Integrating Testability and Diagnosis methods into AIRBUS Systems Development Process

Fassely Doumbia, Odile Laurent
AIRBUS Operations SAS
{Firstname.Lastname}@airbus.com

Chantal Robach
LCIS – Grenoble Institute of Technology
Chantal.Robach@esisar.grenoble-inp.fr

Abstract. This paper exposes methodologies, based on testing strategies (Start-Small and Multiple-Clue), able to support two main steps of AIRBUS systems development process: formal specification validation activities and systems testing activities on AIRBUS Final Assembly Line (FAL). The experiments show that these methods can quantify testing efforts, guide functional tests definition, facilitate detailed design coverage activities and ease fault location.

Introduction

Development of avionic systems must comply with a number of standards (ARP4754 [SEA. 1996], DO178-B [RTCA. 1992], etc.). 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 different phases of V&V are of great interest for the aeronautics domain. Testing is the most commonly used technique for systems V&V activities. In this context, we first propose a testability analysis methodology offering useful methods to support the validation of systems formal detailed specification. The *testability* is defined as a predictive quality factor which is traditionally related to the validation cost: testability is a predictive test effort and test efficiency estimate. We also propose a diagnosis methodology giving information for helping the systems verification process from the tests design or the tests definition phase down to fault location phase during the diagnosis on AIRBUS FAL. The *diagnosis* is defined here as the action of locating a faulty component of the system based on an abstract model.

This paper first presents the AIRBUS systems development process. A methodology supporting the coverage analysis of the system formal specifications during the validation process is then described. And finally, methods supporting systems testing on FAL are depicted.

AIRBUS systems development process

Figure 1 presents AIRBUS systems V&V activities. This V&V cycle relies on a generic V-cycle process for which validation activities are added due to the modelling process at system level. 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 test activities. The "*System level*" represents the system specifications and design definition phase. The "*Equipment level*" corresponds to the implementation of system specifications and design in real equipment. We will focus on systems specifications validation and ground tests activities on FAL in the rest of the paper.

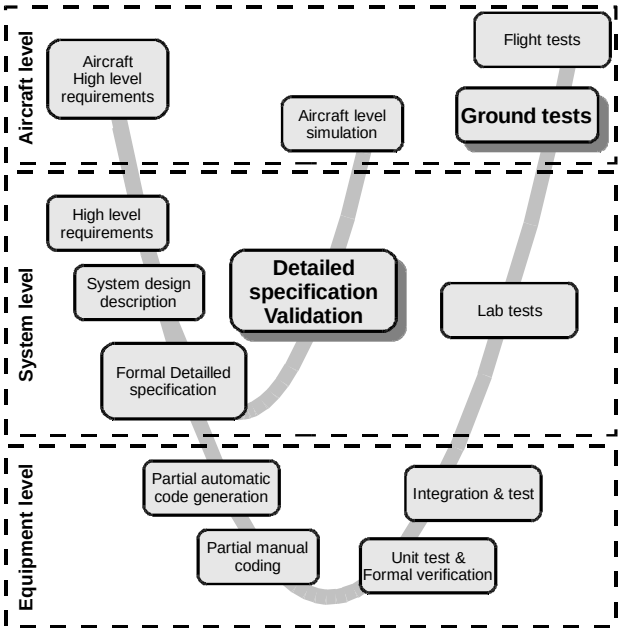


Figure 1. Airbus Systems development process

System validation cycle. System validation consists in determining whether the requirements are correct, or formulated as the following question: "Are we building the right product?" The validation cycle activities are mainly based on testing and traceability activities (see Figure 2). The model validation activities (5) consist in executing tests on a desktop simulator. The traceability process relies on a documents cascade and on a data model managed by DOORS tool (`www.telelogic.com`):

- *SRD* (System Requirements Document) (1) specifies the system requirements refined from aircraft requirements.
- *DFS* (Detailed Functional Specification) (2) is the document describing the system detailed functions that meet the system requirements.
- *TRD* (Test Requirements Document) (3) defines for each DFS function the test data (functional tests description, tests vectors and expected tests results).
- *Formal model* (or detailed specification) (4) corresponds to a formal implementation of the detailed functions, from which the embedded code is automatically generated. The detailed formal specification is described using data-flow formal languages such as SCADE [Esterel Technology SA. 2005].

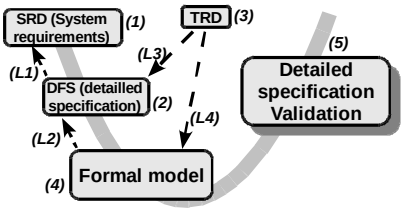


Figure 2. System validation cycle

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

- System requirements described in the SRD are considered in the system functions (L1);
- The SCADE model implements 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).

Ground tests process. This test phase is led on Airbus Final Assembly Line (FAL). It consists in determining whether a system satisfies its requirements, or formulated as the following question: “Is the product rightly?” The FAL activities consist in assembling the aircraft structure elements (sections equipped with appropriate wires and equipments, wings, etc.) and installing avionics systems and engine, and testing the aircraft behavior before aircraft delivery to airlines. Figure 3 presents this functional testing process on FAL which aims at verifying installed systems and their interconnections. Considering the aircraft development process, these activities correspond to a part of the integration testing. At this stage of the aircraft manufacturing, it is supposed that all system components involved in FAL tests (computers, push buttons, sensors, etc.) are tested separately and are not faulty.

Two main parts can be highlighted in the systems verification process on FAL: the test cases definition phase; the test case execution and diagnosis phase. The test cases definition part is composed of activities which are described below.

