introduced by Freedman [2] as a combination of controllability and observability. *Controllability* is related to the effective coverage of the declared output domain from the input domain. *Observability* is related to the undeclared variables that must be avoided. A component is then said to be domain-testable if it is both controllable and observable. Observability and controllability measures are proposed which take into account the modifications to be carried out in order to obtain an observable or controllable component. Voas and Miller [3] refined Freedman's analysis. *Testability* is then defined as "the tendency for failures to be observed during testing when faults are present" and they focused on the random black box testing strategy. They studied inputs/outputs domains of a component - the Domain/Range Ratio (DRR)- to classify components into categories with respect to testability. The classification is based on the cardinality of the inputs and outputs domains, which can be "fixed" (finite set of values) or "variable" (infinite).

Up to now, none of these works suggests satisfactory testability measurements : Freedman's measures can only be applied after software has been modified and Voas only provides an ordered classification.

On the other hand, the problem of hardware testability estimation has been tackled, and several tools are available to predict testability. Most of them, such as [4] and [5], are based on the difficulty to put and observe a given logical value (0 or 1) on the different lines of the considered circuit. Besides, other methods such as [6] and [7], are adapted to a higher description level, by modelling information transfers. This is the case of SATAN tool (System's Automatic Testability ANalysis) which is applicable to a VHDL description of the circuit [8]. The model is automatically obtained from a behavioural or a structural description of the circuit. It is a directed graph of interconnected modules. Each module corresponds to a function of the circuit and each link models the possibility to transfer information from one module to another one. The generation of this graph - the Information Transfer Graph - is more precisely detailed in [9] and [10].

The testability analysis and the functional specification of the test program are performed from the Information Transfer Graph by SATAN tool. This process is divided into three main steps :

- the search of elementary functions of the circuit (called information flows),
- the determination of a test program in terms of a sequence of information flows, by means of a test strategy,
- the measurement of testability for each module of the circuit with respects to various flows and the estimation of the diagnosis quality.

This paper partially presents the extension of such an analysis from hardware to co-designed systems. A case study is used to illustrate our results. Specification is

This work was supported by a grant from Aérospatiale - Centre de Recherche Louis Blériot - Grant n° 504 22455 (1993) [1]

0-8186-7129-7/95 $4.00 © 1995 IEEE
The Fourth Asian Test Symposium

provided in the form of a Computer Assisted Specification (CAS) diagram*, representing data flows. It corresponds to the most common description mode for the designers in avionics. Section 2 details this model and the test strategies we focus on. Section 3 presents the testability analysis and develops the diagnosability aspects. The estimates are illustrated in Section 4 and the conclusion is given in Section 5.

2: Modelling and test strategies

The principle of modelling information transfers consists in representing control and data flow aspects on the same graph. The *Information Transfer Graph* is used to identify paths along which information is transferred.

2.1: Information Transfer Graph

The required information transfer model is represented in SATAN as a bipartite directed graph where two types of nodes are defined : the places (or modules) are associated with various computations ; the transitions represent the mode of information transfers between the places. This model contains all control points, like a conventional control graph, and also includes all relations between variables. This model is particularly suitable to data flow specification languages since it automatically includes the concept of paths between the definition and use of variables. A module corresponds to a function and the interconnection of modules models the capability of transmitting information from one module to another one.

Three modes of information transfers may be considered, as shown in Figure 1 :

- *the junction mode* : the data from the source modules S1, ..., Sn are all needed for the destination module D to be exercized ;
- *the attribution mode* : in order to be exercized, the destination module D needs an information from only one of the source modules S1, ..., Sn ;
- *the selection mode* : the information issued by the source module S is sent to only one of the destination modules D1, ..., Dn.

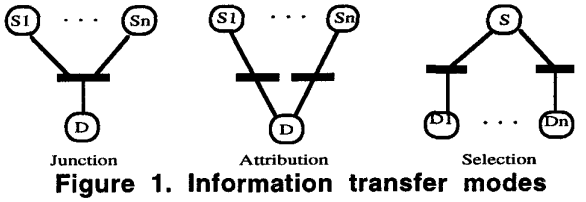

Figure 1. Information transfer modes

The notion of flow, as defined in SATAN corresponds to an information path from a set of sources toward a set of traps, which characterizes a function of the specification. The construction of the Information Transfer Graph from the Computer Assisted Specification diagram is illustrated in Figure 2 and Figure 3. The specification of which the CAS diagram is a representation is expressed in a declarative functional form as a set of equations (Figure 4). It uses elementary arithmetic operators : addition (ADD), multiplication (MULT) and division (DIV).

