CoFI: a Test Process for Space Application Service Validation

Ana Maria Ambrosio[1]

[1]Ground Systems Development Division (DSS)
ana@dss.inpe.br

Eliane Martins[3]

[3]Institute of Computing (IC)
State University of Campinas (UNICAMP)
eliane@ic.unicamp.br

Nandamudi L. Vijaykumar[2]
Solon V. de Carvalho[2]

[2]Associated Laboratory of Computing and Applied Mathematics (LAC)
National Institute for Space Research (INPE)
{solon, vijay}@lac.inpe.br

Abstract

This paper presents a test process named CoFI (Conformance and Fault Injection), which has being defined as part of the doctorate program. CoFI integrates two existing test approaches: conformance test and fault injection technique. It was conceived for validating the space communication services being standardized by ESA. For the test case specification activity this process defines an approach for automatically generating test and fault cases. The main idea behind this process is to create a re-usable abstract test and fault suite that may help to improve the effectiveness of testing and allow the space communication services evaluation under faults. The CoFI is partially illustrated for the telecommand verification service stated in the ESA standard for the Telemetry and Telecommand Packet Utilization (PUS) services.

Keywords: testing process, conformance test, software-implemented fault injection, space software validation.

1. Introduction

In the 80's software testing was widely believed to be one of the phases within the software development life cycle, according to the software engineering concepts. However, nowadays with the increase of the complexity and importance of the computational systems, test has gained much importance to act during all the phases of the software development life cycle. The software development life cycle adopted by ESA includes a verification approach whose test activities closely parallel each development phase [5]. Tests are crucial for a space application software to assure the success of the mission. Reports summarized in [16] illustrates space mission fails caused mainly by ill-designed system and software validation.

Software developed for complying with a standard specification are generally implemented by different manufacturers, so, these implementations must be tested against the specification for conformance evaluation. This activity is known as conformance testing. The conformance test has been strongly explored in the telecommunication for the protocol certification [8] and [9]. In academia, much literature has also been published [2], [18], [4], [10], [11], etc. However, there are yet insufficient techniques to create tests from a specification. Moreover, the tests may consume up to 50% of the project resources in whole software development project [19].

In order to reduce the test effort and yet to improve the effectiveness of the test cases for the evaluation of the packet utilization standard services given in [6], we propose a test process which includes an approach to help generate test cases covering external failure situations. The fault injection is an important technique for testing space applications, as it may represent common space environment faults under which such kind of systems are submitted when in its

operational phase [14]. Prototype tools developed in previous research supports partly the process automation. [15], [20].

The paper is organized as follows: section 2 presents the testing process; section 3, summarizes the approach for test and fault case generation; in section 4 a case study illustrates the CoFI and finally section5 concludes the paper pointing out future works.

2. Testing Process

A testing process is a "course of action to be taken to perform testing"[1]. [3] presents a testing process, whose activities (testing planning, design, test case development, software development, execution and result analysis) are compared with the software development activities (requirement analysis, preliminary design, detailed design and coding/unit test). Figure 1 summarizes this comparison.

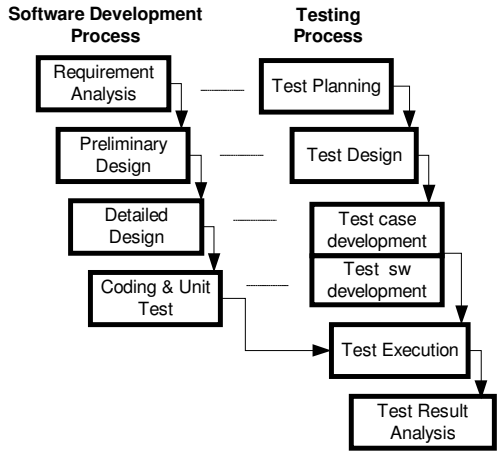

Figure 1: Software Development vs. Testing Process

In telecommunication, the conformance testing process, for ISO protocol certification, is standardized in [8]. The CoFI process presented in this article, extends the evaluation power of the functional conformance (given in the IS-9646 [8]) with the software implemented fault injection technique to accelerate the exceptional events. Furthermore, the test activities are closely related to the application to be tested. Figure 2 summarizes the activities flow and draws the parallels between the CoFI process activities and those activities defined in Drabick. A description of the CoFI activities, the required, and the created artifacts are given below.

[1] According to the process definition in IEEE Software Standards.

