A Model-Based Testing Process for Enhancing Structural Coverage in Functional Testing

Yanjun Sun, Gérard Memmi and Sylvie Vignes

Abstract Developing complex safety-critical systems usually involves developing models as abstractions in the upstream phases of design. It is still today often challenging to convince the industry that performing functional testing on models of systems may help reducing the cost of system testing. This article presents a new model-based testing process. Part of the "CONNEXION" French I&C methodology project, it combines a vast number of verification tools. In this article, we focus on the integration of a specification-based test generation tool, a model-checker and an environment for model test execution to enhance structural coverage rate. To this end, we define a novel process describing how to extend the functional test bed to enhance structural coverage by generating new test cases reaching so far uncovered branches using model-checking.

1 Introduction

In safety-critical industries (avionic, nuclear, automotive, etc.), systems are required to be developed under standards and certifications [1]. Apart from building the specification, upstream phases of design usually involve developing formal models

This work is founded by "CONNEXION" cluster (a project of French Investment in the Future Program) https://www.cluster-connexion.fr/.

Y. Sun (✉) · G. Memmi · S. Vignes
CNRS LTCI-UMR 5141, Télécom ParisTech, 46 Rue Barrault, 75013 Paris, France
e-mail: yanjun.sun@telecom-paristech.fr

G. Memmi
e-mail: gerard.memmi@telecom-paristech.fr

S. Vignes
e-mail: sylvie.vignesg@telecom-paristech.fr

M.-A. Cardin et al. (eds.), *Complex Systems Design & Management Asia*,
Advances in Intelligent Systems and Computing 426,
DOI 10.1007/978-3-319-29643-2_13

as an abstraction of system. It is cost-effective to perform testing on these models since the cost of bugs found later in the actual system can be extremely high.

In 2012, the main industrial and academic partners of French nuclear industry initiated a large R&D project called "CONNEXION" [2, 3]. One among several objectives of this project is to perform tool-supported automatable verification activities on the formal models built in early stages of I&C system life cycle. This falls into the category of model-based testing.

Traditional model-based testing [4, 5] is a variant of testing that relies on abstract formal models which encode the intended behaviour of a system. Test cases can be automatically generated from models, usually they will be executed on the system under test. In our study, test cases will be directly executed on executable models.

In this article, we present a new general tool-supported model-based testing process to be applied in "CONNEXION" with its specific set of verification tools. More precisely, our process combines a specification-based test generation tool, a model-checker and an environment supporting execution of test cases on models. This process aims at enhancing the structural coverage rate. The first tool generates test cases according to the formal specification to test functional aspects of models. With the help of the model-checker, we extend the functional test bed to enhance the structural coverage of models. This process is designed to address verification of vast portions of an I&C system.

The paper is outlined as follows: Sect. 2 presents the system engineering process in "CONNEXION", Sect. 3 summarizes some basic aspects of testing methodology; Sect. 4 presents a new process for enhancing coverage using a model-checker; Sect. 5 describes the environment built in "CONNEXION" that enables our process; Sect. 6 concludes and discusses future work.

2 Triple-V Cycle in "CONNEXION"

The objectives of the "CONNEXION" cluster concern Control Systems and Operational Technologies to maintain a high level of safety and to offer new services improving the efficiency and the effectiveness of operational activities. In the context of Nuclear Power Plants, an Instrumentation & Control (I&C) system is composed of several hundreds of **Elementary Systems (ES)**, controlling with a very high safety level thousands of remote controlled actuators: about 8000 binary signals and 4000 analog signals sent to the control room concerning over 10 000 I&C sub-functions and over 300 I&C cabinets. An ES is a set of circuits and components which perform an essential function to the operation of the nuclear plant. Each ES is divided into two parts: the **Process**, representing the physical infrastructure (heat exchangers, sluice gates, pipes, etc.) and the **I&C system**, responsible for the protection, control, and supervision of functioning of the process. The functional aspect of an elementary system is described by **Functional Diagram (FD)**, built in a non-executable formal language which complies with IEC standards [6, 7].