– *Test Requirements (TR) specification*: from the aircraft high level requirement, TR specification documents are produced by test designers. These documents contain system requirements description and test objectives definition for testing activities on FAL before delivery for the first flight;
– *Test Instructions (TI) description*: for each test objective of the TR, a set of test instructions is defined. A test instruction is composed of tests definition and test environment on which the tests will be run. To optimize the testing effort (required configuration, controlling the complexity, etc.), the test objectives definition order has to be respected during tests execution phase. A TI description document is carried out for each system.

The main activities of test execution and the diagnosis part are described below.

– *Test execution and verdict*: Each test objective is verified using test cases and expected tests results described in the TI documents. Automatic testing tools execute most of the systems test instructions and give the verdict. But, for some particular tests, human expertise is needed to give the verdict;
– *Diagnosis after failure detection*: This activity aims at identifying and replacing the faulty resource. It is carried out, for each test objective, when one or more test cases results are incorrect. At the FAL level, diagnosis is essentially based on human expertise.

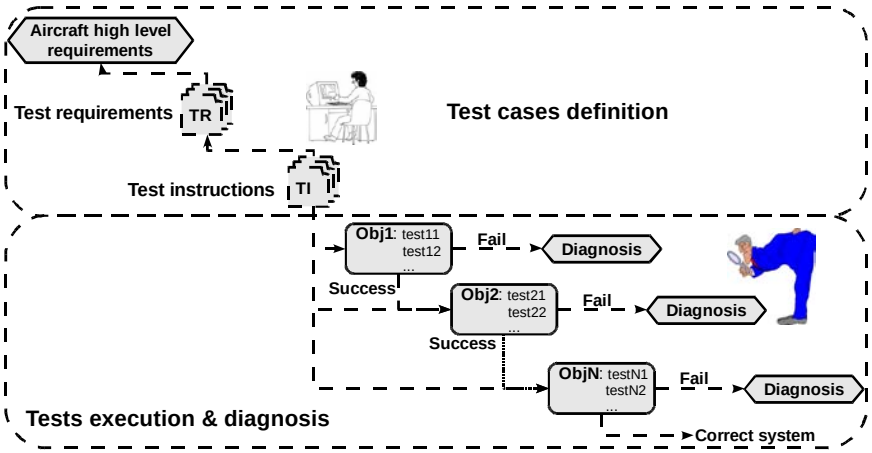


Figure 3. Systems verification process on Airbus FAL

Coverage analysis of the system formal specifications

As depicted in Figure 2, coverage analysis activities are mainly based on a traceability process using a documents cascade. In order to alleviate drastically these coverage analysis activities, we propose a process relying on the testability flows determined from the SCADE model (Figure 4). This process is composed of: the testability analysis which consists in identifying testability flows and selecting relevant flows using a strategy and in performing system formal specification coverage analysis methodology implemented using testability flows (L2, L3 and L4). The process cannot support the coverage analysis (L1). Indeed, L1 is based on document analysis.

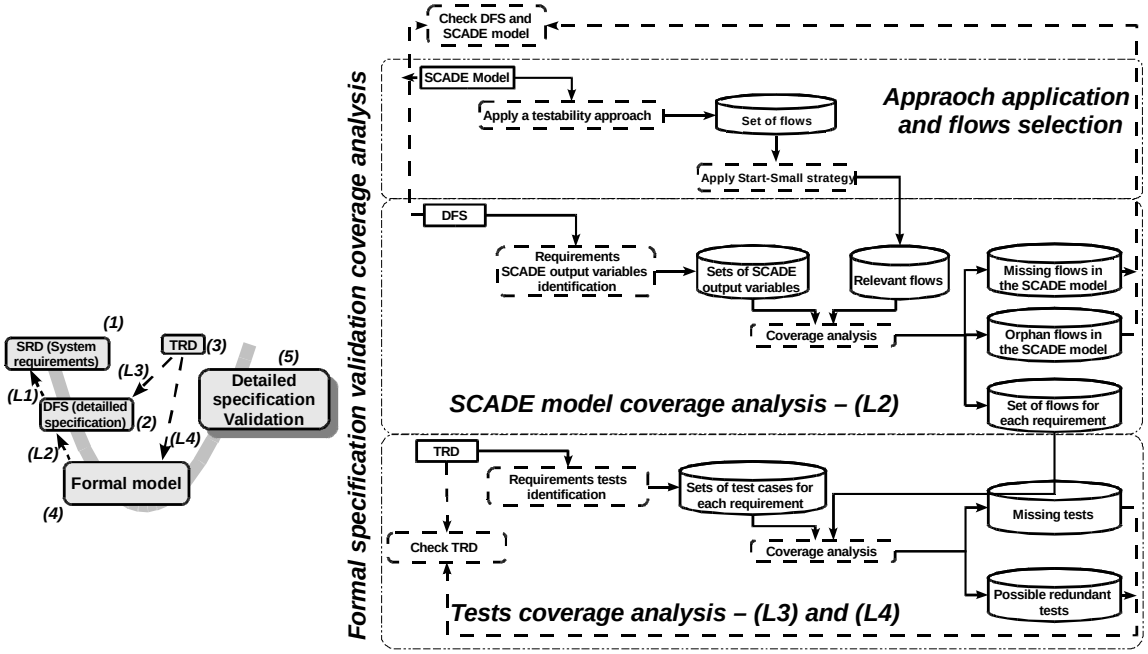


Figure 4. System formal detailed specification validation coverage analysis methodology

Testability flows and Start-Small strategy

Testability flows. We propose a method for analyzing the testability of a system data-flow specification. This method identifies system testability flows. A testability flow is an information path that carries information from one or several inputs, through modules and transitions, to one output of the system. The testability analysis is based on a model representing the data-flow aspect of the system [Doumbia, F, Laurent, O, Robach, C and Delaunay, M. 2009]. Figure 5 below depicts two testability flows (F_1 and F_5) using bold lines on a SCADE specification.