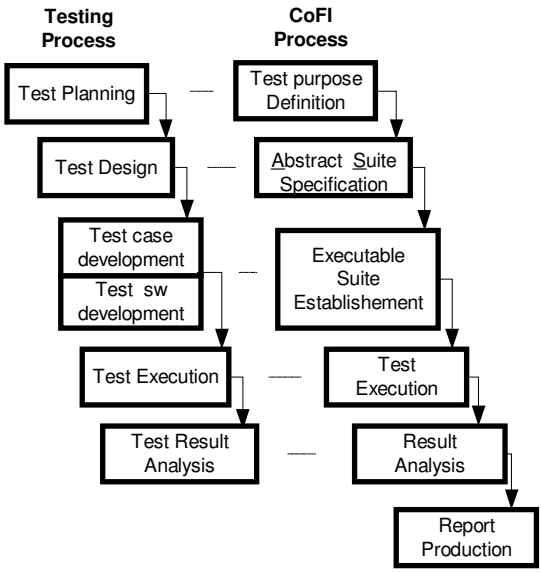

Figure 2: Testing Process vs. CoFI process

Test Purpose definition: This activity consists in defining the test purposes (or test objective) based on the service standard specification description. (See Figure 3)

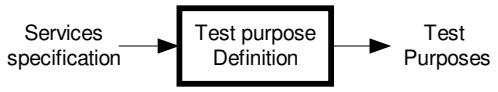

Figure 3: Test Purpose Definition activity

Abstract Suite Specification: In this activity an abstract test suite (ATS) is specified for each test purpose. The ATS comprises implementation-independent *test* and *fault cases*. Both test and fault cases are limited to the test architecture, which strongly affects the conformance requirements that can effectively be checked by the testing means. Furthermore, specification is required to provide assumptions about the service observability and testability (the definition of the points of control and observation (PCO)). An *external fault* list helps defining the fault cases. The abstract suite is written in a *test notation,* either in the ISO standard TTCN[2] or in a language supported by an automated testing tool. Every test/fault case is associated to an *expected output*. Looking at the process automation, the test and fault cases are derived from formal methods. Consequently the textual description of the services must be translated into a formal specification model. The test approach is presented in section 3 in a step-by-step manner. (See Figure 4)

[2] In earlier versions, TTCN stands for Tree and Tabular Combined Notation. The new meaning is Testing and Test Control Notation apt for TTCN-3.

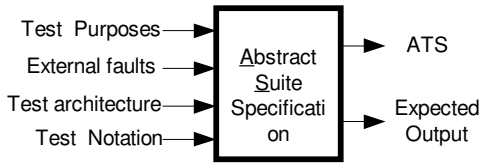

Figure 4: ATS Specification activity

Executable Suite Establishment: This activity is characterized by the choice, from the ATS, of the test/fault cases to be applied to the implementation under test. The available test architecture influences the test selection. Once the tests are selected, they are completed and parameterized with the provided implementation details and information like values of counters and timers reported in both the Implementation Conformance Statement (ICS) and the Implementation extra Information for Testing (IXIT) documents. The parameterized test set is named *parameterized executable test suite* (PETS). In CoFI, the PETS are supposed to be written in PLUTO [7], or any script-like language. (See Figure 5)

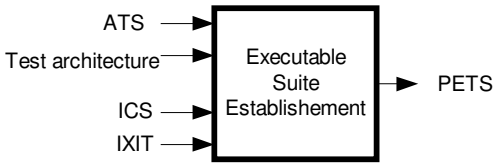

Figure 5: Executable Suite Establishement activity

Test Execution: In this activity the PETS are already implemented and the test campaigns are carried out over the implementation under test. The test architecture is supposed to provide the necessary tools for the test running including the fault injections. All the observable reactions of the implementation are observed for both normal and exceptional situations. All inputs, outputs and other test events, i.e., the read-outs produced during the test campaigns are logged not only for the result analysis but also for future reference. (See Figure 6)

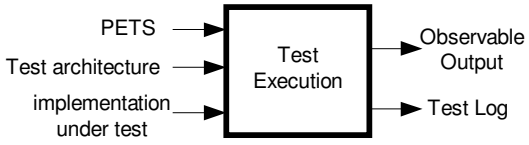

Figure 6: Test Execution activity

Result Analysis: This activity consists in comparing the *observable output,* obtained during the test campaigns, with the *expected outputs* created during the test and fault cases (ATS) generation in the *Abstract Suite Specification* activity. (see Figure 7)