The development process of I&C system is illustrated by Fig. 1. The current approach corresponds to a V cycle including phases 1, 2, 4, 7, 8, 9, 10. The process starts with building a **specification of functional requirements** for both process and I&C of one ES. A more detailed functional specification of the I&C system is then deduced in form of a Functional Diagram. This Functional Diagram is progressively elaborated into a **Refined Functional Diagram (RFD)**, ready to be transformed to programs implemented on automata. The Verification and Validation (V&V) of I&C system is performed at the level of automata: first verification of I&C system of one ES with respect to its RFD; then verification of this I&C system integrated with I&C systems of other ES that are already validated. All the validated Elementary Systems are finally integrated through a platform to further validate the proper functioning of these automata in a simulated environment [2].

It has been proposed in "CONNEXION" to introduce two additional verification processes in the upstream phases of design: verification of FD and RFD with respect to functional requirements. This is defined as functional validation in [8]. The original V cycle is therefore enriched by two sub-cycles: 1, 2, 3 and 1, 2, 4, 5, 6. Due to the fact that FD and RFD are formally verifiable but not executable, these verification activities are currently manual. To automate the functional testing, the FD and RFD need to be equivalently recreated in an executable formal language. The partners of "CONNEXION" provides a complete tool chain to automate functional validation as much as possible. The development process is aligned with IEC standard [9], encouraging test automation tools. Our process proposes to use formal tools as test generators not as provers. Our process proposes to assist the manual test production phase but not to substitute it. In such a way, the integration of these tools in our process does not imply a challenging qualification of the tools. The current life cycle relies on a document-centric system engineering approach; "CONNEXION" enables the transition to a model-centric practice as advocated by the INCOSE [10]. The introduction of the two V sub-cycles answers for a key

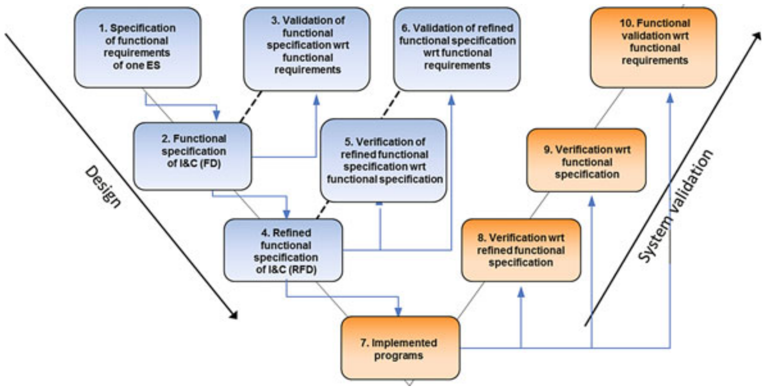

1. Specification of functional requirements of one ES
3. Validation of functional specification wrt functional requirements
6. Validation of refined functional specification wrt functional requirements
10. Functional validation wrt functional requirements
2. Functional specification of I&C (FD)
5. Verification of refined functional specification wrt functional specification
9. Verification wrt functional specification
4. Refined functional specification of I&C (RFD)
8. Verification wrt refined functional specification
7. Implemented programs
Design
System validation

Fig. 1 I&C system life cycle in "CONNEXION"

challenge of Industrial Cyber-Physical Systems: early stage of Verification and Validation [11].

In this article we present a new testing process intended for the two sub-cycles of functional validation. It is designed to be applied first to a single ES and then to scale up to a cluster of ES.

3 Testing Methodology

3.1 Black-Box Testing and White-Box Testing

In general, there are two mainstream testing strategies: black-box testing (or functional testing) and white-box testing (or structural testing).

The goal of black-box testing is to find errors in the program by testing the functions listed in the specification. Designing test set does not require analyzing the details of code but using the specification. On the contrary, white-box testing focuses on the structure of the program. A test set in which every line of code is executed at least once is an example of white-box testing.

A mixture of different strategies can be used to improve the effectiveness of testing. For example, since black-box testing does not require knowledge of the structure of the program, there may be some parts of the program that are unreachable because they are defective or insensitive to certain inputs. These problems may not show up in functional testing.

The "CONNEXION" project looks for functional verification on models of system with respect to the specification of functional requirements. Meanwhile, covering as much of the model structure as possible (or at least some parts) is also an objective. This inspires us to design a testing process that combines black-box and white-box testing.

3.2 Coverage Criteria

Coverage criteria indicate how adequately the testing has been performed. According to the testing strategies presented just above, there are at least two categories of coverage criteria: requirement coverage and structural coverage. We hereby adopt the definitions of different coverage criteria given in [12].

