Relying on Testability Concepts to ease Validation and Verification activities of AIRBUS Systems

Fassely Doumbia, Odile Laurent
Systems Design
AIRBUS France
Toulouse, France
{Fassely.Doumbia, Odile.Laurent}@airbus.com

Chantal Robach, Michel Delaunay
Systems Design & Test
LCIS – Grenoble Institute of Technology
Valence, France
Chantal.Robach@esisar.grenoble-inp.fr

***Abstract*—The experiments, carried on AIRBUS systems, show that testability analysis can ease system formal detailed specifications validation activities. Indeed, testability information can highlight testing efforts, guide functional tests definition, facilitate detailed specification coverage analysis against system requirements, and support tests coverage analysis against formal detailed specification. This paper highlights the assessment result of testability concepts on AIRBUS systems.**

Keywords- requirement; data-flow design; testability flows; testability measures; testing strategy; test; coverage analysis

I. INTRODUCTION

Development of avionics systems must comply with the DO-178B standard [17]. The Validation and Verification (V&V) process is very demanding and contributes to high development costs. Therefore, innovative methods and tools that can alleviate and efficiently support V&V activities are of great interest for aeronautics domain. Functional testing is the most commonly used technique for systems requirements V&V. But, testing methods present some limits: exhaustive test data generation is most often unselected because of the size and the complexity of the systems. In this way, controlling testing effort is major, but systems quality demands are so big. Indeed, testing effort characterizes much as test scenarios and test data definition as error identification after fault detection during the diagnosis.

In this context, the testability analysis methodology proposed in this paper can offer useful methods to support the validation of systems formal detailed specification. Our testability approach deals with detailed specification relying on data-flow languages like SCADE [5]. The testability analysis proposed is based on SATAN [3, 7] (System's Automatic Testability ANalysis) technology. This approach determines testability flows and metrics that can be helpful information for system designers and system design validation engineers.

A significant number of research projects have dealt with software testability and a set of complexity metrics have been proposed. Each of these metrics address to either black-box or white-box testing technique. The black-box testing consists in considering the system under test as a box of which we know only the specification. Test data are generated from this specification which is most often formalized. The black box testing does not consider the internal structure of the system and defined test data are independent of the implementation. Freedman [6] and Voas and Miller [11] defined some testability approaches related to the black box testing technique. Freedman introduced the domain testability of software components based on controllability and observability. A component becomes observable when it produces distinct outputs from different inputs. A component is controllable when its specified output field corresponds to its produced output. Voas and Miller proposed the DRR (Domain / Range Ratio) metric, which exhibits fault-hiding tendencies of software subcomponents considering the input and the output field cardinality. This technique can be used to predict a subcomponent's ability to cause program failure if it contains a fault.

The white box testing technique is based on the internal structure of the system under test to define test data and coverage criteria. This technique assumes the program under test is available as well as the system specification. McCabe [8], Nejmeh [9] and Do, Le Traon and Robach [2, 3, 7] proposed some approaches based on the internal structure of a program. McCabe defined the Cyclomatic number which measures the number of linearly independent paths through the control graph built by using a program source code. Nejmeh introduced the Npath metric which computes the number of possible execution paths through a function. Do, Le Traon and Robach proposed a testability measurement applicable to data-flow designs. The study presented in this paper is based on the same approach.

This paper proceeds as follows: Section II introduces SATAN technology and the associated testability analysis concepts. Section III describes the AIRBUS flight control systems validation process. Section IV proposes some adaptation techniques for Airbus systems analysis. The defined methodology which aims at enhancing systems validation process is presented in Section V. Experiments results are depicted in Section VI. Finally, conclusion and perspectives are given in section VII.

II. TESTABILITY ANALYSIS

Fig. 1 gives a simple view of testability analysis, proposed by SATAN technology. This approach has been initially applied to hardware systems [3]. Further studies [2, 7] demonstrated that this approach could be used for analyzing the testability of data-flow designs. This method

provides metrics that allow the identification of critical parts (which can be hard to test). It also identifies testability flows which are a set of operators involved in the computation of one output from relevant inputs. They can be used to guide tests definition in the system design validation cycle.

The SATAN approach is based on a testability model that represents the transfer of information into the system. This model is called *ITM* "Information Transfer Model" (Section A). Testability flows are identified from this ITM (Section B). Measures quantified the testability of system components are calculated from these testability flows (Section C). Test strategies (Section D) can be applied to select relevant testability flows to help the tests generation process.

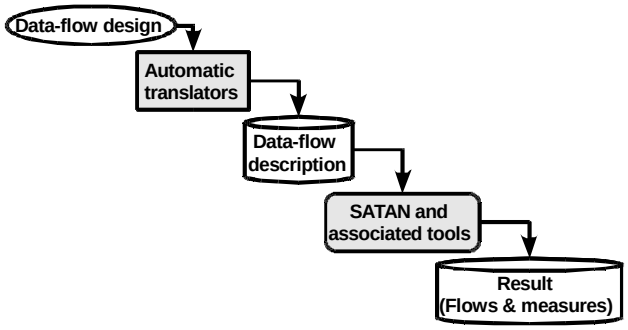

Figure 1. Testability analysis process.

A. Information Transfer Model

The testability model is a graph defined by a set of places, transitions and arrows. Places represent inputs, constants, functional modules, outputs and test injection points specified in the system. Transitions express information transfer modes between places. Arrows connecting places and transitions represent information media throughout the system.

Three different information transfer modes Fig. 2 are used in an ITM:

- Junction mode: the destination place needs information from all source places;
- Attribution mode: the destination place needs information from one of several source places;
- Selection mode: the same information is sent from the source place to some destination places.

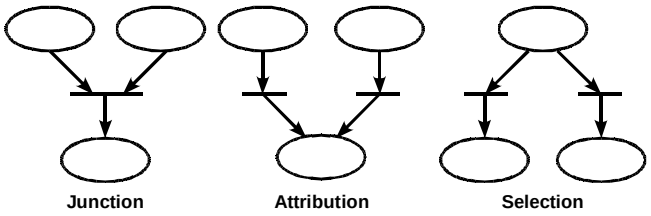

Figure 2. Information transfer modes.

A data-flow description of a system is hierarchically composed of operators. Each operator has an elementary ITM. This model corresponds to the data-flow representation of the operator. According to the level of testability analysis, this elementary model can be more (or less) detailed. The basic representation of an operator associates a functional module to each output. Graphic representations Fig. 3 illustrate the notion of elementary ITM.

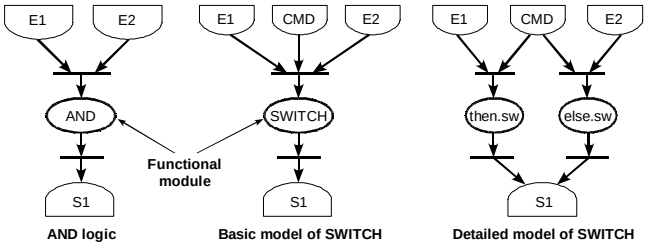

Figure 3. Elementary ITM of AND logic and SWITCH operators.