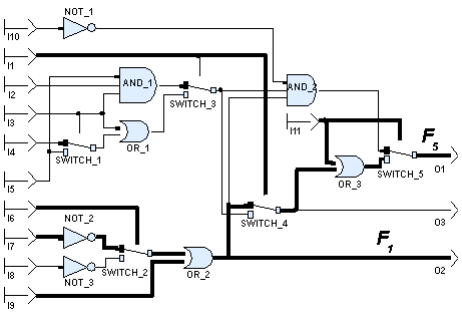


Figure 5. Graphical representation of testability flows

Start-Small strategy. Start-Small is a testing strategy. A test strategy defines the way to conduct test activities and to analyze test results. Start-Small allows the selection of a set of relevant testability flows for testing purpose. Flows are chosen according to the following criterion: every operator used in the system must be activated at least once in order to ensure the coverage of all operators.

The Start-Small strategy gradually covers the operators by choosing flows with an increasing number of covered operators. 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 operators. The next one contains a minimal number of operators that are not activated. In this strategy, a new flow is tested only if all faults detected in previous flows are corrected, as depicted in Figure 6. Tests are defined either automatically or manually for each selected flow.

- **(a)**: the first selected flow (F_1) to be tested contains the following operators: NOT_2, SWITCH_2 and OR_2.
- **(b)**: the second selected flow (F_2) to be tested contains the following operators: NOT_3, SWITCH_2 and OR_2. NOT_3 is the operator which is not activated yet.
- **(c)**: the third selected flow (F_3) contains the following operators: NOT_2, SWITCH_2, OR_2, SWITCH_4. SWITCH_4 is the new operator to activate.
- And so on until cover all operators.

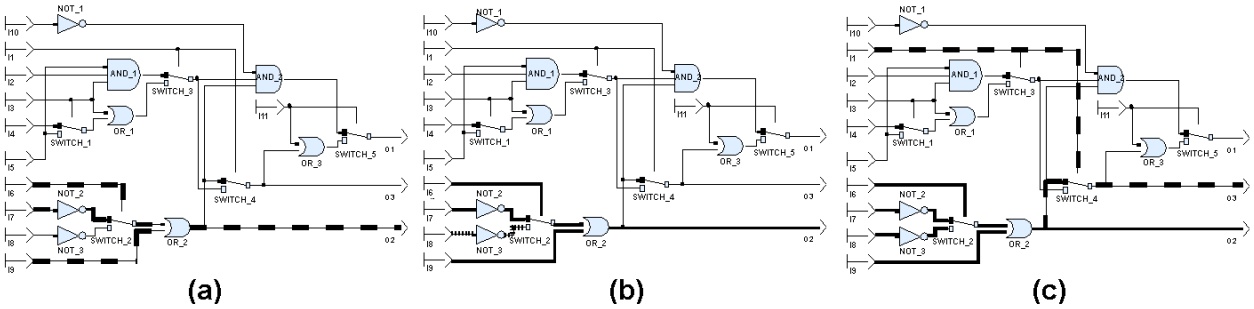


Figure 6. Start-Small principle illustration

Start-Small strategy selects eight relevant testability flows instead of the sixteen determined for the system specification depicted in Figure 6. This illustration highlights the importance of respecting testability flows selection order during the testing phase. Indeed, it reduces testing and diagnosis effort. Start-Small ensures branch and decision coverage criteria (F_1 and F_2) in this case. These criteria can be adapted according to the system criticality.

Coverage analysis methodology

The coverage analysis methodology defined for AIRBUS systems is based on the DFS document, the TRD document and the SCADE model. It aims at highlighting information about: the completeness of systems formal specification describing functional requirements defined in the DFS; the completeness and the relevancy of defined tests for systems validation. This methodology is composed of three main activities: the testability flows identification and relevant flows selection; the coverage assessment of the SCADE model against the requirements, and coverage analysis of the test cases against the SCADE model.

Testability flows identification and relevant flows selection. Testability analysis information is useful only if it reflects the development process. We defined three different testability methods in order to fit AIRBUS needs [Doumbia, F, Laurent, O, Robach, C and Delaunay, M. 2009].

Start-Small strategy is applied for relevant flows selection.

- The first method is based on an *"output variable"*: it consists in identifying and analyzing the part of formal specification related to each SCADE output variable involved in the DFS. Testability flows identified using this approach allow a local testability view of the system for all output variables;
- The second method is based on a *"set of output variables"*: it consists in identifying and analyzing the part of formal specification related to a set of output variables associated with each system function. Testability flows identified using this approach allow the testability assessment for each function defined in the system;
- The last method is based on the *"component"*: It aims at analyzing each independent part of the formal specification from a data-flow point of view (called hereafter components). This allows analyzing separately the system independent parts.

To illustrate testability methods, we use the system specification depicted in Figure 5. Table 1 below summarizes the result when applying these methods. We assume *"O1"* performs a function and *"O2"* and *"O3"* perform another function.

Table 1. Testability approaches illustration

	Methods		
	Output variables (1)	Set of output variables (2)	Component (3)
Number of selected flows	**12** (6 for *O1*, 2 for *O2* and 4 for *O3*)	**11** (6 for the *1st function* and 5 the *2nd function*)	**8** (5 for *O1*, 2 for *O2* and 1 for *O3*)

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 method (1), these common parts are analyzed for each output variable. Thereby, an important number of testability flows is determined. Figure 7 highlights the notion of common parts. Indeed, part (A) performs the output variable *"O2"* and (B) performing *"O3"* 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 method (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. (A) performs *"O2"* and *"O3"* output variables; (B) performing *"O1"* involves (A).
- In method (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.