A **requirement** is a testable statement of some functionality that the system must perform. **Requirement coverage** demands that all requirements are covered in the functional test set. In other words, a measurement of the requirements that are covered in the test set indicates how well the functional testing has been performed.

As for **structural coverage**, many criteria have been discussed in the literature. Statement coverage and branch coverage are two of the most widely used criteria in practice. A measurement of statements or branches covered in the test set indicates the test adequacy [13].

Definition 1 (*Statement coverage*) The test set must execute every reachable statement in the program.

Definition 2 (*Decision coverage* (*also called branch coverage*)) The test set must ensure that each reachable decision is made true and false at least once.

The testing of a decision depends on the structure of that decision in terms of conditions: a decision contains one or more conditions combined by logic operators (*and, or* and *not*). Several decision-oriented coverage criteria are hence derived:

Definition 3 (*Multiple condition coverage* (*MCC*)) A test set achieves MCC if it exercises all possible combinations of condition outcomes in each decision. This requires up to 2^N test cases for a decision with N conditions.

Definition 4 (*Modified condition/decision coverage* (*MC/DC*)) A test set achieves MC/DC when each condition in the program is forced to true and to false in a test case where that condition independently affects the outcome of the decision. A condition is shown to independently affect a decision's outcome by varying just that condition while holding fixed all other possible conditions. For a decision containing N conditions, a maximum of 2N test cases are required to reach MC/DC.

Indeed, these different structural coverage criteria are not equally strong. For example if we reach MCC then we reach MC/DC since MCC requires strictly more test cases than MC/DC. Similarly decision coverage is stronger than statement coverage. More detailed information about the hierarchy of coverage criteria can be found in [12].

Requirement coverage and structural coverage are somewhat independent in the sense that a 100 % requirement coverage does not guarantee a 100 % structural coverage and vice versa. In practice, the functional testing of large scale system is usually performed by executing a set of functional test cases in a harness. Often the statement coverage is considered as an indicator of testing effectiveness [14].

In this paper, we consider the branch coverage criterion, but the methodology can be applied to other structural coverage criteria with support of proper tools.

4 A Tool-Supported Model-Based Testing Process

The idea of coverage-based test generation using model-checking has already been studied by a few researchers. Geist et al. [15] proposes using a model-checker to generate a test case that covers certain areas of the program. Ratzaby et al. [16] uses model-checking to perform reachability analysis, i.e. whether certain areas can ever be covered by any test case. In [17], model-checking is used to derive test cases from all uncovered branches to enhance structural coverage. Geist et al. [15] and Ratzaby et al. [16] explains how one test case can be built while [17] presents a procedure for dealing with every uncovered branch. In this paper, we improve the procedure in [17] in particular by a refined termination test and by considering that a model-checker can have a "time-out" situation. In this last case, we propose an hybrid simulation using both a simulation and a model-checker as it is done for hardware design verification (see [18] for instance). Model-checker explores every possible execution whereas hybrid simulation explores only partially the set of all possible execution although driven by user inputs.

As to answer the question "how to design a test case using model-checking", [19] presents a framework where structural coverage criterion MC/DC is formalized as a temporal logic formula used to challenge a model-checker in order to find test cases. Also techniques for using model-checker to generate test cases from coverage criteria including branch coverage and MC/DC are described in [1].

Figure 2 illustrates our testing process and positions three tools that are requested to implement it. A model-based test generation tool is first used to derive a functional test set (*TS*) according to the specification (step 1). This test set is then executed in a proper environment on the model of system (step 2). We suppose that the requirement coverage (*RC*) and structural coverage (*SC*) of the test set are measured after the execution (step 4). Another assumption is that the information