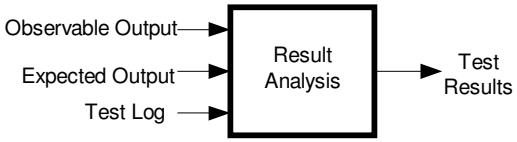

Figure 7: Result Analysis activity

Report Production: This activity consists in attributing to each executed test/fault case, one of the following results: pass, fail, inconclusive, error in the abstract or executable test case or abnormal test case termination. This classification of the test results is according to the IS-9646 definition. The complete analysis of the test results leads to a verdict related to the conformance requirements. Thanks to the *expected output* obtained from the service formal behavior model one may infer the verdicts about the test. The verdicts are not only related to the implementation behavior under normal events but also under external faults. Furthermore, *statistics* about the test execution may be computed and included to the *test report* for diagnostics, once the comparison between the observable and the expected outputs is performed. (See Figure 8).

Figure 8: Report Production activity

3. Approach for ATS specification

The approach to derive the ATS guides the tester to translate the service specification in a Finite State Machine.

The formal notation-based specification allows the use of formal algorithms to automatically generate test cases, so reducing time and efforts with testing and assuring great coverage of the specification with the tests. However, to get the PUS specification in a formal notation acceptable by the algorithms is a hard and lengthy work, requiring many iterations [17].

In order to facilitate generating the formal specification, the CoFI approach specifies a set of steps making use of UML notation. The test cases are obtained from normal situation scenarios, whereas the fault cases are obtained from the exceptional situations. In separating normal from exceptional part, the number of cases generated each time is smaller than if the global modeling were created. In short, the main steps of the CoFI approach are:

1. describe the service as an use case: identify the requests, the reports and *operational variables* [1] handled by the service;
2. create *normal* and *exceptional scenarios* deduced from the operational variables whose variation of values causes a service behavior change;
3. design *Sequence Diagrams* one mapping all the normal scenarios and other the exceptional;
4. create an *event/state matrix*: where a line is an event (valid, invalid and inopportune are represented by requests or operational variable); a column is a state, a cell is both an output (normal or abnormal explicitly specified, abnormal terminations) and the next state;
5. create a *Normal Finite State Diagram* with the events triggering the explicitly specified normal outputs;
6. generate the *test cases* submitting the FSM to a tool, for example Condado [14];
7. create an *Exception Finite State Diagram* mapping only the events triggering the explicitly specified exception outputs;
8. generate the *fault cases*, by submitting the exception FSM to algorithms of graph traversing [12] for defining the set of transitions necessary to put implementation in the state in which the fault has to be injected. Then the fault is identified in the transitions leaving this state. The fault must be executed by special components of the test architecture. The fault is characterized by parameters: *when* (time or message identification), *what* (fault type), *how* (mask or byte position), *where* (the communication fault is associated with a PCO).

4. Case Study

The PUS standard ECSS-E-7041[A], specifies services to be provided by the satellite at the application process layer during its communication with the ground control system (service client).

The *telecommand verification* service provides feedback about the execution of long execution duration telecommands. The stages of such deep space mission telecommands may spend hours, so they may pass for: reception, execution start, execution progress and execution completion stages. Then, on requiring this service, the client should indicate, the reports he or she wants, in the Ack bits.

On applying the CoFI approach, one finds the test purpose *as to verify the telecommand execution.* The ATS is obtained on applying the approach presented in section 3, which are:

1. the request and reports are directly obtained in the capability set [6]; the operational variables are obtained from the Ack bits, whose four-bit combination reflects the ground-board communication operational scenarios. They are The use case description is not presented here;
2. As the normal scenarios are based on the values of the four Ack bits, there are 16 normal scenarios.; Concerning the exception reports, there are four scenarios. All represent the replies sent to the ground system;
3. In mapping the scenarios to sequence diagrams, we assumed three PCOs for this service: PCO_1 (observes the telecommand application data arrival), PCO_2 (allows to recognize the status of the telecommand execution) and PCO_3 (allows to observe the generated telemetry reports to be sent to the service client).
4. Steps 3 and 4 are not shown here;
5. The Normal Finite State Machine is in Figure 9.
6. On submitting the FSM shown in FIGURE 2 to the Condado tool [14] 38 test cases were generated. One test case in the Condado output format is in Table 1. The Condado deals with only two points of control (the U -Upper tester and L-lower tester) and this service requires three points, then the solution is to substitute L for PCO_1 whenever the next message is Tcarriv or receiveTC, or for PCO_3 in the other cases. The U substitutes the PCO_2. The word senddata and recdata mean respectively send and receive the data from PCO. The same test case is shown in TTCN notation in Table 2.
7. The Exception Finite State Machine is in Figure 10.
8. One fault case is illustrated in TTCN in Table 3.