In this representation, inputs and outputs are depicted by semicircles, modules by circles, and transitions by bars.

The system ITM is obtained by concatenating the elementary ITMs.

B. Testability flows

SATAN technology identifies flows from a system ITM. A testability flow is an information path that carries information from one or several inputs, through modules and transitions, to one output. Several testability flows can be associated to an output.

The system specification represented in Fig. 4 contains sixteen testability flows. Two testability flows (F_1 and F_5) are depicted using bold lines. These flows are described below by a set of modules and output.

F_1 = {NOT_2, then.SWITCH_2, OR_2 | O2}

F_2 = {NOT_2, then.SWITCH_2, OR_2, then.SWITCH_4, OR_3, else.SWITCH_5 | O1}

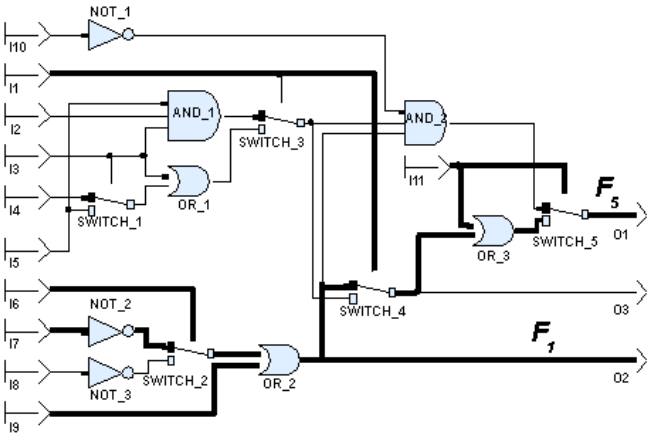

Figure 4. Graphical representation of a testability flow

C. Testability measures

Two different measures are defined to characterize the testability of a component using SATAN approach: the controllability and the observability. The controllability expresses how ease the input values of an internal component can be controlled through the input values of the system. The observability expresses how ease the results of an internal component can be observed at the outputs of the system. Fig.5 illustrates these measures.

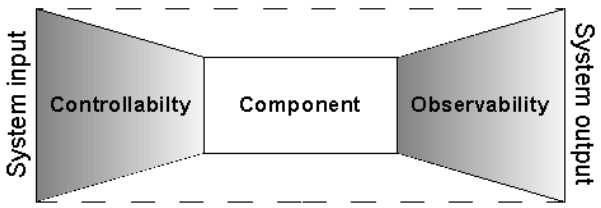

Figure 5. Controllability and observability of a component

These measures can be calculated for each defined testability flow. SATAN associates these measures to all the system ITM modules. Therefore, the testability measures of a component are deduced from modules which represent it in the ITM. Testability measures computation is based on information theory. Let F be a testability flow and M a module activated by F, we can define the following variables:

- I_F represents the sources of F;
- O_F is the output of F;
- I_M represents the input of M;
- O_M is the output of M.

The controllability of a module M is computed as follows: if M is isolated, all the possible combinations of its input values can be produced. We denote $C(I_M)$ the input capacity of M corresponding to the maximum information quantity on its inputs when it is isolated. If M belong to the testability flow F, the maximum information quantity that can be brought to its inputs is denoted $T(I_F;I_M)$. The controllability of M is expressed as follows:

$$Cont_F(M) = \frac{T(I_F;I_M)}{C(I_M)}$$

In the same way, the observability of a module M is computed as follows: the maximum information quantity produced by M is $C(O_M)$ when it is isolated. The maximum information quantity which can be delivered to the output of F from O_M is denoted $T(O_M;O_F)$. The following equation defines the observability of M:

$$Obs_F(M) = \frac{T(O_M;O_F)}{C(O_M)}$$

To determine the information quantities $T(I_F;I_M)$ and $T(O_M;O_F)$, the approach uses the Information transfer Net (ITN) to simulate the transfer of information in the system. The ITN is a weighted ITM, whose elements are associated with their information capacities. These capacities quantify the information carried by the ITM elements using bits as information unit.

- The information capacity of the ITM sources are determined from their data types. The capacity is equal to 1 for Boolean, 8 for Integer (when we reduce the domain to 2^8 elements), etc.
- The capacity of a module corresponds to the information quantity of its output. Indeed, we suppose that the output takes its value in a finite set. A probability is associated with the occurrence of each element of this set at the output. In this context, the capacity is equal to the entropy of the module output variable. The Information Loss Coefficient (ILC) of a module is determined from its capacity. It expresses the intrinsic loss of information of each ITM module. The ILC corresponds to the ratio between the effective capacity and the maximum capacity of the module [12, 13].
- The capacity of an arrow leaving from a place is equal to the capacity of that place.
- The capacity of an arrow arriving to a place is equal to the capacity of this place.
- In the junction information transfer mode, the capacity of the arrow that leaves the transition is the sum of capacities of all arrows arriving at the transition.
- In the selection mode, the capacity of each arrow leaving from the source place corresponds to the capacity of this source place.
- In the attribution mode, the capacity of the destination place is equal to the maximum value of capacities associated with arrows arriving at this place.

As to build the ITM, the ITN is obtained from elementary ITNs of system operators. “Fig 6” presents the ITM (a) and the ITN (b) of the AND operator.

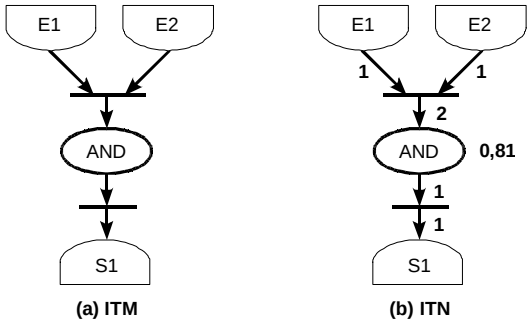

Figure 6. ITM and ITN of the operator AND logic

As previously mentioned, testability measures can be calculated for each testability flow. The approach described in this paper considers only selected flows after the application of a test strategy in order to alleviate the testability analysis process.

D. Test strategies

A test strategy defines the way to conduct test activities and to analyze test results. Test strategies allow us to select a set of relevant testability flows for testing purpose. Flows are chosen according to the following criterion: every place in the graph must be activated at least once in order to ensure the coverage of all operators. SATAN supports three strategies: *Start-Small* (progressive structural strategy) [10] suitable for the progressive detection of faults during system validation, *Multiple-Clue* (cross-checking strategy) [10] suitable for diagnosis during maintenance and *All-paths* [10] which chooses all flows contained in the ITM. We will focus on the Start-Small strategy in this paper.