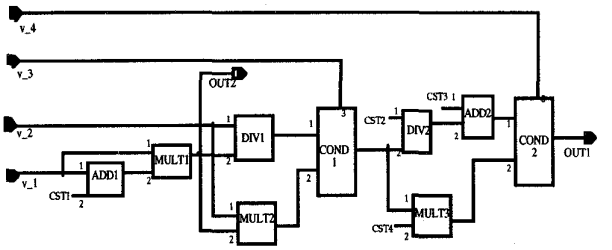

Figure 2. CAS diagram

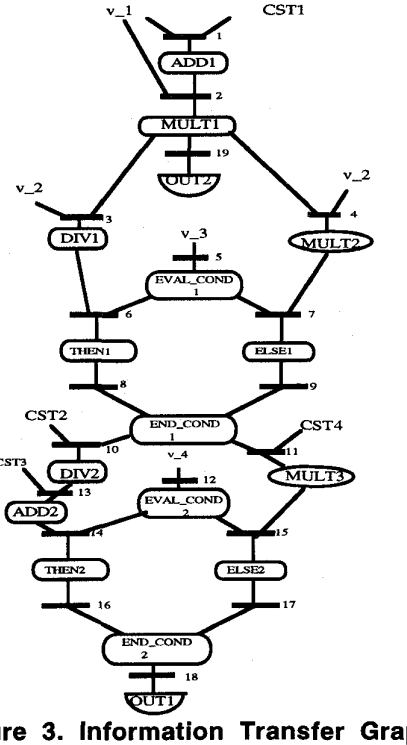

Figure 3. Information Transfer Graph

```
procedure EX(v_1, v_2 : integer; v_3, v_4 : boolean; var
OUT1, OUT2 : integer);
var
local_var : integer;
   begin
   OUT2 = MULT1(v_1, ADD1(v_1, CST1));
   local_var = if v_3 then DIV1(v_2, OUT2)
               else MULT2(v_2, OUT2);
   OUT1 =if v_4 then ADD2(CST3, DIV2(CST2, local_var))
               else MULT3(local_var, CST4);
      end;
```

Figure 4. Declarative functional form

We need to introduce some definitions before detailing the test strategies.

* The CAS language is a home-specification language of Aerospatiale similar to SADT.

	ADD1	MULT1	DIV1	MULT2	Eval -cond1	End _cond1	THEN1	ELSE1	MULT3	DIV2	ADD2	Eval _cond2	End _cond2	THEN2	ELSE2
F1	1	1	1	0	1	1	1	0	0	1	1	1	1	1	0
F2	1	1	1	0	1	1	1	0	1	0	0	1	1	0	1
F3	1	1	0	1	1	1	0	1	0	1	1	1	1	1	0
F4	1	1	0	1	1	1	0	1	1	0	0	1	1	0	1
F5	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 1. Coverage matrix

2.2: Definitions

Flow : a flow is an information path in the graph between inputs (or sources) and outputs (or traps). It is fully defined by all transitions passed through.

Coverage matrix: this is an array M[i, j] where:
- i is a flow,
- j is a module in the graph
- M[i,j] = 1 if flow i exercizes module j
  = 0 otherwise.

Indistinguishable modules : two modules are indistinguishable if they are exercized together by the same subset of flows, in other words if they have the same column in the coverage matrix.

Indiscernability sets: A set of indistinguishable modules is denoted indiscernability set.