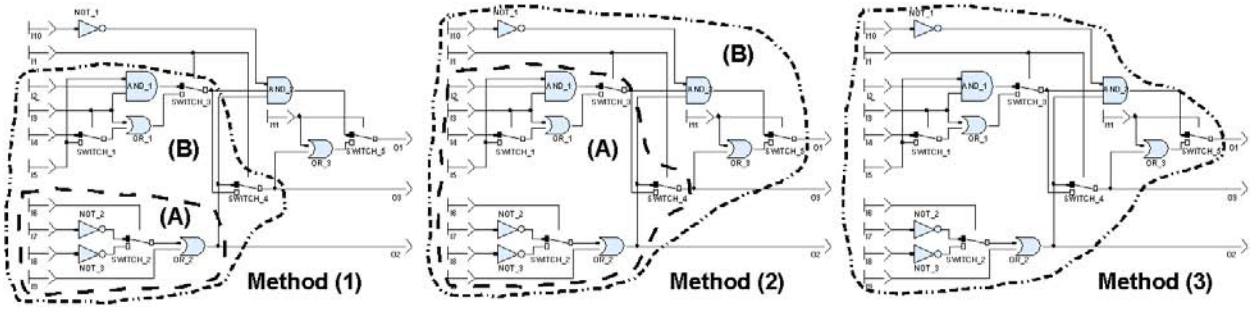


Figure 7. Specification common parts illustration

Considering these methods, experiments show that the principle of the last one sticks to the AIRBUS testing process for systems validation [Doumbia, F, Laurent, O, Robach, C and Delaunay, M. 2009].

SCADE model coverage against 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. Figure 8 presents the SCADE model coverage analysis process. It is mainly composed of three activities.

- The first activity consists in using the DFS document to identify SCADE output variables involved in each system requirement definition. The identification is performed manually.
- The second activity corresponds to the formal model coverage analysis. This activity consists in associating automatically relevant testability flows with identified output variables. This step highlights a functional link between requirements and testability flows. It also points out the presence of orphan flows and missing flows.
  An orphan flow is 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.
- The third activity consists in analyzing orphan and missing flows data and may lead to modifications in the SCADE model: deleting the SCADE part related to orphan flows; adding the SCADE part related to the unimplemented DFS requirements highlighted by the missing flows. This checking activity needs human intervention.

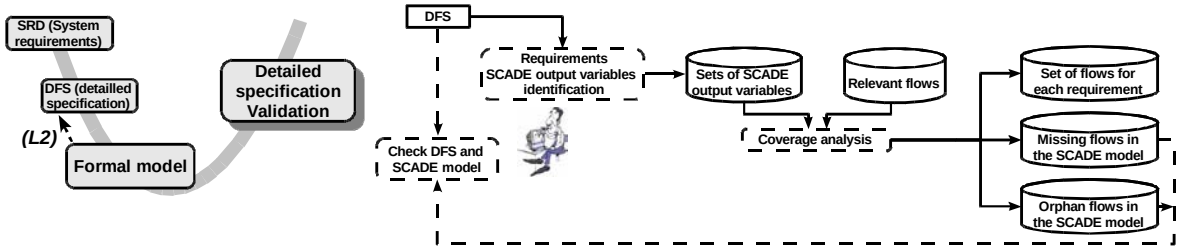


Figure 8. SCADE model coverage analysis against requirements process

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. Figure 9 presents the tests coverage analysis process. It is mainly composed of three activities.

- The first activity consists in using the TRD to identify test cases associated with each requirement described in the DFS. The identification is performed manually.
- The second activity corresponds to the tests coverage analysis based on the testability flows, as described above. This activity highlights the link, for each functional requirement, between selected relevant testability flows and identified test cases. It also provides information about missing tests and potential redundant tests.

A *missing test* is identified when a relevant flow associated with a requirement cannot be linked to any described test in the TRD.

Potential *redundant tests* are identified when a same flow is linked to different tests. A functional analysis of these tests has to be performed to conclude about these tests.

– The third activity consists in analyzing missing and potential redundant tests data and may lead to modifications in the TRD: defining additional tests related to testability flows which are not associated with any test and deleting tests when analysis shows they are functionally redundant. This checking activity needs human intervention.

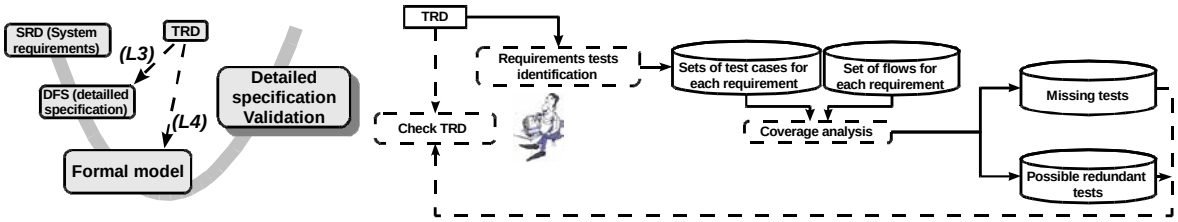


Figure 9. Tests coverage analysis against SCADE model process

Experiments

Systems formal specification coverage methodology has been experimented with two systems, LM1 and LM2, provided by AIRBUS. These examples are extracted from the AIRBUS Auto Flight systems. Auto Flight systems allow maintaining the flight path defined by the crew. They also control the aircraft and the engines. LM1 and LM2 contain respectively 69 and 146 SCADE nodes. Table 2 exposes the result of the component based testability analysis of these systems.