The Start-Small strategy gradually covers the modules by choosing flows with an increasing number of covered modules. The main idea of this strategy is to minimize test and diagnosis efforts. The first testability flow to be tested contains the minimal number of modules. The next one contains a minimal number of modules that are not activated yet. In this strategy, a new flow is tested only if all faults detected in previous flows are corrected, as depicted in Fig. 5. Tests are defined for each selected flow.

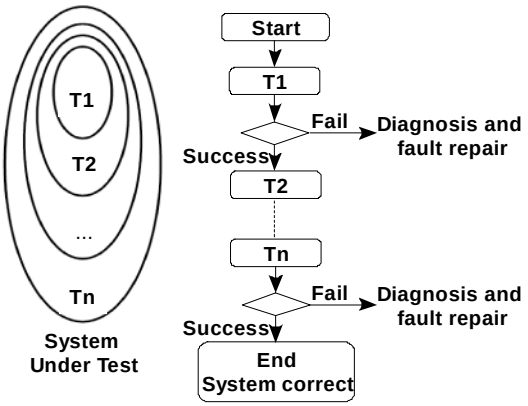

Figure 7. Start-Small strategy illustration

Start-Small strategy selects eight relevant testability flows instead of the sixteen determined for the system specification depicted in "Fig 4".

Choosing a relevant testing strategy depends on industrial practices and system development stage. In the next section, we present the AIRBUS system validation and verification process in order to point out the detailed specification validation activities.

III. AIRBUS VALIDATION AND VERIFICATION (V&V) PROCESS

In this section, we focus on AIRBUS flight control and Auto flight systems V&V activities. Flight control systems allow the pilot to control the aircraft in flight. Auto flight systems allow maintaining the flight path defined by the crew. They also control the aircraft and the engines. The V&V cycle of these reactive systems relies on a generic V-cycle process for which validation activities are added due to the modelling process at system level Fig. 6.

Three main levels can be identified in this V&V process.

- The "*Aircraft level*" is composed of aircraft level requirements definition, aircraft simulation, ground and flight tests activities.
- The "*System level*" represents the system specifications and design definition stage. The system specification validation activities are led in the phase.
- The "*Equipment level"* corresponds to the implementation of system specifications and designs in real equipment.

We will focus on systems validation activities in the rest of the paper.

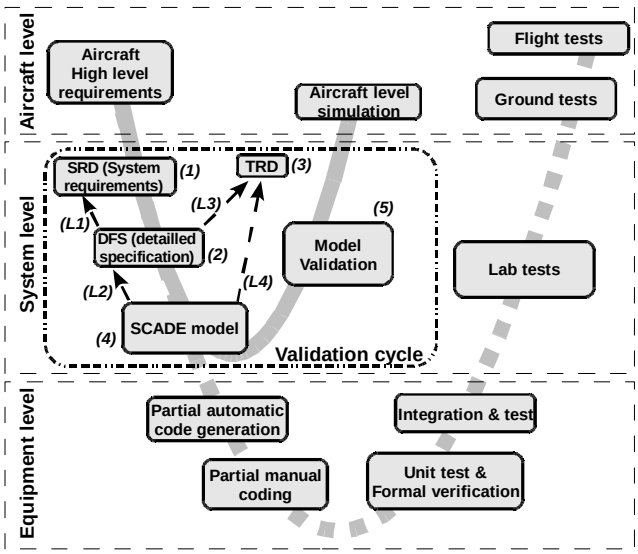

Figure 8. AIRBUS Flight control systems V&V process

A. System validation cycle

System validation activities are mainly based on testing and traceability activities. The model validation activities (5) consist in executing tests on a desktop simulator dedicated to flight control system. This simulator embeds system functions code automatically generated from the SCADE model. The traceability process relies on a documents cascade and on a SCADE model managed by the DOORS tool [5]:

- *SRD* (System Requirements Document) (1) specifying system requirements refined from aircraft requirements.
- *DFS* (Detailed Functional Specification) (2) document describing the system detailed functions that meet the system requirements.
- *TRD* (Test Requirements Document) (3) defining for each DFS function the tests data (functional tests description, tests vectors and expected tests results).
- SCADE model (or detailed specification) (4) corresponding to a formal implementation of the detailed functions and from which the embedded code is automatically generated.

The traceability activities, based on DOORS tools, consist in checking that:

- All the system requirements described in the SRD are considered in the system functions (L1);
- The SCADE model implements only once all the systems functions specified in the DFS (L2) (neither under-specification nor over-specification);
- The tests defined in TRD cover all the functions of the DFS (L3) and the functions of the SCADE model (L4) (no missing tests).

In order to alleviate drastically the coverage analysis activities described above, we propose to rely on the testability flows determined by SATAN on the SCADE model.

Indeed, if we can demonstrate that there is a clear link between testability flows and DFS requirements, the relationship between SCADE model and DFS becomes obvious. Another important issue of this approach in the validation process is the definition of relevant set of tests ensuring the coverage of the SCADE model. Enough tests have to be identified to cover all the system requirements, but redundant tests must be avoided for cost-efficiency reasons.

The next Section IV presents developed methods in order to take AIRBUS systems specific characteristics into account during testability analysis.

IV. ADAPTING THE TESTABILITY ANALYSIS TO THE AIRBUS CONTEXT

Appling the testability analysis on AIRBUS systems needs the development of new testability concepts. Indeed, systems specification use specific operators defined in the AIRBUS in-house libraries in addition to those predefined in the SCADE environment. The testability analysis must build ITM for each specific operator and integrate new approaches in order to comply with system validation functional tests definition process. We first describe the proposed technique for modeling AIRBUS specific operators ITM and ITN (Section A). Indeed, previous research works highlight the interest of testability measures on some systems [12, 15]. In this paper, we will check the relevance of these measures in the AIRBUS systems functional testing context. Then, we define testability flows classification method relying on the different phase of systems operation (Section B). Finally, we present defined testability approaches for systems validation (Section C).

A. AIRBUS specific operators ITM and ITN

The in-house libraries operators can be mainly split in two categories: combinative and temporal operators. Indeed, SCADE uses the synchronous approach for systems specification [14]. In this approach, the time is divided into discrete instants (cycles) defined by a global clock. At instant t, the system receives input i_t from its external environment, and computes output o_t. Fig. 9 illustrates the synchronous approach principles.

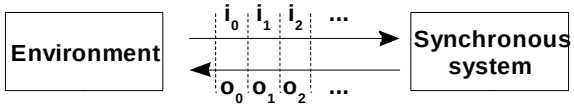

Figure 9. Synchronous mechanism

The synchronous hypothesis expresses that the computation of the output values is made instantaneously at the same instant t.

The combinative operators compute the output considering information of the current cycle. Arithmetic and logical and switch operators belong to the category of combinative operators. SATAN already proposes techniques to model these operators.