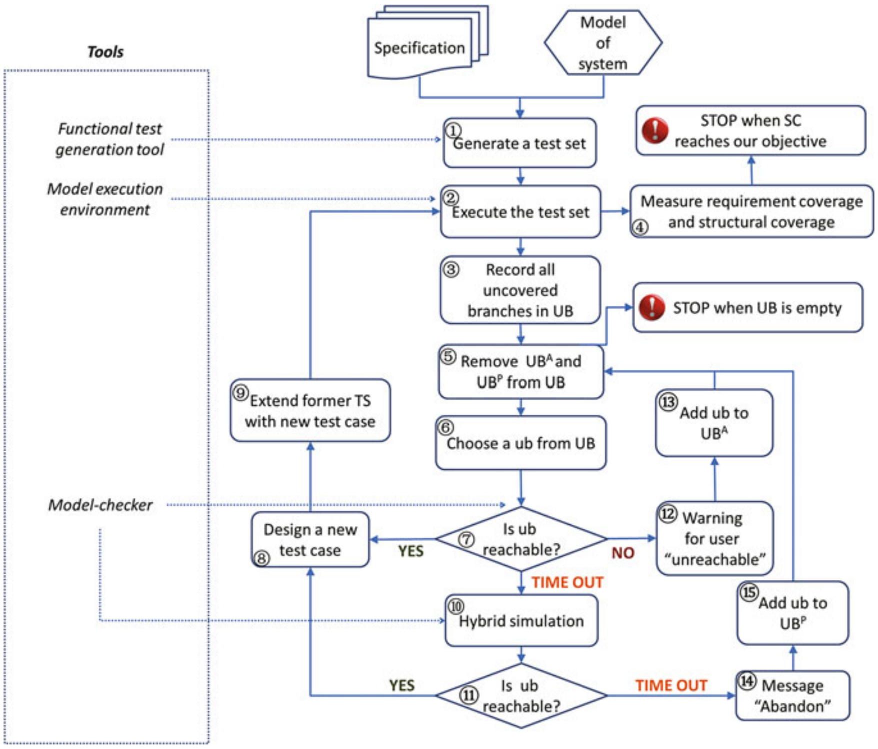


Fig. 2 Model-based testing process

about all uncovered branches in the test set is also provided. We define *UB* as the set of all uncovered branches after the execution of a *TS* (step 3). UB^A is the set of all **actual unreachable** branches and UB^P the set of all **potentially unreachable** branches, for which the method did not succeed to answer the reachability question. Initially they are both empty. Take an uncovered branch *ub* from *UB* (step 6) and apply a model-checker to perform coverability analysis [16] to check if *ub* is reachable (step 7):

- If *ub* is not reachable, send a warning message to user (step 12) and record *ub* as an **actual** unreachable branch (step 13): $UB_j^A = UB_{j-1}^A \cup ub$. Go to step 5 and continue the following process.
- If *ub* is reachable, the model-checker has probably produced a trace of inputs and outputs as a counterexample to the assertion "*ub* is not reachable". Use the trace built by model-checker to design a new test case (*ntc*) that covers this particular branch and possibly others (step 8). Complete the former *TS* with this new test case (step 9): $TS_i = TS_{i-1} \cup ntc$. Go to step 2 and continue the following process.

A third possibility is a "time out" situation: the model-checker takes too much time to decide if a branch is reachable. As a last solution, we propose to apply hybrid simulation as described in [18] to check the reachability of this branch (step 10). If the branch is reachable then go to step 8 and continue the following process. Hybrid simulation can also "time out" which leads to sending an "abandon" message to the user (step 14) and then record *ub* as a **potentially** unreachable branch (step 15): $UB_k^P = UB_{k-1}^P \cup ub$. Go to step 5 and continue the following process. Notice that in step 5, we have $UB_i = UB_i - UB_j^A - UB_k^P$.

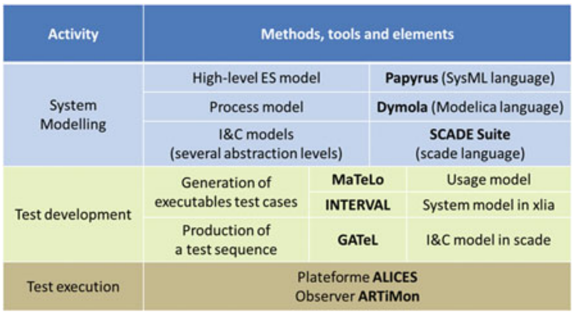
This looping process converges when either the structural coverage reaches our objective or there are no more unexplored uncovered branches i.e. $UB_i = \emptyset$. The convergence of this process is obvious: since $TS_i \supset TS_{i-1}$, that leads to $UB_i \subset UB_{i-1}$ and $SC_i > SC_{i-1}$ because at least one more branch is covered.

It is possible that the process terminates immediately after execution of the first functional test set TS_0 if the corresponding SC_0 already reaches our objective. Otherwise, at the end of this process, if the loop at left is executed at least once, we have an improved test coverage; if the loop at right is executed at least once, i.e. $UB^A \cup UB^P \neq \emptyset$, further analysis is required since at least one branch may be suspected to be dead code or even the manifestation of a bug.