The example of Figure 2. contains five flows. Flows F1 to F4 are related to output OUT1 while flow F5 uses output OUT2. Flows Fi are represented as a list of transitions together with the associated output :
F1 = (1, 2, 3, 5, 6, 8, 10, 12, 13, 14, 16, 18 ; OUT1)
F2 = (1, 2, 3, 5, 6, 8, 11, 12, 15, 17, 18 ; OUT1)
F3 = (1, 2, 4, 5, 7, 9, 10, 12, 13, 14, 16, 18 ; OUT1)
F4 = (1, 2, 4, 5, 7, 9, 11, 12, 15, 17, 18 ; OUT1)
F5 = (1, 2 , 19 ; OUT2)

The coverage matrix is given in Table 1. ADD1 and MULT1 columns being identical, this implies that ADD1 and MULT1 are indistinguishable modules. The other basic indiscernability sets are :
S1 = (ADD1, MULT1)
S2 = (EVAL_COND1, END_COND1, EVAL_COND2, END_COND2)
S3 = (DIV1, THEN1)
S4 = (MULT2, ELSE1)
S5 = (DIV2, ADD2, THEN2)
S6 = (MULT3, ELSE2)

Reduced coverage matrix : the reduced coverage matrix is deduced from the coverage matrix by associating only one column to all the elements of an indiscernability set.

Table 2 presents the reduced coverage matrix.

Flow length : it is defined as the number of modules exercized by a flow (i.e. the number of modules contained in the flow).

Indiscernability size : it corresponds to the number of modules contained in an indiscernability set.

	S1	S2	S3	S4	S5	S6
F1	1	1	1	0	1	0
F2	1	1	1	0	0	1
F3	1	1	0	1	1	0
F4	1	1	0	1	0	1
F5	1	0	0	0	0	0

Table 2. Reduced coverage matrix

2.3: Test strategies

Flows are generated automatically from the Information Transfer Graph. Each flow is associated to effective test data. Hence, a set of chosen flows directly determines a test strategy. Besides, coverage criteria are commonly used in software to characterize the type of testing. For example, an *all-paths* criterion implies to exercize all the paths, or all the flows in the graph. The all-paths criterion is expected to be a more complete testing than an *all-nodes* citerion which covers each node (each module) at least once. Other intermediate criteria are defined in the literature [11] but we restrict ourselves to the two mentioned above.

In this study, we assume that the test method is applied by cross-checking. Indeed, we globally analyze the results of test sets before locating and repairing faults. Hence and in order to locate the module containing the fault, we analyze all faults detected from a set of flows. The opposite assumption, that we will not discuss in this paper, concerns the incremental repairing of faulty modules [12].

In the remaining of the paper, we consider three strategies. Two of them are based on an *all-nodes* criterion : the Multiple-Clue and the Start-Big strategy. The Start-Big strategy uses a minimum set of flows to cover each module of the graph. The third strategy corresponds to the *all-paths* criterion and is called All-paths in this paper.

For the considered Figure 2 example, the Start-Big strategy is performed by the set (F2, F3) which is sufficient to cover all the indiscernability sets. However the All-paths strategy implies the application of the set (F1, F2, F3, F4, F5).

The *Multiple-Clue* strategy proceeds as follows : a set of flows is exercized, the results are stored (regardless of whether or not a fault is detected), and globally analyzed. Faulty modules are located by cross-checking. The principle is illustrated in Figure 5. If exercizing test T1

(covering sets A and B) detects a fault whereas test T2 (covering sets B and C) does not detect it, then this fault can be located in set A.

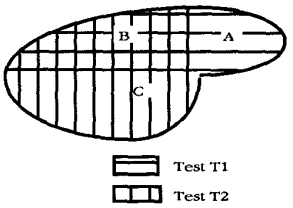

Figure 5. Multiple Clue analysis

If we use 1 to denote detection of a fault (otherwise 0) on the activation of the set of flows (F1, F2, F3, F4, F5), we obtain a fault vector. Thus, if the fault vector obtained is (1, 0, 1, 0, 0) and if we compare this vector with the columns in the corresponding coverage matrix, we can note that this vector corresponds to the third column (S5 set). A fault can thus be located in the S5 indiscernability set. However, we have no means of locating the fault within group S5, since the modules are indistinguishable. Such an analysis of the coverage matrix is used in Multiple-Clue to choose an optimal set of flows that allows to locate a fault as precisely as possible with respect to the basic indiscernability sets.

The Multiple-Clue strategy is only applicable for the case of a single fault, since otherwise the analysis becomes impossible. Therefore, it is suitable particularly for maintenance purposes, where it can be reasonably assumed that the software (and particularly embedded software) only has a very small number of faults.

The main steps for the Multiple-Clue strategy are detailed in the next section.