Temporal operators compute the output considering information of previous cycles (operators memory) in addition of the current one. These operators implement the past linear temporal logic. The flip-flop, state change confirmation and state change stabilizer operators belong to the category of temporal operators. This temporal aspect is not currently taken into account by the testability analysis. We use the flip-flop with priority to reset (BASCR) to describe the ITM and ITN modeling techniques in order to integrate the temporal aspect into the testability analysis proposed by SATAN.

Fig. 10 highlights the graphical representation of BASCR. Four different inputs and one output can be identified for this operator.

- E_S is the "Set" input (boolean);
- E_R is the "Reset" input (boolean);
- $Init$ represents the initialization value (boolean);
- B_Init corresponds to the initialization boolean;
- $S1$ is the output (boolean) of the operator.

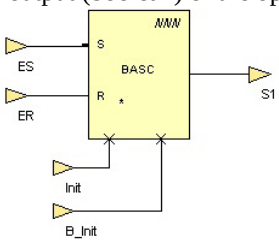

Figure 10. SCADE representation of BASCR

The algorithm highlighting the behavior of this operator is described below. We denote $E(k)$ as the value of the flot E at the cycle k.

Initialization

If $B\_Init(k) = True$ *Then* $S1(k) = Init(k)$

Else Calculation

Calculation

If $E_R(k) = True$ *Then* $S1(k) = False$

Else If $E_S(k) = True$ *Then* $S1(k) = True$

Else $S1(k) = S1(k-1)$

1) ITM of BASCR operator

The data-flow modeling of BASCR for testability analysis is represented using two modules (see Fig. 11).

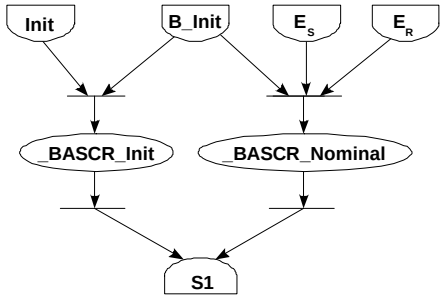

Figure 11. ITM of the temporal operator BASCR

The first module (_BASCR_Init) computes the value of $S1$ when B_Init is *True* from the input *Init* in the initialization cycle. The module "_BASCR_Nominal" computes $S1$ from E_S, E_R and the value of $S1$ in previous cycle when B_Init is *False*.

2) ITN of BASCR operator

The ITN modeling of BASCR operator returns to calculate the capacity of its modules ("_BASCR_Init" and "_BASCR_Nominal").

The basic hypothesis is to consider the occurrence of an operator inputs value as independent events from a cycle to another one. Indeed, it is difficult to make a correlation between these values according to the complexity of their production and the use of the operator in the system specification. We also suppose that these events comply with the Bernoulli distribution [16].

The "_BASCR_Init" module produces *Null* when B_Init is equal to *False or* 0 and calculates $S1$ when B_Init is equal to *True or* 1. As result, three different values (*Null*, 0 and 1) can be observed on the output of this module. Therefore, the maximum capacity of the output is equal to $Q = \log_2(3)$. We denote P_B the probability of the event $B\_Init = 1$. Let $P_{i\_m}(A)$ be the occurrence probability of A at the output of "_BASCR_Init".

$$P_{i\_m}(Null) = P(B\_Init = 0) = 1 - P_B$$

$$P_{i\_m}(0) = P(B\_Init = 1 \text{ and } Init = 0) = P_B \times P(Init = 0)$$

$$P_{i\_m}(1) = P(B\_Init = 1 \text{ and } Init = 1)$$

$$= P_B \times P(Init = 1) = P_B \times (1 - P(Init = 1))$$

The capacity of this module can be expressed as follows:

$$Capa(\_BASCR\_Init) = -\sum_{j=\{Null,0,1\}} P_{i\_m}(j) \times \log_2(P_{i\_m}(j))$$

In addition, the ILC (Information Loss Coefficient) of this module is defined in the following expression.

$$ILC(\_BASCR\_Init) = \frac{Capa(\_BASCR\_Init)}{\log_2(3)}$$

Considering the module "_BASCR_Nominal", it produces *Null* when $B\_Init = 1$ and calculates $S1$ when $B\_Init = 0$. As result, three different values (*Null*, 0 and 1) can be observed on the output of this module. Therefore, the maximum capacity of the output is equal to $Q = \log_2(3)$. Let $P_{n\_m}(A)$ be the occurrence probability of A at the output of "_BASCR_Nominal". We denote p the probability of the events $P(E_R(k) = 1)$ or $P(E_S(k) = 1)$.

$$P_{n\_m}(Null) = P(B\_Init = 1) = P_B$$

$$P_{n\_m}(1) = P(B\_Init = 0 \text{ and } S1(k) = 1)$$

$$= (1 - P_B) \times P(S1(k) = 1)$$

When $B\_Init = 0$, the probability of the event $S1(k) = 1$ can be expressed using the following table.

$E_R(k)$	$E_S(k)$	$S1(k)$
0	0	$S1(k-1)$
1	0	1
0	1	0
1	1	0

$$P(S1(k) = 1) = P(E_R = 1 \text{ and } E_S = 0)$$

$$\text{or } P(E_R = 0 \text{ and } E_S = 0 \text{ and } S1(k-1) = 1)$$

$$\Rightarrow P(S1(k) = 1) = (p \times p) + ((1-p) \times (1-p) \times P(S1(k) = 1))$$

$$P(S1(k) = 1) = \frac{1-p}{2-p}$$

Therefore, $P_{n\_m}(1) = (1 - P_B) \times \frac{1-p}{2-p}$

$$P_{n\_m}(0) = P(B\_Init = 0 \text{ and } S1(k) = 0)$$

$$= (1 - P_B) \times (1 - P(S1(k) = 1))$$

$$P_{n\_m}(0) = (1 - P_B) \times \frac{1}{2-p}$$

The capacity of this module is:

$$Capa(\_BASCR\_NoMinal) = -\sum_{j=\{Null,0,1\}} P_{n\_m}(j) \times \log_2(P_{n\_m}(j))$$

The ILC is defined in the following expression.

$$ILC(\_BASCR\_NoMinal) = \frac{Capa(\_BASCR\_NoMinal)}{\log_2(3)}$$

These capacities depend on the probabilities P_B and p. Fig. 12 presents an ITN of BASCR with $P_B = 0{,}5$ $p = 0{,}5$ and $P(Init = 1) = 0{,}5$.

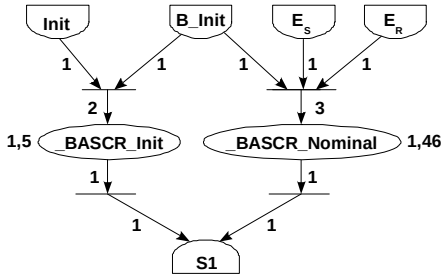

Figure 12. ITN of the temporal operator BASCR

This modeling technique can be applied to other temporal operators. It introduces new aspects (the state of memory and the initialization phase of operators) in the testability measures computing process.