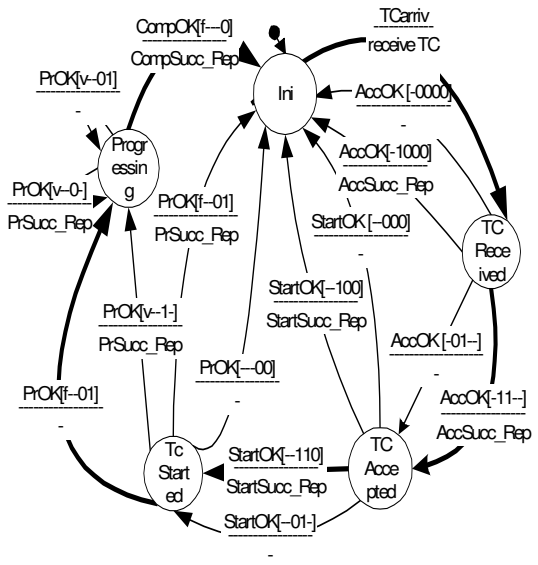

Figure 9 – Normal Finite State Machine

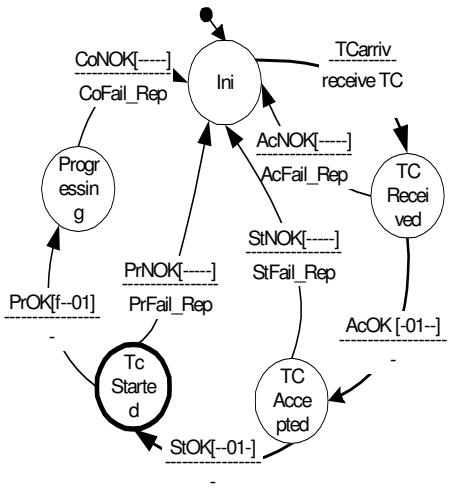

Figure 10 – Exception Finite State Machine

Table 1- Test case – Condado output

Input	Output
senddata(L,TCarriv)	recdata(L,receiveTC)
senddata(U,AccOK[-11--])	recdata(L,AccSucc_ Report)
senddata(U,StOK[--110])	recdata(L,StartSucc_ Report)
senddata(U,PrOK[f--01])	recdata(L,)
senddata(U,CompOK[f---0])	recdata(L,CompSucc _Report)

Table 2. Test case – TTCN notation

Behavior Description	Constraint
PCO_1 ? TCarriv	
PCO_2 ? AccOk PCO_3 ! AccSuc_Rep	[Ackbits = 1---]
PCO_2 ? StartOk PCO_3 ! StartSuc_Rep	[Ackbits = -1--]
PCO_2 ? ProgOk	[Nstep =1 = Max] [Ackbits = --01]
PCO_2 ? CompOk PCO_3 ! CompSuc_Rep	[Nstep =1 = Max] [Ackbits = ---1]

Table 3. Fault Case – TTCN notation

Behavior Description	Constraint
PCO_1 ? TCarriv	
PCO_2 ? AccOk	[Ackbits = 0---]
PCO_2 ? StartOk PCO_3 ! StartSuc_Rep	[Ackbits = -1--]
Fault:** ***PCO_2 ? ProgNOk PCO_3 ! ProgFail_Rep	

5. Comments and Future Plans

This paper presented a test process named CoFI – Conformance and-fault-injection which is based on the conformance test process standardized in IS-9646, over which test activities with software-implemented fault injection (SWIFI) were included. An approach to generate a set of abstract (implementation-independent) test and fault cases is included in the process activities looking at automation needs. The Telecommand Verification service defined in the ECSS-E-70-41^A PUS standard, prepared by the European Space Agency was used to illustrate the process.

The test approach presented here does not assure only efficiency of the testing activities but also improves confidence in the space application of PUS services. Additionally, it offers a real opportunity to reduce the testing costs in a mission once it guides the design of the test cases that are still largely performed by testers by interpreting specification written in natural language.

Although the general conception for the fault case generation has been included into the CoFI process, the formal definition of exceptional finite state machine and the algorithms for automatically generating fault cases are in progressing.

References

[1] Binder, R. Testing object-oriented systems Models, patterns and tools. Boston: Addison-Wesley, 2000. 1191p.(ISBN 0-201-80938-9).