5 Application in "CONNEXION"

According to their purposes, the tools brought by partners of "CONNEXION" can be divided into three categories:

- For modelling the I&C system (build executable models corresponding to FD and RFD);



Activity	Methods, tools and elements		
System Modelling	High-level ES model	**Papyrus** (SysML language)	
	Process model	**Dymola** (Modelica language)	
	I&C models (several abstraction levels)	**SCADE Suite** (scade language)	
Test development	Generation of executables test cases	**MaTeLo**	Usage model
		INTERVAL	System model in xlia
	Production of a test sequence	**GATeL**	I&C model in scade
Test execution	Plateforme **ALICES** Observer **ARTiMon**		

Fig. 3 The tool chain in "CONNEXION"

- For developing test cases;
- For executing test cases.

They are summarized in the following Fig. 3. We present these tools very briefly. More details can be found in the corresponding references.

Papyrus [20] from CEA List: based on platform Eclipse, Papyrus offers an open source graphic editor for modelling in SysML [21]. In "CONNEXION", a system model (both Process and I&C are included) will be created in SysML with Papyrus. This model has a high abstraction level, corresponding to that of the specification of functional requirements.

INTERVAL [20] from CEA List: capable of generating executable test cases. It takes as input the system model in xLia [22], obtained by a semi-automatic transformation of system model in SysML created in Papyrus.

Dymola[1] is a commercial modelling and simulation environment based on open source Modelica[2] language. Dymola is used to modeling the process.

SCADE Suite [23] from ESTEREL Technologies has been chosen as the executable modeling tool. Based on Lustre language [24], SCADE Suite is tailored for designing critical system and its code generator is qualified under several certifications.[3]

Functional testing of the models built in SCADE suite can be as automated as possible with support of proper tools: GATeL [25] from CEA List and MaTeLo [23] from ALL4TECH have been chosen.

GATeL works on models described in Lustre language, therefore is compatible with SCADE models, and performs theoretical proof. It verifies properties that are invariant or characterizations of reachable states of the system. With a test objective expressed in an extended version of Lustre, GATeL generates a trace of inputs which drive the system to a state satisfying the test objective. In the testing process presented in this paper, GATeL will be used as the model-checker.

[1] http://www.3ds.com/products-services/catia/products/dymola.

[2] https://www.modelica.org/.

[3] http://www.esterel-technologies.com/products/scade-suite/.

MaTeLo is a test generation tool for statistical usage testing [26]. It works on a usage model, created manually from the specification of functional requirements. It then generates test cases and provides measurement of the requirements covered by these test cases. In the testing process presented in this paper, MaTeLo will be used as the functional test generation tool.

The test cases generated by MaTeLo do not include the expected outputs (oracle) to be compared with the actual outputs. The oracle is performed by a test observer ARTiMon [20] from CEA List. ARTiMon and the SCADE model are both integrated to a platform ALICES [27] from CORYS, which provides an environment of executing test set on the model.

6 Conclusion

The current design process of I&C systems requires a Functional Diagram and a Refined Functional Diagram before transforming to implemented programs. Verification of FD and RFD with respect to functional requirements are performed manually because FD and RFD are not executable. "CONNEXION" provides a tool chain allowing to create executable models equivalent to FD and RFD as well as to automate functional validation of these models as much as possible.

This paper presents a model-based testing process with the objective of enhancing structural coverage in functional testing. We have seen that this process also allows the detection of suspicious code branches that require analysis to determine whether they are truly unreachable or a bug is occurring in a condition guarding this branch. With support of proper tools, the functional test set can be extended by test cases derived from uncovered branches using model-checking. We consider branch coverage criterion in this article but principles and methodology can be well applied to other structural coverage criteria.

The "CONNEXION" project is recently supplied a unique set of verification tools. This enables our process and makes it applicable to I&C applications. "CONNEXION" has determined a progressively complex case study where our process will be applied. We also imagine mapping the requirement coverage of the functional test set to its structural coverage. This will support test cost reduction by reusing some test cases when some changes are being made to the system.