B. Testability flows classification

Testability flows are the basic element of the proposed testability analysis. Indeed, they represent system elementary functions. The initialization phase modeling for temporal operators ITM leads to the identification of two different categories of testability flows. These categories reflect the normal operation of reactive systems (initialization and nominal phases).

- The initialization phase corresponds to the program memory initialization in order to ensure a

deterministic behavior. However, the memories of all the used temporal operators represent the memory of the program. In this instance, this phase returns to the initialization of these operators.

- The nominal phase represents the cyclic operation of the program. This periodic operation is composed of three main parts: the inputs reading, the outputs computing and the memory updating. In a synchronous paradigm, this period corresponds to a cycle.

In order to make the testability analysis more representative of systems operation, we define two classes of testability flows corresponding to each operation phase. The method of testability flows classification is illustrated in aid of the Fig. 13.

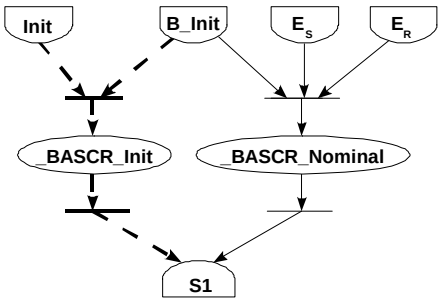

Figure 13. An ITM representing the two classes of flows

As previously described, the ITM of the BASCR operator is composed of two modules representing respectively the initialization and the nominal phases. Considering this ITM, one flow coming from "_BASCR_Init" or "_BASCR_Nominal" is required to activate *S*1. Indeed, it corresponds to the attribution mode (see Fig. 2). Therefore, the classification method consists in separating testability flows which contain initialization modules from the other testability flows. Referring to Fig. 13, the dotted line represents the flow which activates "_BASCR_Init". This technique can be applied to other temporal operators and information transfer modes. It allows the identification of testability flows (functions) which are activated in the both phases. This can be interesting for test definition issue.

C. Testability analysis approahces

Testability analysis information is useful only if it reflects the development process. In this section, we present three testability approaches defined for AIRBUS needs.

1) Output variable approach

This first approach consists in identifying all the SCADE output variables involved in the DFS. The part of the SCADE model related to each output variable is extracted. Thereby, testability flows are determined for each extracted part of the SCADE model. This approach allows a local testability view of the system for all output variables.

2) Set of output variables approach

This second approach consists in identifying the set of output variables per function. The part of the SCADE model related to this set of output variables is extracted and testability flows are determined. It allows the testability assessment for each function defined in the system.

3) Component approach

In this third approach, we propose to split the SCADE model into independent parts from a data-flow point of view (called hereafter components). This approach consists in extracting each component and performing the testability analysis. It allows the analysis of system independent parts separately.

To illustrate testability approaches, we use the system specification depicted in Fig. 4. "Tab I" below summarizes the result when applying these methods. We assume O_1 performs a function and O_2 and O_3 perform another function.

We observe that the number of selected testability flows decreases from (1) to (3). Indeed, the description of some output variables can share SCADE model parts.

- In approach (1), these common parts are analyzed for each output variable. Thereby, an important number of testability flows is determined. Fig. 14 highlights the notion of common parts. The part (A) computes the output variable O_2 and (B) performing O_3 involves (A). Using this approach, testability flows determined for (A) are also considered during flows identification for (B). (A) is then a common part of the model.
- Regarding the approach (2), common parts used by a set of output variables related to a function are analyzed once during testability analysis. Common parts related to different functions are analyzed several times. So, the number of testability flow decreases compared to (1) but is still high. The part (A) performs O_2 and O_3 output variables; (B) performing O_1 involves (A) (see Fig.14).

TABLE I. TESTABILITY APPROACHES ILLUSTRATION

	Methods		
	Output variables (1)	Set of output variables (2)	Component (3)
Number of selected flows	**12** (6 for O_1, 2 for O_2 and 4 for O_3)	**11** (6 for the *1st function* and 5 the *2nd function*)	**8** (5 for O_1, 2 for O_2 and 1 for O_3)

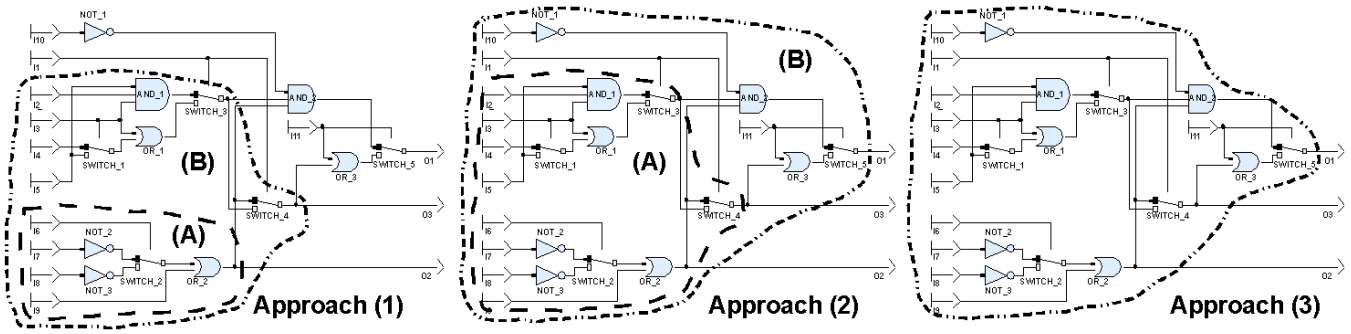

Figure 14. Specification common parts illustration

- In approach (3), all SCADE model related parts are analyzed together. Each part of the specification is explored only once during testability analysis. As a result, testability flows determined by this approach represent the lowest number compared to the two others. The specification depicted in Fig. 14 contains only one component.

We present, in this section, the main developed methods for AIRBUS systems testability analysis. The next section (Section V) defines a methodology based on testability analysis to support system coverage analysis.

V. TESTABILITY ANALYSIS METHODOLOGY

The methodology defined for AIRBUS systems is based on DFS document, TRD document and SCADE model. It is composed of three main activities: a testability approach application, the coverage assessment of the requirements against the SCADE model, and the test cases against the SCADE model.

A. Testability approach application

The testability analysis stage is depicted in Fig. 15. It consists in applying one of the three testability approaches defined in the previous section. Relevant testability flows are selected using the Start-Small strategy (Section II). These flows are exploited for coverage activities. Testability measures are determined from selected flows. The relevance of these measures is checked during experiments (Section VI).