[2] Bochmann, G.; Petrenko, A. Protocol Testing: Review of Methods and Relevance for Software Testing. International Symposium on Software Testing and Analysis. August 17-19, 1994. Seatle, Washington, USA. Proceedings... 1994. p.109-124.

[3] Drabick, R. D. Best Practices for the Formal Software Testing Process - A menu of Testing Tasks. Dorset House, 2004.

[4] Dssouli, H.; Salek, K.; Aboulhamid, E ; En-Nouaary, A ; Bourhfir, C. Test Development for Communication Protocols: Towards Automation. Computer Networks 31, 1999. p.1835-1872.

[5] European Space Agency (ESA). ESA Software Engineering standards. PSS-05-0 Issue 2. February 1991.

[6] European Space Agency (ESA). European Cooperation for Space Standardization (ECSS) Space Engineering –Ground Systems and Operations: Telemetry and Telecommand Packet Utilization.. ECSS-E-70-41 A. January, 2003.

[7] European Space Agency (ESA). Cooperation for Space Standardization (ECSS). Space Engineering Procedure Language for User in Test and Operations:PLUTO. ECSS-E-70-32A. European Issue Draft 5, 2001

[8] International Organization for Standardization/International Electrotechnical Commission (ISO/IEC). ISO/IEC-9646 Information Technology/ International Standard Conformance Testing Methodology and Framework. Geneve, 1991.

[9] International Telecommunication Union - Telecommunication Standardization Sector of ITU (ITU-T). Recomendation X.290 – OSI Conformance Testing Methodology and Framework for Protocol recommendations for ITU-T Applications – General Concepts. April, 1995.

[10] Kim, Y.G.; Hong, H.S.; Cho, S.M.; Bae, D.H.; Cha S.D. – Test Case Generation from UML State Diagrams – IEE Proceedings software, V. 146, N. 4, , Aug. 1999, 187-192.

[11] Lai, R. – A survey of communication protocol testing – The Journal of Systems and Software, 62, 2002, pp. 21-46.

[12] Lee, D.; Yannakakis, M. Principles and Methods of Testing Finite State Machines – a Survey – Proceedings IEEE, 84 (8): 1090-1123. 1996.

[13] Madeira, H.; Some, R. R.; Moreira, F.; Costa, D.; Rennels, D. Experimental evaluation of a COTS system for space applications. International Conference on Dependable Systems and Networks (DSN'02), June 23 - 26, 2002, Washington, D.C., USA. (http://computer.org/proceedings/dsn/1597/15970325abs.htm).

[14] Martins, E. Sabião, S.B.; Ambrosio, A. M. - ConData: a Tool for Automating Sapecification-based Test Case Generation for Communication Systems. Software Quality Journal, 8 (4) (1999) 303-319.

[15] Martins, E.; Ambrosio, A.M; Mattiello-Francisco M.F. ATIFS: a testing toolset with software fault injection. In: Workshop UK Testing Research, 2. (SoftTest), 4-5 September 2003, York, England. Proceedings... York: Department of Computer Science/University of York, 2003.

[16] Mazza C. Standards: the Foundations for Space IT. Workshop on Space Information Technology in the 21st Century, Darmstadt, Germany, 27 sep. 2000. Available in the digital library URLib: <www.esoc.esa.de/pr/documents/workshops/it_2000/it_in_future/esa_c_mazza.ppt >. Access in:02 sep. 2002.

[17] Merri, M. ; Rüting, J.; Schurman, P. Validation of the ESA Packet Utilization Standard by object-Oriented Analysis. In: International Conference on Space Operations, 4, 1996. (SpaceOPs 96), 16-20 September 1996, Munich, Germany. **Proceedings**.... 1 CD-ROM. Available in the digital library URLib: <www.spaceops1996.org>. Access in: 2 fev.2004.

[18] Tan, Q.M.; Petrenko, A.; Bochmann, G. A Test Generation Tool for Specifications in the Form of State Machines. Proceedings of the International Communications Conference ICC 96, Texas, 1996, 225-229.

[19] Tretmans J.; Belinfante, A. Automatic Testing with Formal Methods. Conference on Software Testing Analysis and Review, 7. (EuroSTAR´99), 8-12 November,1999,, Barcelona, Spain. In: **Proceedings**... 1999.

[20] ATIFS project. Available in the digital library URLib: In: <http://www.inpe.br/atifs>. Access in: 5 sep.2002.