References

1. Enoiu, E.P., Causevic, A., Ostrand, T.J., Weyuker, E.J., Sundmark, D., Pettersson, P.: Automated test generation using model-checking: an industrial evaluation. In: ICTSS 2013
2. Collective. Cluster CONNEXION: Spécification d'un environnement de verification de la partie contrôle-commande. Livrable 2.1.2 (2014)

3. Devic, C., Morilhat, P.: CONNEXION Contrôle Commande Nucléaire Numérique pour l'Export et la rénovatION—coupler génie logiciel et ingénierie système: source d'innovations. *Génie Logiciel*, 104:2–11, mars (2013)
4. Pretschner, A., Philipps, J.: 10 Methodological Issues in Model-Based Testing. In: Broy, M. et al. (eds.) Model-Based Testing of Reactive Systems, LNCS 3472, pp. 281–291 (2005)
5. Utting, M., Pretschner, A., Legeard, B.: A taxonomy of model-based testing. Working Paper Series (2006)
6. IEC61804-2: Function blocks (FB) for process control—Part 2: Specification of FB concept, 2.0 edition (2006)
7. IEC61131-3: Programmable controllers—Part 3: Programming languages, 3.0 edition (2013)
8. IEC61513: Nuclear power plants—instrumentation and control important to safety—general requirements for systems (2011)
9. IEC60880: Nuclear power plants—instrumentation and control systems important to safety—software aspects for computer-based systems performing category A functions (2006)
10. INCOSE Systems Engineering Vision 2020. INCOSE (2007)
11. Fisher, A., Jacobson, C., Lee, E., Murray, R., Sangiovanni-Vincentelli, A., Scholte, E.: Industrial cyber-physical systems—icyphy. In: Proceedings of the Fourth International Conference on Complex Systems Design & Management, pp. 21–37 (2013)
12. Utting, M., Legeard, B.: Practical Model Based Testing: A Tools Approach. Morgan Kaufmann (2007)
13. Zhu, H., Hall, P.A., May, J.H.: Software unit test coverage and adequacy. ACM Comput. Surv. **29**(4), 366–427 (1997)
14. Piwowarski, P., Ohba, M., Caruso, J.: Coverage Measurement Experience During Function Test. In: ICSE 93 Proceedings of the 15th International Conference on Software Engineering, pp. 287–301 (1993)
15. Geist, D., Farkas, M., Landver, A., Lichtenstein, Y., Ur, S., Wolfsthal, Y.: Coverage-directed test generation using symbolic techniques. Lect. Notes Comput. Sci. **1166**, 143–158 (1996)
16. Ratzaby, G., Ur, S., Wolfsthal, Y.: Coverability Analysis Using Symbolic Model Checking, CHARME 2001. In: Lectured Notes in Computer Science, vol. 2144. Springer (2001)
17. Fantechi, A., Gnesi, S., Maggiore, A.: Enhancing Test Coverage by Back-tracking Model-checker Counterexamples. In: Electronic Notes in Theoretical Computer Science, vol. 116, pp. 199–211 (2004)
18. Memmi, G.: Integrated circuits analysis, system and method using model-checking. US Patent 7493247 (2009)
19. Rayadurgam, S., Heimdahl, M.P.: Coverage Based Test-Case Generation using Model Checkers. IEEE (2001)
20. Gallois, J., Pierron, J., Rapin, N.: Validation test production assistance. ICCSEA 2013
21. A practical guide to SysML: The Systems Modeling Language. Morgan Kaufmann/OMG Press (2011)
22. Gaudin, E.: Automatic test generation based on functional coverage. UCAAT (2014)
23. Chastrette, F., Vallee, F., Coyette, L.: Application of model-based testing to validation of new nuclear I&C architecture. ICCSEA 2013
24. Halbwachs, N., Caspi, P., Raymond, P., Pilaud, D.: The synchronous data-flow programming language LUSTRE. Proc. IEEE **79**, 1305–1320 (1991)
25. Marre, B., Arnould, A.: Test Sequences generation from LUSTRE Descriptions: GATEL. In: 15th IEEE Conference on Automated SW Engineering, pp. 47–60 (2000)
26. LeGuen, H., Thelin, T.: Practical Experiences with Statistical Usage Testing. In: Proceedings of the Eleventh Annual International Workshop on Software Technology and Engineering Practice (STEP'04)
27. Neyret, M., Dormoy, F., Blanchon, J.: Méthodologie de validation des spécification fonctionnelles du contrôle-commande—Application au cas d'étude du Système de Réfrigération intermédiaire (SRI). Génie Logiciel, hors-séries:12–25, mai (2014)