Table 2. Component based testability analysis results

	Number			
	Functional requirements	Output variables	Defined tests	Selected flows
LM1	34	58	70	**84**
LM2	57	92	114	**127**

The coverage analysis of LM1 and LM2 outlines the following points:

– No orphan testability flows has been detected after running our testability analysis process. No missing flow is identified in the detailed specification.

– 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 the test 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 coverage analysis methodology described in the previous sections can support efficiently the AIRBUS system detailed specification validation. Nevertheless, the applicability of the method in terms of scalability must be confirmed on larger operational systems.

Tests design and diagnosis on Airbus FALs

The optimization of the testing process conducting to time saving is an important point to consider during systems verification on FAL (Figure 3). Then, methods that can improve tests design and diagnosis shall be explored. More precisely, for test cases specification, the main challenges are:

the definition of relevant and non-redundant test cases; the test requirements coverage against test cases. In the diagnosis phase, the main challenge is the identification for certain the faulty resource [Doumbia, F, Laurent, O, Atger D and Robach, C. 2009]. We propose an enhancement methodology of this process in order to meet some of these challenges. This methodology does not support test requirements coverage analysis against test cases. Figure 10 depicts this improvement methodology relying on the Mutiple-Clue strategy suitable for systems diagnosis during the final test campaigns at the end of the development.

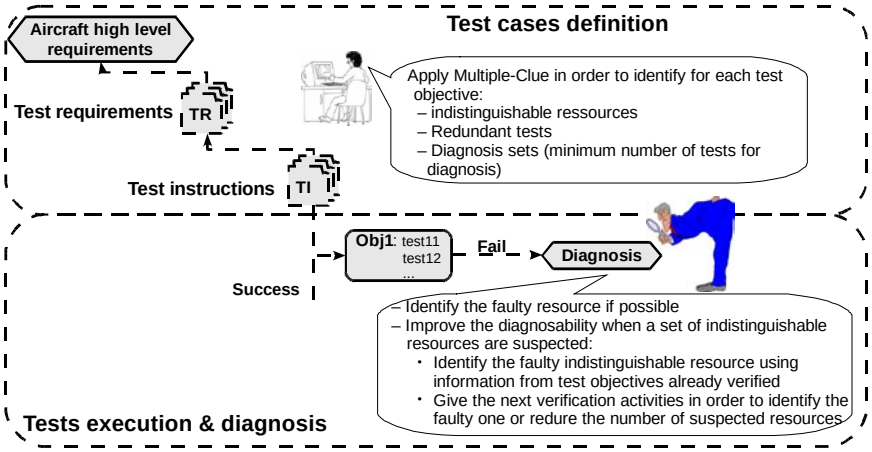


Figure 10. Systems testing process improvement methodology on FAL

Multiple-Clue strategy

The Multiple-Clue is based on a "cross-checking" test strategy. The cross-checking test strategy consists of running a set of tests and then collecting all test results before trying to locate and fix any faulty component. Multiple-Clue chooses relevant tests in order to exercise each resource at least once during the tests campaign and to ensure a precise diagnosis when a failure occurs.

Hypothesis. Multiple-Clue application supposes the following hypothesis: only one faulty resource among the exercised components is revealed by a test case set (single fault hypothesis) and oracles are sure [Doumbia, F, Laurent, O, Atger D and Robach, C. 2009]. Indeed, a test case that does not reveal a fault proves that the executed components are correct (probability of infection and propagation of the faulty result when a statement is executed = 1): all the test cases are taken into account in the diagnosis strategy.

Implementation. Multiple-Clue uses an abstract model of the system for analyzing the ease of diagnosis of its components. A system S can be generally considered as a set of components (hardware and software) implementing functions in order to fulfill its requirements. These functions are verified by executing the system based on a set of test cases *T*. The system model corresponds to a matrix representation associating all the system resources and related test cases. A *resource* R_i corresponds to any component that can cause a failure in system S. A *test case* T_i corresponds to a combination of system inputs that allows exercising a set of resources contributing to a system function. Each test case T_i may reveal a failure among exercised resources.

Let *R* be the set of resources and *T* be the set of defined test cases for a system *S*, we can write the following expressions:

$$ntests = |T| \; ; \; nresources = |R| \; ; \; trace(T_i) \subseteq R$$

The matrix representation, called hereafter the diagnosis matrix, corresponds to an array *M[i,j]* ($j \in [1 \ldots nresources]$, $i \in [1 \ldots ntests]$) where *j* refers to resources and *i* refers to test cases. It is built such as:

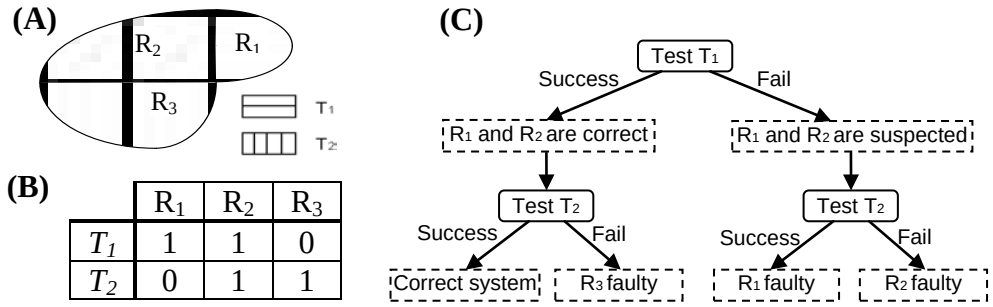
$$\begin{cases} m_{ij} = 1; R_j \in trace(T_i) \text{ or } T_i \text{ exercises } R_j \\ m_{ij} = 0 \text{ otherwise} \end{cases}$$

Figure 11 (A) shows a system "3R" composed of three resources (R_1, R_2 and R_3). Test case T_1 exercises R_1 and R_2; test case T_2 exercises R_2 and R_3. The diagnosis matrix associated with this system is represented in Figure 11 (B).