2.4: Multiple-Clue strategy

The coverage matrix can be used directly to choose a set of flows that cover all resources (i.e. all modules). However, if a fault is detected, the coverage matrix is not sufficient to precisely choose the flows that will distinguish the faulty module. In order to set up a fault diagnosis, another matrix (called the differential matrix D) has to be set up expressing distinguishability relations between modules for each flow.

The differential matrix D contains the same number of rows as the coverage matrix C. It uses the same flows. The number of columns in this matrix is equal to the number of distinct comparisons between modules. If there are m modules, there will be m(m-1)/2 columns in D, since all distinct (m_i m_j) pairs have to be compared. Table 3 illustrates how to build up the matrix given the two distinct modules m_i and m_j.

Note that the value of m_{ij} for flow F_k in matrix D is equal to 1 only when the values of m_i and m_j for flow F_k are distinct.

It is preferable to process the choice of flows from the *C_D* matrix composed of columns in both the C and the D matrices. This allows to directly generate diagnosis sets (detection and location).

	.. m_i ..	.. m_j ..	m_{ij}
F1	0	0	0
F2	1	0	1
F3	0	1	1
F4	1	1	0

Table 3. The differential matrix

Starting from the *C_D* matrix, a conventional boolean coverage and minimisation algorithm can be applied [13] which will determine all possible choices of flows for optimum fault diagnosis. In fact, the columns of the differential matrix must be considered as modules for which coverage must be done. Flows can then detect a fault by cross-checking with optimum diagnosis resolution. Since these algorithms are too costly when the *C_D* matrix is very large, optimized algorithms can be found for the solution of these matrices in [14]. The result of this minimisation will give a choice between several sets of flows that can give the same diagnosis quality.

The Multiple-Clue choice of flows for the previously given Figure 2. CAS diagram is : (F1, F2, F4, F5). The only flow which is not used in that case is F3, because it is not necessary to improve diagnosis resolution.

3: Significant testability factors

Testability is concerned with three characteristics : generation of a test set, interpretation of the results and fault repair. The repair characteristics depend first on the diagnosis of detected faults since faults must be located before being corrected. We limit the study of the repair characteristics to the fault location.

Therefore, as a first approximation in measuring the difficulty of testing software, we consider that:

- the number of flows needed for a given test strategy increases the size of the corresponding test set,
- as the flow length increases, the effort required to generate and interpret the corresponding test will also increase,
- the difficulty of locating faults increases with the number of indistinguishable modules.

The first point is justified very simply by the fact that there is a test set that has to be generated to exercize a given flow. The number of test sets to be generated will increase with the number of flows.

The second point is justified :

- firstly by the difficulty of determining data which activate control points that will select a flow;
- secondly by the difficulty of analyzing results when input data have been subjected to a large amount of processing.

For sake of simplicity, the generation and interpretation efforts are considered as undifferentiated in this paper. In fact, different weights should be applied to

the modules involved in the test strategy depending on whether we are talking about the difficulty of generation or interpretation.

3.1: Evaluation of the testing effort

One testability estimate may be specified as the effort of the test for a given strategy.

The generation and interpretation effort for a given flow may be given in first approximation as a linear function of the flow length. So, for a given flow i, the Flow Generation/Interpretation Effort (FGIE) is defined as follows:

$$FGIE(i) = \sum_{j \in \text{modules}} m_j$$

where mj = weight of module j in flow i

Note : the generation/interpretation effort could increase quadratically or exponentially without questioning the FGIE estimate.

As we already said, all modules are considered as having the same weight. The global difficulty in implementing the test can then be estimated as the sum of the FGIEs for each flow. Therefore the total Generation/Interpretation effort will be defined as being equal to *GIE*, as defined below:

$$GIE(S) = \sum FGIE(i)$$

$i \in F_S$ where F_S is the set of flows induced by the chosen strategy S

It can therefore be applied to estimate the test efforts of different test strategies, as it will be illustrated in our case study.

As a first example, let us consider the All-paths, Start-Big and Multiple-Clue strategies already studied on the CAS diagram of Figure 2. We can estimate their Generation/Interpretation Efforts as follows :

GIE(All-paths) = FGIE(F1) + FGIE(F2) + FGIE(F3) + FGIE(F4)+FGIE(F5) = 11+10+11+10+2 = 44

GIE(Start-Big) = FGIE(F2) + FGIE(F3) = 10+11 = 21

GIE(Mult-Clue) = 33

This example shows that the Start-Big strategy is easier to generate and interpret in terms of test efforts.