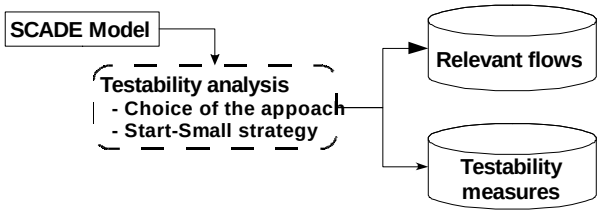

Figure 15. Testability approach application stage

B. SCADE model coverage against system requirements

Systems functional requirements have to be specified into a formal model for their validation. This coverage analysis uses relevant testability flows selected in the previous section to give information about the completeness of the formal specification. Fig. 16 presents the SCADE model coverage analysis process.

This coverage analysis is mainly composed of three activities.

a) Requirements output variables identification: This activity consists in using the DFS document to identify SCADE ouput variables involved in each system requirement definition. The identification is performed manually.

b) SCADE model coverage analysis: Testability flows are associated automatically with output variables in this step. This coverage activity allows highlighting a link between requirements and testability flows. It also points out the presence of:

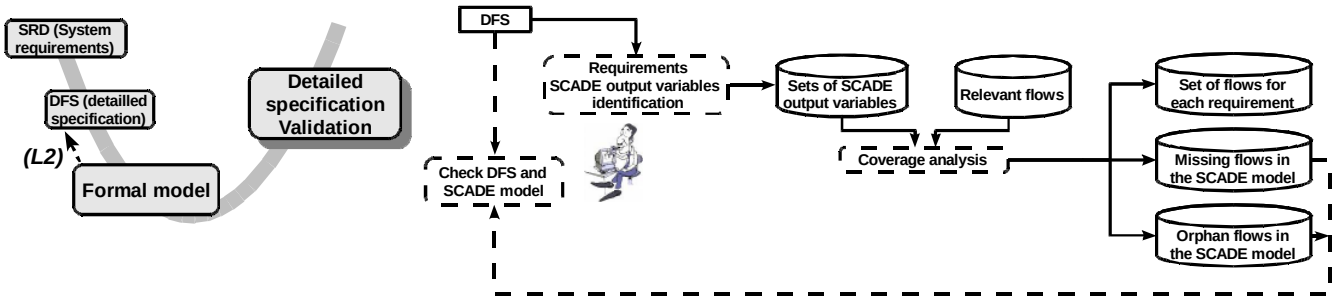

Figure 16. SCADE model coverage analysis against requirements process

- *Orphan flows*: Is called orphan a flow which has not any association with a requirement. The identification of these flows highlights the presence of SCADE output variables which are not defined in the DFS. The presence of these flows points out either a possible over-specification of the SCADE model or the implementation of derived requirements in the SCADE model.
- *Missing flows*: This situation is due to the absence of flows associated with some output variables. The detection of these outputs means that no SCADE model part corresponds to these output variables. It highlights the SCADE model is incomplete: the implementation of some DFS requirements is missing.

In addition, for each DFS requirement, the minimum number of tests to be defined can be deduced from the number of testability flows: at least one test per flow. It allows evaluating the testing effort.

c) SCADE model check: It consists in analyzing orphan and missing flows data and may lead to modifications in the SCADE model: suppress the SCADE part related to orphan flows; add the SCADE part related to the unimplemented DFS requirements highlighted by the missing flows. This checking activity needs human intervention.

C. Tests coverage against SCADE model

Testing is the dynamic verification technique led on the simulated code of the system in order to verify the completeness and the correctness of implanted requirements. This coverage analysis uses testability flows associated with system requirements in the previous section to give information about defined test cases for validation. Fig. 17 presents the tests coverage analysis process.

This tests coverage analysis is mainly composed of three activities.

a) Requirements test cases identification: This activity consists in using the TRD to identify test cases associated with each requirement described in the DFS. The identification is performed manually.

b) Tests coverage analysis: This step highlights the link, for each requirement, between relevant testability flows and identified test cases. When a relevant flow associated with a requirement cannot be linked to a described test in the TRD, a missing test is identified. When that a same flow is linked to different tests, potential redundant tests are identified. It is also pointed out that the number of determined flows is dependant of the ITM modeling (Section II). In the current study, the ITM modeling principle is based on branch and decision coverage criteria.

c) TRD document check: This third activity consists in analyzing missing and potential redundant tests data and may lead to modifications in the TRD. These modifications could be: the definition additional tests related to testability flows which are not associated with any test and the deletion tests when analysis shows they are functionally redundant.

Otherwise, the relation between relevant flows and test cases can be used to define tests scheduling. Indeed, Start-Small strategy (Section II) proposes an order of execution of the tests related to the selected flows. This order aims at minimizing diagnosis effort.

In the following section, we apply our methodology to an academic example and present the results of its use for two industrial case studies.

VI. CASE STUDIES

In this section, we first depict two experiments on coverage analyses: the pumping system which is academic and two Auto-flight systems which are AIRBUS operational systems. Then we give some conclusion on the relevance of the testability measures in the AIRBUS systems functional testing process.

A. Coverage analyses

For interpretation purpose: (1) means the "Output variable approach", (2) means the "Set of output variables approach" and (3) means the "Component approach". We will focus on nominal tests coverage analysis in this paper.

1) A house pumping system

This academic case study controls the water supply of a house. Two pumps are used to specify this system: the first one brings up water from a well to fill a tank; the second pump supplies water to the residence. Three main functions can be defined for controlling this system: one manages the first pump; another controls the second pump and the last one performs the system's global status. These functions can be described by the following requirements:

- R1: the system shall actuate the first pump when the water-level in the tank decreases and reaches the "filling level";