The Multiple-Clue assesses the diagnosis effort using the diagnosis matrix in order to provide useful information about:

- *Indistinguishable resources*: two resources are indistinguishable from each other if they are always exercised together;
- *Equivalent test cases for diagnosis*: two test cases are equivalent or redundant for diagnosis if they exercise the same set of resources;
- *Diagnosis set*: A diagnosis set is a minimum subset of necessary and sufficient test cases to exercise at least once all the system resources and to identify either the faulty resource or the set of suspected indistinguishable resources during diagnosis.

The cross-checking method is carried out using the diagnosis set for faulty resources identification. It is based on the "dicing" method, which is a particular use of slicing [Gallagher K.B. and Lyle J. R. 1991]. It allows the best fault location after failure detection in the single fault hypothesis. This method is illustrated using the diagnosis tree built for System "3R" in Figure 11 (C). A test case is successful when its execution result corresponds to the expected one described in the oracle and all exercised resources are correct. It fails otherwise.



	R_1	R_2	R_3
T_1	1	1	0
T_2	0	1	1

Figure 11. Multiple-Clue Implementation steps

Methodology for enhancing systems testing process on FAL

Automatic implementation of the Multiple-Clue approach needs an abstract model of the system (diagnosis matrix) which represents all the existing relations between system resources and test cases. Considering the system verification process on AIRBUS FAL, information about resources and test cases is clearly specified in the TI description documents for each test objective (Figure 3).

Modeling resources and tests. In the context of integration verification activities based on functional test cases led on FAL, a system resource is a component of the functional chain which can cause a failure as an avionics computer, a cable, a circuit breaker, a sensor, an actuator, etc.

For each test objective defined in the IT, the required configuration is specified and functional test

cases are defined. These test cases encompass system functions and resources verification. Resources modeling depends on the expected accuracy of the diagnosis during the test campaign. More detailed is the resources modeling better the accuracy of the faulty resource identification. Indeed, when a resource represents a part of the system which can cause a set of failures, the diagnosis does not limit to the identification of the faulty resource but another activity is needed to precisely point out the type of failure in order to repair the error. For example, the component managing an aircraft tracking gears *"ANTI-SKID"* mechanism has two different states (active *"state ON"* or inactive *"state OFF"*). Each of these states can cause a failure during test. If the *"ANTI-SKID"* component is modeled as a resource, its incrimination does not give idea about the wrong state. So, if we model each state of this component as two different resources (*"ANTI-SKID ON"* and *"ANTI-SKID OFF"*), the diagnosis is more accurate.

In this paper, we consider a resource as an association of a component and its state. Let *E* be the possible states set of a component and *nR* be the number of modeled resources for the component. We can write the following expression: $nR = |E|$. This expression means that a component having *n* different states is represented by *n* different resources.

Figure 12 shows a graphical representation of the system model. As depicted in Figure 3, each test objective is verified before switching to the following one. Then, we can identify a diagnosis (sub-) matrix for each test objective.

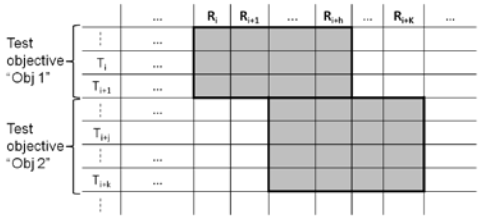


Figure 12.Graphical representation of the system diagnosis matrix

Multiple-Clue application on each test objective points out information about:

- Potential redundant test through the equivalent test cases identification "Figure 13 (b)";
- Resources belonging to an indistinguishable set which cannot be declared faulty for certain after tests execution related to the test objective "Figure 13 (a)";
- The minimum set of tests to run for an optimal diagnosis through the diagnosis set selection;

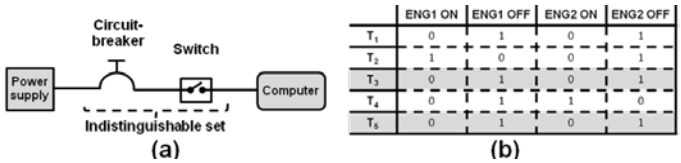


Figure 13. Diagnosis information

When tests fail and involve indistinguishable resources, additional verification activities based on tests execution results of other test objectives sharing suspected resources may be necessary to conclude about the faulty resource.

Improving diagnosis methodology. To reduce the diagnosis effort when indistinguishable resources are suspect, we propose a method which aims at using several test objectives (diagnosis matrices) analysis information in order to help the identification of the faulty resource in a set of indistinguishable resources. The implementation of this method consists in:

- identifying other test objectives which have common resources with the set of suspected resources;
- using the cross-checking approach in order to incriminate the faulty resource or to reduce the set of suspected indistinguishable resources.

The basic principles of this method, considering the single fault hypothesis, are illustrated using the test objective *"Obj N"* (Figure 14). The final objective of this method is to reduce the diagnosis effort according to systems verification process on AIRBUS FAL. *"Obj N"* is composed of five resources with an indistinguishable set of three resources (set 1). Two test cases are selected by Multiple-Clue for diagnosis. We assume that the verdict of T_1 is "success" and T_2 is "fail" during the verification of *"Obj N"*. Then, the diagnosis tree is built and "Set 1" is suspected (Figure 14).