3.2: Inherent diagnosability

We focus now on the third aspect of testability, the diagnosability. *Diagnosability* is defined as the property of a system of allowing an easy location of faults. The estimate we propose is based on the analysis of the flows set used in a test strategy.

3.2.1: Intrinsic diagnosis effort

The Multiple-Clue strategy that consists in determining a minimum set of flows for a maximum diagnosis resolution brings up the intrinsic difficulty that will be encountered in locating faults. The effort will increase together with the number of cross-checkings to be performed to locate the fault. This means that the number of flows to be applied for the Multiple-Clue strategy is an immediate clue to the test effort for having a diagnosis. Independently of the effective application of the Multiple-Clue test strategy, the structure of the graph is analyzed and suggests criteria for an initial estimate of co-designed systems diagnosability.

We will denote as *Diag* the number of flows needed to obtain an optimum level of fault diagnosis resolution. *Diag* is equal to the number of flows induced by the Multiple-Clue strategy. As *Diag* increases, the cross-checkings needed to locate faults become more complex. The *Diag* criterion can be useful in maintenance purposes since the components which have the worse inherent diagnosability can be isolated.

However, the *Diag* criterion may not be a sufficiently precise criterion to compare two specifications when observation points are added in order to improve diagnosability. The *Diag* number may not be modified after modification but the diagnosability will be changed because intermediate values can be observed. The diagnosability effort must be taken into account, which is defined as follows :

DiagEff = GIE(Multiple-Clue)

DiagEff allows to decide between several versions or improvements of a software which version has the best diagnosability in terms of diagnosis effort.

3.2.2: Diagnosis quality factor

The diagnosis quality must be taken into account in an estimate of diagnosability since a specification of a component may be more difficult to diagnose but can also allow a more precise diagnosis. The addition of observation points will often allow a better diagnosis quality but it will imply an increasing testing effort. Consequently, the improvement of diagnosability is often a compromise between the improvement of the diagnosis quality factor and the increasing diagnosis effort.

We assume that a detected fault has an equivalent probability to occur for each module of the graph. Consequently, the quality factor depends on the size of the indiscernability sets in which the detected fault may be hidden. A co-designed specification in which each component can be isolated has a better quality factor than another in which all components are indistinguishable.

Thus a *degree i Diagnosis quality factor* (Diag-Quality) is defined as the probability to have a detected fault hidden in an indiscernability set of size i :

Diag-Quality(i) = (i x $n\text{-sets}_i$)/ n

with $n\text{-sets}_i$ = number of indiscernability sets of size i

n = total number of modules

3.2.3: Comparison of the strategies

In order to compare diagnosis resolutions of different strategies, let us take the CAS diagram of Figure 2. The Multiple-Clue, the All-paths and the Start-Big strategies imply the following generation/interpretation efforts :

	Multiple-Clue	Start-Big	All-paths
GIE	33	21	44

The Start-Big strategy does not allow to distinguish S1 from S2, S4 from S5, S3 from S6. The (S1, S2) set has a size of 6, the (S4, S5) set has a size of 5 and the (S3, S6) set has a size of 4. So, we have the following diagnosis quality factors for the strategies :

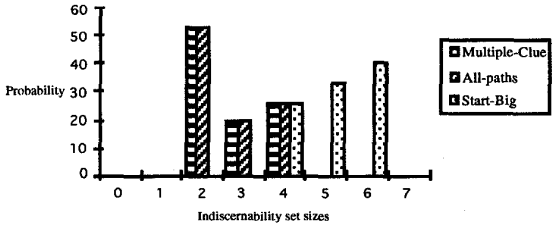

Figure 6. Diagnosis quality factors

No difference appears in Figure 6 between the All-paths and the Multiple-Clue strategies. They allow the same diagnosis resolution. In fact, this result is obvious since the Multiple-Clue strategy optimizes the choice of a diagnosis set of flows inside the set of all possible flows. Hence, the Multiple-Clue is better than All-paths since it is far less expensive in terms of GIE. The Start-Big strategy has a significantly worse diagnosis resolution. Indeed, if a fault is detected, it has a 73 % probability to be hidden in an indiscernability set of size 5 or 6, while the Multiple-Clue has the same probability to have it hidden in sets of size 2 or 3. The Start-Big is less expensive than Multiple-Clue but the diagnosis quality is worse. Other examples would show that Multiple-Clue is a good compromise between test generation/interpretation efforts and diagnosis quality, because of the optimum diagnosis obtained with a minimum number of flows.