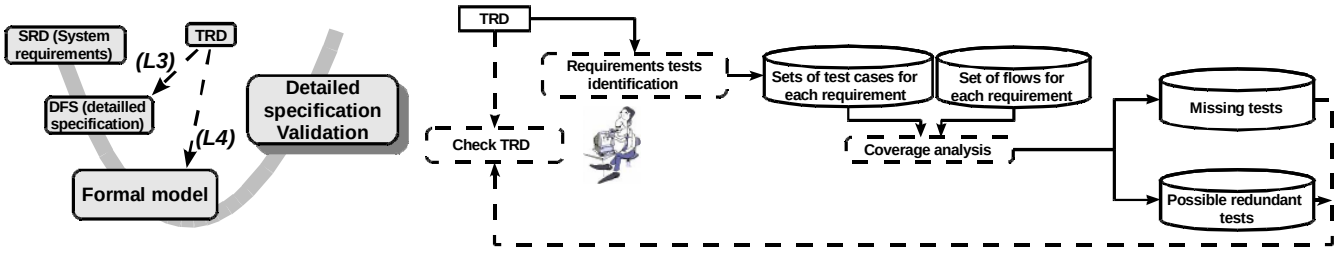

Figure 17. Tests coverage analysis against SCADE model process

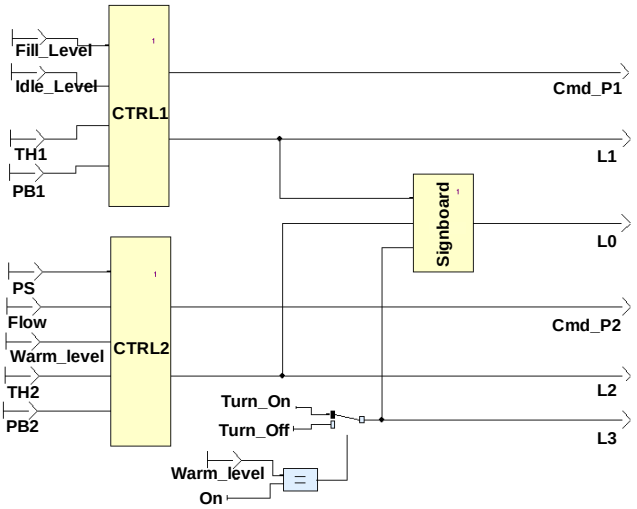

Figure 18. Formal specification of pumping system

- R2: The pumping shall stop when the water-level in the tank raises and reaches the "pumping idle level";
- R3: The first pump shall stop running when the temperature of the pump is abnormally high (only a push button can actuate it).
- R4: The second pump shall be actuated by the decrease in pressure due to the turning on of a tap in the residence. Its shutdown is provoked by the turning off all the taps in the residence;
- R5: the system shall warn and idle the second pump when water reaches the "warning level" (the water-level is lower than the "filling level");
- R6: The pump shall be idled when the water flow rate is too low;
- R7: The pumping shall stop when its temperature is abnormally high (only a push button can actuate it).
- R8: the system shall indicate its global status by taking into account the two pumps.

The following figure Fig. 18 shows pumping system SCADE model.

The "Tab. II" gives the result of testability analysis process on this case study. It shows that the number of tests (48) is higher than the number of testability flows (45, 45 or 27) selected using respectively (1), (2) or (3).

We observe that the number of selected testability flows is stable from using (1) to (2) in this case study. This is mainly due to the structure of the SCADE model. Indeed, each output variable corresponds to a function description. The number of selected flows decreases from using (2) to (3).

The requirement R8 is verified in combination with the seven other requirements. Indeed, L0 represents the state that is checked what ether the pump behavior.

An example of functional tests defined for requirement R1 is described as follow: *"the water-level reaches the filling level (Fill_level = true and Idle_level = false); the pump temperature is normal (TH1 = false and PB1 = false); as result the first pump is activated (Cmd_P1 = true)"*.

The experiment result analysis outlines interesting points:

- For this academic example, we found no orphan testability flows; neither missing implemented requirement in the SCADE model. Indeed, this specification is a final version which has been tuned;
- In order to introduce missing test problem, we deliberately omitted to define tests verifying the second pump is running without output rate (R5). As a result, three testability flows have not been linked to any test;
- For redundant tests, we identify several tests verifying the second pump behavior when its temperature is too high (R7). These tests are linked to the same testability flows. Consequently, they can be considered redundant because they all address the same pump behavior.

TABLE II. PUMPING SYSTEM ANALYSIS RESULT

Functional requirements	Output variables	Number: Defined tests	Number: Selected nominal flows (1)	(2)	(3)	Number: *Selected initialization flows* (1)	(2)	(3)	Input variables
R1 R2	Cmd_P1	6	**5**	**5**	**5**	*5*	*5*	*5*	*Fill_level*; *Idle_level*; *TH1*; *PB1*.
R3	Cmd_P1 L1	6	**6**	**6**	**4**	*6*	*6*	*4*	*Fill_level*; *Idle_level*; *TH1*; *PB1*.
R4	Cmd_P2	6	**4**	**4**	**4**	*4*	*4*	*4*	*PS*; *Flow*; *Warn_level*; *TH2*; *PB2*.
R5	Cmd_P2 L3	6	**3**	**3**	**3**	*2*	*2*	*2*	*PS*; *Flow*; *Warn_level*; *TH2*; *PB2*.
R6 R7	Cmd_P2 L2	24	**13**	**13**	**9**	*13*	*13*	*9*	*PS*; *Flow*; *Warn_level* *TH2*; *PB2*.
R8	L0	48	**14**	**14**	**2**	*14*	*14*	*3*	*Fill_level*; *Idle_level*; *Warn_level*; *TH1*; *PB1*;*TH2; PB2*.

1) Industrial examples

Our testability analysis process has been experimented with two systems LM1 and LM2 provided by AIRBUS. These examples are extracted from an AIRBUS program Auto Flight systems. LM1 and LM2 contain respectively 69 and 146 nodes.

The following table "Tab. III" exposes the result of the defined testability analysis methodology application on these systems.

We observe that the number of selected testability flows decreases from (1) to the (3). The "Section IV" describing the different testability approaches gives explications about the mechanism. Indeed, the description of some output variables can share SCADE model parts.

In (1), these common parts are analyzed for each output variable. Thereby, an important number of testability flows is determined using this approach.

Regarding (2), common parts using by a set of output variables related to a function are analyzed once during testability analysis. Common parts related to different functions are analyzed several times. So, the number of testability flow decreases compared to (1) but is still high.

In (3), all SCADE model related parts are analyzed together. Each common part is explored only once during testability analysis. As a result, testability flows determined by this approach represent the lowest number compared to the two others.

Considering these approaches, the principle of (3) sticks to the AIRBUS testing process of system validation.

The experiment result analysis of LM1 and LM2 outlines the following points:

- No orphan testability flows has been detected after running our testability analysis process. Neither missing flow is identified in the detailed specification.
- Considering (3), for LM1 (resp. LM2), the number of tests (70 (resp. 114)) is lower than the number of flows (84 (resp. 127)). At least, 14 (resp. 13) tests are missing.
- During tests coverage analysis process, we identified that several tests are linked to the same testability flows. Consequently, they can be considered redundant.

These experiments show the testability analysis methodology described in Section IV can support efficiently the AIRBUS system detailed specification validation process. Nevertheless, the applicability of the method in term of scalability must be confirmed on larger operational systems.

B. Testability measures assessment

This assessment aims at checking testability measures relevance in the AIRBUS systems validation context. Indeed, these measures should give information about the complexity introduced by some operators during test definition. The main idea is the highlighting of a possible correlation between predictive controllability and observability measurements and the test effort during the validation of the system specification. The results show that it is difficult to build this relation in AIRBUS context based on functional test. We use the diagram of operators described below Fig. 19 to illustrate this difficulty. It is extracted from the system LM1. This diagram is composed of temporal operators (PULSE1, PULSE2 and PREV) and logic operators (AND and NOT). PULSE1 (resp. PULSE2) outputs pulses on the rising (resp. falling) steps of a Boolean signal. PREV delays a Boolean signal. I_1, I_2, I_3, I_4 are the inputs of the diagram and O_1, O_2 represent the outputs the diagram of operators.

The "Tab. IV" exposes the testability measures associated with each operator used in the diagram above.