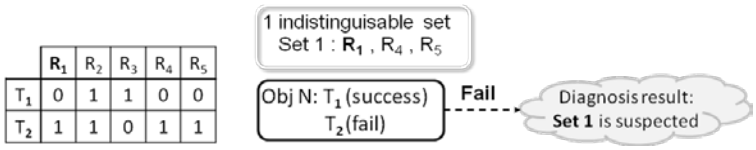


Figure 14. "Obj N" verification information

To improve this situation (three suspected resources), the improvement method consists in searching and verifying other test objectives using some of suspected resources belonging to "Set 1". In this way, we choose another test objective *"Obj M"* sharing R_1 with "Set 1" (test objective *"Obj N"*). Four resources are used by *"Obj M"* with an indistinguishable set of two resources (Set 2). Two test cases are also selected (T_3 and T_4) by Multiple-Clue for diagnosis. We propose two scenarios in order to highlight the main benefits of this method.

The first scenario consists in assuming the verdict of T_3 execution is "fail" and the one of T_4 is "success". This information is used to suspect "Set 2" in Figure 15 (a). As a result, the method implements a cross-checking analysis and identifies the resource R_1 as the faulty resource in Figure 15 (b).

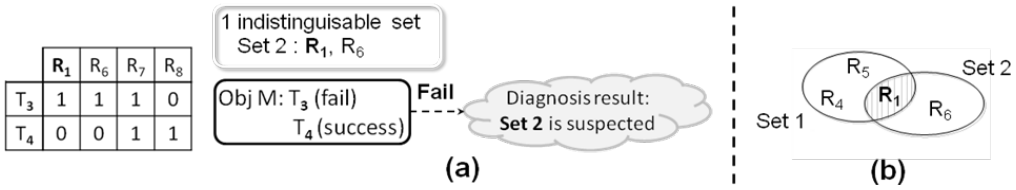


Figure 15. Identification of the faulty resource in an indistinguishable set

The second scenario assumes the verdict of T3 and T4 are "success" in Figure 16 (a). This information means that *"Obj M"* resources are all correct. As a result of the method implementation, the number of suspected resources in the test objective *"Obj N"* (Set 1) is reduced from 3 to 2 resources (R4, R5) in Figure 16 (b).

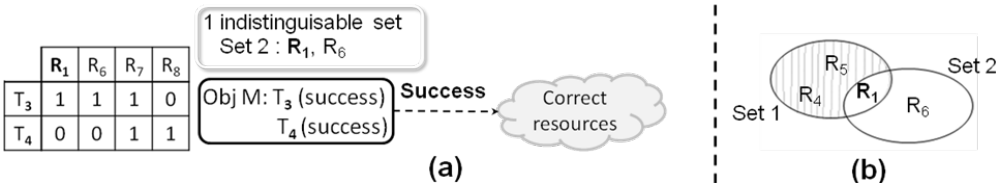


Figure 16. Reduction of the number of suspected resources

An algorithmic study of the principles of this diagnosis improvement methodology has been

performed. Figure 17 presents the process implementing these principles. The main steps of this process are:

- **(A)**: This step corresponds to the automatic diagnosis tree construction using selected tests verdict provided by the tester.
- **(B)**: In this step, the process looks for matrices corresponding to test objectives using some suspected resources. These test objectives may help to improve the diagnosis accuracy.
- **(C)**: This step needs an operator intervention. Indeed, the tester has to choose a next useful test objective to verify, according to the IT document in order to improve the diagnosis.
- **(D)**: The chosen test objective is verified and the selected tests verdict is used to build the diagnosis tree.
- **(E)**: The state of the shared resources used by the chosen test objective is automatically defined.
- **(F)**: This step corresponds to the case where the number of suspected resources may be reduced. It consists in performing automatically an intersection operation between the initial set of indistinguishable resources and the shared suspected resources.
- **(G)**: This step corresponds to the situation where the number of suspected resources may be reduced. It consists in performing automatically a difference operation between the initial set of indistinguishable resources and shared non suspected resources.
- **(H)**: The last step consists in replacing the initial set of indistinguishable resources by the result of either (F) or (G) operation.