3.2.4: Specifications comparison

As an example of specifications comparison, the Figure 2 can be considered as an improvement of the same CAS diagram which would not have the OUT2 output. We can then compare the diagnosability of the two versions, the Version-1 without OUT2 output and the Version-2 with the two outputs, for the Multiple-Clue strategy. The results are :

Diag(version-1) = 4

Diag(version-2) = 5

The diagnosis quality factors are given in Figure 7.

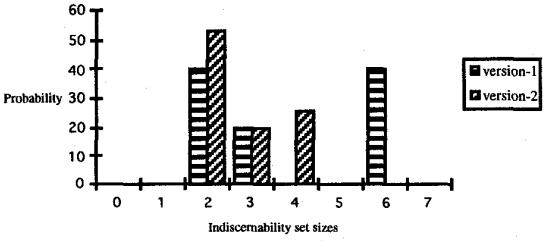

Figure 7. Specifications comparison

In terms of diagnosis quality, Version-2 is better than Version-1. Indeed, any detected fault in Version-2 will have to be located in indiscernability sets of size lower than 4. However, the Diag numbers shows that diagnosis efforts are easier for Version-1 than for Version-2. Consequently, a choice must be done between a specification allowing a better diagnosis resolution and another one easier to test.

4: Case study

An actual case study provided by Aérospatiale, presented in Figure 8, is used to illustrate our evaluations of testability. For sake of conciseness, we do not present the associated ITG.

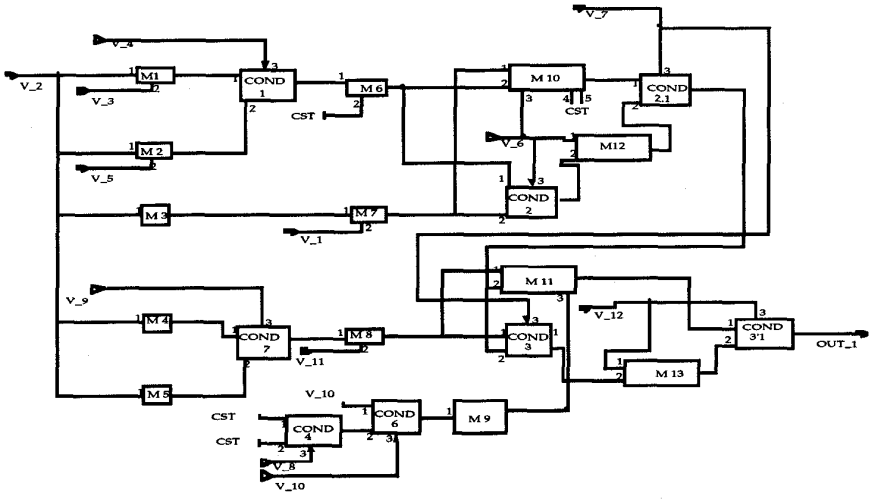

Figure 8. AÉROSPATIALE case study

As an illustration of the strategies evaluation, we compare Start-Big, Multiple-Clue and Start-Big strategies.

We have the following results :

Strategy	Start-Big	Multiple-Clue	All-paths
GIE	115	126	560

Start-Big strategy is the more efficient strategy in terms of generation and interpretation efforts. This result is consistent since this is the weakest coverage criterion.

The Multiple-Clue is a little more expensive and ensures the same all-nodes criterion. All-paths is five times more expensive which is logical since we have to perform an exhaustive coverage of the graph.

In terms of diagnosis quality factors, Figure 9 shows that All-paths and Multiple-Clue are equivalent and that Start-Big is worse. Indeed, we have for the Start-Big strategy a 50% probability to have a detected fault hidden in an indiscernability set of size 6 or more (14 % for 6, 16% for 7 and 19% for 8), while any fault for Multiple-Clue or All-paths strategies will be inside sets of sizes lower than 4.