1) Controllability measurement

The "Tab. IV" highlights a difference of about 13% between the controllability measure associated with AND_1 (1.0) and AND_2 (0.8754). This difference is due to the lost of information caused by AND_1 on AND_2 input. This reduces the effective information quantity available on AND_2. Therefore, the controllability measurement of AND_2 becomes lower than the AND_1.

TABLE III. LM1 AND LM2 TESTABILITY ANALYSIS RESULT

	Number								
	Functional requirements	Output variables	Defined tests	Selected nominal flows			Selected initialization flows		
				(1)	(2)	(3)	(1)	(2)	(3)
LM1	34	58	70	**615**	**198**	**84**	*155*	*83*	*73*
LM2	57	92	114	**534**	**202**	**127**	*203*	*132*	*112*

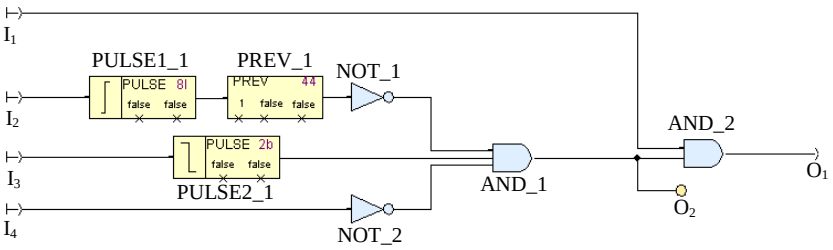

Figure 19. Diagram of operators extracted from the system LM1

TABLE IV. TESTABILITY MEASURES

	PULSE1_1	PREV_1	PULSE2_1	NOT_1	NOT_2	AND_1	AND_2
Controllability	1.0	0.9539	1.0	0.9443	1.0	**1.0**	**0.8754**
Observability	0.9273	0.9271	0.9177	**0.9271**	**0.8754**	1.0	1.0

In a functional test context, the activation effort of AND_2 is at the most equal to the needed effort to activate AND_1. The coverage effort of AND_2 is limited to the production of "true" and "false" by AND_1. Indeed, the second input of AND_2 corresponds to a system input (I_1). Considering this example, we can conclude that it is difficult to construe the controllability measurement (defined as such) in terms of the effort of an operator or a component.

2) Observability measurement

The observability measures associated to operators presented on "Tab. IV" show a difference of about 5% between NOT_1 (0.9271) and NOT_2 (0.8754). Indeed, the effective information quantity available on the output NOT_1 is lower than NOT_2 one. Then, the information loss from NOT_1 is less important than NOT_2 from the both operators to the outputs (O_1 and O_2).

In a functional test context, the observation effort of the output of NOT_1 is equal to NOT_2 one. Indeed, the information flows produced by the both operators are treated in the same way to the outputs (O_1 and O_2). Considering this case, we can conclude that it is difficult to interpret the observability measurement (defined as such) in terms of the effort of an operator or a component.

VII. CONCLUSION AND FUTURE WORK

This paper deals with the definition of a methodology based on testability principles to support traceability activities and test design in the system validation process.

We propose methods highlighting links between the detailed specification, the SCADE model, the test data and the testability flows. AIRBUS systems testability analysis required the extension of SATAN (System's Automatic Testability ANalysis) technology principles. This extension concerns the definition of Airbus specific operators modeling techniques.

We also introduce the temporal aspect in the testability analysis process by proposing the information loss assessment technique considering several execution cycles of the system. A testability flows classification technique has been defined. It splits testability flows in two categories: the initial flows related to system initialization phase; and the nominal flows. Such an approach allows facilitating coverage analysis activities and tests design. The validation phase is thus shortened generating important cost and effort reduction. However, we demonstrate that testability measures, such as defined, do not provide relevant information about the complexity introduced by some system parts during test definition in the functional testing context.

In the future, our work will focus on two main objectives. The first one consists in leading new experiments on a larger number of systems in order to enhance tools supporting the systems specification coverage analysis methodologies. This allows their consolidation for their deployment in operational conditions.

The second objective of our future work aims at adapting testability measures to a functional testing context. This requires the consideration of value and constraints related to information flows in the system specification during information loss coefficient assessment of operators.

REFERENCES

[1] F. Doumbia, O. Laurent, C. Robach, and M. Delaunay, "Using the Testability Analysis Methodology for the Validation of AIRBUS Systems," VALID 2009, First International Conference on Advances in System Testing and Validation Lifecycle, pp.86-91, September 2009.

[2] H. V. Do, M. Delaunay, and C. Robach, "Integrating testability into the development process of reactive systems", *IASTED SE 2007*, Innsbruck, Austria, February 2007.

[3] C. Robach: "Test et testabilité de systèmes informatique", *PhD Thesis*, 1979.

[4] N.Halbwachs, P. Caspi, P. Raymond, and D. Pilaud: "The synchronous dataflow programing language LUSTRE", *Proceedings of the IEEE*, 79(9): 1305-1320, September 1991.

[5] Esterel Technologies SA. *SCADE Technical Manual*, 2005.

[6] R. S. Freedman. Testability of Software Components. *IEEE Transactions on Software Engineering*, 17(6):553–564, Jun 1991.

[7] Y. Le Traon and C. Robach. Testability Measurements for Data Flow Design. *In Proceedings of the Fourth International Software Metrics Symposium*, pages 91–98, Albuquerque, New Mexico, Nov 1997.

[8] T. J. McCabe. A complexity measure. *IEEE Transactions on Software Engineering*, 2(4):308–320, December 1976.

[9] B. A. Nejmeh. Npath: A complexity measure of execution path complexity and its applications. *Communications of the ACM*, 31(2):188–200, February 1988.

[10] C. Robach and P. Wodey. Linking design and test tools: an implementation. *IEEE Transactions on Industrial Electronics*, 36:286–295, 1989.

[11] J. M. Voas and K. W. Miller. Software testability: The new verification. *IEEE Software*, 12(3):17–28, May 1995.

[12] H. V. Do, C. Robach, M. Delaunay, and J. S. Cruz. Testability Analysis for Grapically Describec Algorithms of Reactive Systems. *EMBEDDED REAL TIME SOFTWARE (ERTS) 2006*, Toulouse, France, January 2006.

[13] A. Dammak: "Etude de Mesures de Testabilité de Systèmes Logiques", PhD Thesis, 1985.

[14] A. Benveniste and G. Berry. The Synchronous Approach to Reactive and Real-Time Systems. *Proceedings of the IEEE*, 79(9), 1991.

[15] C. Robach, H. V. Do, and M. Delaunay. Testability as a component of CASE tools. *International Conference on Degradation, Damage, Fatigue and Accelerated Life Models in Reliability Testing*, Angers, France, May 2006.

[16] Bermoulli Distribution. *http://en.wikipedia.org/wiki/Bernoulli_distribution*.

[17] RTCA/DO-178B, "*Software Considerations in Airborne Systems and Equipment Certification*", December 1992.