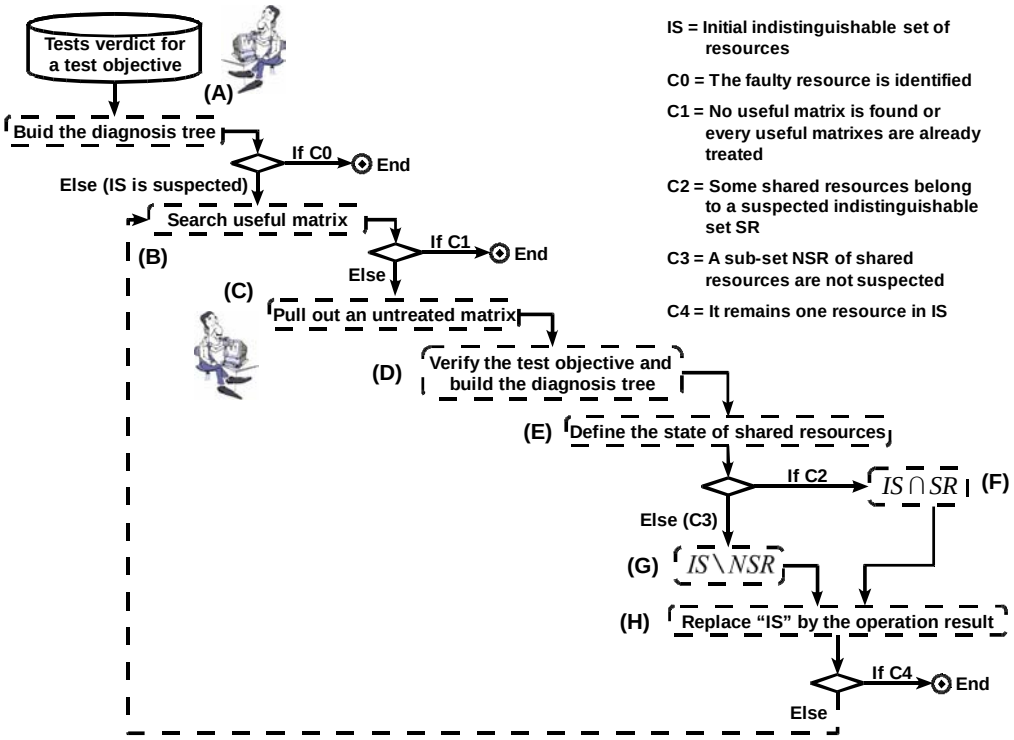


Figure 17. Guided diagnosis process

Experiments

The described methodology helping systems testing on AIRBUS FAL has been experimented with the "tracking gears orientation" system. Based on the TI description document of "tracking gears orientation" system, the test objectives, test cases and resources are identified. Table 3 presents the

main characteristics of the built diagnosis matrix for this system.

Table 3. Diagnosis matrix data

	Number
Test objectives	26
Test cases	169
Resources	100
Resources per test objective	from 2 to 18
Test cases per test objective	from 2 to 18

The table above exhibits the following information:

- 26 test objectives are identified in the IT description document of the case study;
- 169 test cases and 100 resources are modeled for these test objectives;
- For each test objective, the number of modeled resources goes from 2 to 18 and the number of test cases also varies from 2 to 18.

Table 4 summarizes the results of the application of the described methodology. In this representation, *"Before"* means: data relying on AIRBUS current methodology based on engineering judgment and *"After"* means: data provided by the methodology described in this paper.

Table 4. Quantitative benefits

	Number	
	Before	After
Selected test cases	169	80
Redundant test cases	unknown	30
Indistinguishable sets	26	13
Average size of sets indistinguishable resources	4 or 5	2 or 3

Considering the test cases definition stage, the methodology highlights the following useful information:

- Redundant test cases identification: 30 test cases are pointed out as redundant or equivalent for diagnosis. We had no indication about redundant test cases before this experimentation. This information can be exploited by test designers in order to check if these test cases are also redundant from a functional point of view.
- Indistinguishable sets identification: this information, generally proposed after diagnosis, is now available in the test cases definition phase. A possible use of this information is that test designers can decide to define additional test cases or to foresee appropriate test tools in order to improve performance during the diagnosis phase.
- Selected test cases or diagnosis set: from test cases definition phase, experimented methodology identifies relevant test cases (80 among 169) for diagnosis. This information allows test designers to functionally analyze the remaining test cases in order to take a decision regarding their utility.

For the diagnosis phase, our approach also provides information about:

- Faulty resource identification: for each test objective, the selected diagnosis set allows the construction of the diagnosis tree (see Figure 11 (C)) helping faulty distinguishable resource

identification. As a result, the number of test cases execution results to consider for diagnosis is reduced and this contributes to troubleshooting effort reduction.

- Resources diagnosis improvement: The suspicion of a set of indistinguishable resources is common during systems verification on FAL. Information given by our approach can be used to guide next test objectives to verify in order to identify the faulty resource or reduce the number of suspected resources.

Conclusion

The methodologies described in this paper rely on Airbus current development process. In the system formal specification validation process, these methods address the main test challenges by supporting traceability activities and test design using testability methods. As a result, they facilitate coverage analysis activities and tests design. The validation phase is thus shortened, inducing important cost and effort reduction. This methodology, based on start small strategy, can be adapted to other system modeled in another data-flow formalism such as SCADE and to the system criticality tuning the testability flows criteria coverage.

In the testing process on AIRBUS FAL, the methods provide information (using cross-checking approach) about diagnosis effort by helping the definition of relevant and non-redundant test cases and easing the troubleshooting. Therefore, this methodology contributes to reduce testing effort. It can be adapted to systems which can be modeled by a diagnosis matrix.

The adaptability of these methods to other systems and other industrial contexts show their relevance in a system engineering approach.

References

Doumbia, F, Laurent, O, Atger, D and Robach, C. 2009. Using the Multiple-Clue approach for system testing on AIRBUS FAL (Final Assembly Line). In *the Proceedings of the IEEE International Test Conference*.

Doumbia, F, Laurent, O, Robach, C and Delaunay, M. 2009 (Best paper award). Using the Testability analysis methodology for the Validation of Airbus Systems. In *the Proceedings of the First International Conference on Advances in System Testing and Validation Lifecycle* 86-91.

Esterel Technologies SA. 2005. *SCADE Technical Manual*.

Gallagher K.B. and Lyle J. R. 1991. Using Program Slicing In Software Maintenance. In *the IEEE Transaction on Software Engineering* 17(8): 751-761.

Khalil, M, Le Traon, Y and Robach, C. 1998. Automated strategies for software diagnosis. In *the Proceedings of the IEEE International Symposium on Software Reliability Engineering*.

Khalil, M, Novak, F and Robach, C. 2002. Diagnosis strategies for hardware or software systems. In *the Journal of Electronic Testing: Theory and Applications* 18(2): 239-249.

Le Traon, Y, Ouabdesselam, F and Robach, C. 2000. Analyzing testability on Data Flow Designs. In *the Proceedings of the International Symposium on Software Reliability Engineering (ISSRE)* 8(11): 162-173.

O'Farrill, C, Moakil, B and Eklow, B. 2005. Optimized reasoning-based diagnosis for non-random, board level, production defects. In *the Proceedings of the IEEE Board International Test Conference* 179-185.