To conclude, the Multiple-Clue seems a good compromise between the generation and interpretation efforts and the diagnosis quality it allows. With a far less expensive test effort than an All-paths strategy, it guarantees the same diagnosis resolution.

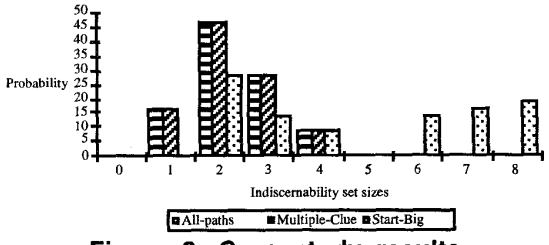

Figure 9. Case study results

5: Conclusion

Testability cannot be estimated without precisely describing what is actually estimated and based on what criteria. As a first approximation, the criterion used to evaluate the test effort is directly related to the number of flows (and the flow length) associated with this strategy. The estimating criterion for the fault repair aspect of testability has been to relate the difficulty of locating a fault in the system to the effort that has to be made by cross-checking test sets results. We complete this estimate by an evaluation of the diagnosis quality allowing to choose between an increasing diagnosis effort or a better diagnosis.

The considerations that we have discussed in this paper and the estimates that we have suggested are general and can be applied to specifications types other than data flow specifications. However, this work forms part of a more detailed study to locate the least testable parts of the specification, within the precise framework of a data flow based specification, clearly adapted to modelling in the form of information transfers.

Acknowledgements. The authors are grateful for the helpful and stimulating advice they had from Alain Cheilan from Aérospatiale.

References

[1] AEROSPATIALE, "Testabilité du logiciel", *AEROSPATIALE Technical report*, Report DCR/Q 120405.94, Chantal ROBACH & Farid OUABDESSELAM, January 1994.

[2] R. S. FREEDMAN, "Testability of Software Components", *IEEE Transactions on Software Engineering*, Vol. 17, n°6, June 1991, pp. 553-564.

[3] J. M. VOAS, K. W. MILLER, "Semantic Metrics for Software Testability", *J. Systems Software*, 1993; 20 : 207-216.

[4] L. H.GOLDSTEIN and E. L. THIGPEN, "SCOAP: Sandia Controllability / Observability Analysis Program", *IEEE Design Automation Conference*, Minneapolis, June 1980, pp. 190-196.

[5] J. GRASON, "TMEAS, a Testability Measurement Program", *16th Design Automation Conference*, San Diego, 1979, pp. 156-161.

[6] J. A. DUSSAULT, "A Testability Measure", *IEEE Semiconductor Test Conference*, Chery-Hill, October 1978, pp. 113-116.

[7] C. ROBACH, P. MALECHA, G. MICHEL, "Computer-Aided testability evaluation and test generation", *IEEE International Test Conference*, Philadelphia, October 1983, pp. 338-345.

[8] P. WODEY, C. ROBACH, "Using a VHDL description to generate hardware test", *International Symposium on Computer Hardware Description Languages and their applications- CHDL 91*, Marseille, 22-24 April 1991, pp.413-431.

[9] C. ROBACH, P. MALECHA, C. MICHEL, "CATA : A Computer-Aided Test Analysis System", *IEEE Design & Test of Computers*, Vol. 1, n°2, May 1984, pp. 68-79.

[10] C. ROBACH, P. WODEY, "Linking design and test tools : an implementation", *IEEE Transactions on Industrial Electronics*, Vol. 36, n°2, May 1989, pp. 286-295.

[11] S. RAPPS, E. J. WEYUKER, "Selecting Software Test Data Using Data Flow Information", *IEEE Transactions on Software Engineering*, Vol. SE-11, N°4, Avril 1985, pp.367-375.

[12] Y. LE TRAON and C. ROBACH, "From hardware to software testability", *IEEE International Test Conference*, Washington, October 23-25, 1995.

[13] C. V. RAMAMOORTHY, L. C. CHANG, "System modeling and testing procedures for microdiagnostics", *IEEE Transaction's on Computers*, Vol. C-21, n°11, November 1972, pp. 1169-1183.

[14] ROBACH C., LUTOFF D, GARCIA N., "Knowledge-based functional specification of test maintenance programs", *IEEE Transactions on CAD/ICAS*, Vol. 8, n°11, November 1989, pp. 1145-1156.