MINISTÉRIO DA CIÊNCIA E TECNOLOGIA
INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS

INPE-15078-PRE/9987

DESIGNING FAULT INJECTION EXPERIMENTS USING STATE-BASED MODEL TO TEST A SPACE SOFTWARE

Ana Maria Ambrosio
Maria de Fátima Mattiello Francisco
Valdivino A. Santiago Jr.
Wendell P. Silva
Eliane Martins *

*UNICAMP- Instituto de Computação (IC)

Presented at Third Latin-American Symposium on Dependable Computing (LADC), Morelia, México, Sept. 26-28, 2007.

INPE
São José dos Campos
2007

Publicado por: **esta página é responsabilidade do SID**

Instituto Nacional de Pesquisas Espaciais (INPE)
Gabinete do Diretor – (GB)
Serviço de Informação e Documentação (SID)
Caixa Postal 515 – CEP 12.245-970
São José dos Campos – SP – Brasil
Tel.: (012) 3945-6911
Fax: (012) 3945-6919
E-mail: pubtc@sid.inpe.br

Solicita-se intercâmbio
We ask for exchange

MINISTÉRIO DA CIÊNCIA E TECNOLOGIA
INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS

INPE-15078-PRE/9987

DESIGNING FAULT INJECTION EXPERIMENTS USING STATE-BASED MODEL TO TEST A SPACE SOFTWARE

Ana Maria Ambrosio
Maria de Fátima Mattiello Francisco
Valdivino A. Santiago Jr.
Wendell P. Silva
Eliane Martins *

*UNICAMP- Instituto de Computação (IC)

Presented at Third Latin-American Symposium on Dependable Computing (LADC), Morelia, México, Sept. 26-28, 2007.

INPE
São José dos Campos
2007

Published in Dependable Computing. Editors: Bondavali, A.; Brasileiro, F.; Rajsbaum, S. Springer, Berlin. 2007; pages 170-178. Third Latin-American Symposium on Dependable Computing (LADC). Morelia, México, Sep. 26-28, 2007.

Designing Fault Injection Experiments using State-based Model to test a Space Software

Ana Maria Ambrosio[1], Fátima Mattiello-Francisco [1],
Valdivino A. Santiago Jr[1], Wendell P. Silva[1], Eliane Martins[2]

[1]National Institute for Space Research (INPE)
Av. Dos Astronautas, 1758 - São Jose Campos -12227-010 - Brazil
Phone: +55-12-3945-6586,
(ana, fatima) @dss.inpe.br, (valdivino, wendell) @das.inpe.br

[2]Institute of Computing (IC)
State University of Campinas (UNICAMP)
P.O. Box 6176 - Campinas, 13083-970, SP, Brazil
eliane@ic.unicamp.br

Abstract. Software for space applications requires significant testing. This paper presents an evaluation of the CoFI testing methodology as applied to actual space software, where deterministic fault cases derived from state-based models were executed using the software-implemented fault injection technique. Different models were used to represent the behavior of embedded software in a real satellite computer under the presence of both normal inputs and external faults in communication, processor, and memory. CoFI methodology was used for model construction, the Condado tool for test derivation, and the QSEE-TAS tool for test execution. In total, 8,620% of 471 fault cases detected errors in the software; this is a very large number, and more so considering that the software had already been tested by the company which developed it before being subject the CoFI methodology.

Keywords: deterministic fault injection, software testing method, state-based models.

1 Introduction

The testing phase in software development lifecycle has attracted software engineer attention to answer the question, "how can one test a complex embedded software in a short time without losing testing accuracy?"

Model-based test techniques have been used for protocol conformance testing to complement the ISO practical testing guides, checking the implementation with respect to a specification written in a formal notation [12], from which tests are automatically generated [5], [8], [18].

A set of conformance test cases aims to establish that a given Implementation Under Test (IUT): (i) performs all functions of the original specification over the full

range of parameter values and (ii) can properly reject erroneous inputs in such a way that it is consistent with the original specification [11]. These test cases generate a certain number of detected errors, but for dependability assessment, fault injection methods are recommended. Fault injection execution is an activity highly dependent on the facilities provided by the test environment [3], [6] and constraints in test execution impose constraints in test generation. The CoFI (Conformance and Fault Injection) testing methodology [1] was designed to help determine which faults to inject using the same principles as model-based techniques "starting from a textual specification towards formal models" [13]. Thus, CoFI reinforces the systematic derivation of test cases that may be executed with software-implemented fault injection (SWIFI).

This article presents the results of the use of CoFI to define which tests should be generated to validate the SWPDC (SoftWare embedded into the Payload Data Handling Computer (PDC)) that is intended to be part of a scientific X-ray instrument onboard of the MIRAX satellite under development at the National Institute for Space Research (INPE), Brazil. This software was developed by a private company and delivered to INPE as part of INPE's Quality of Space Application Embedded Software (QSEE) research project of the [15], [16].

The paper is organized as follows. Section 2 presents an overview of the SWPDC. Section 3 shows the testing tools that were used. Section 4 explains the CoFI methodology applied to the SWPDC. Section 5 discusses the test results. Finally, Section 6 presents pertinent conclusions.

2 Overview of the SWPDC

Figure 1 illustrates the SWPDC software in charge of collecting scientific data from the Event Pre-Processors (EPPs); executing commands from the main on-board computer (OBDH); generating housekeeping data; performing data memory management, loading programs, and detecting external faults that can occur at anytime, as is typical in computer space systems.

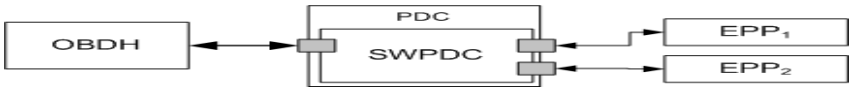

Fig. 1. Context of the SWPDC.

Given that the SWPDC is a software embedded in a satellite computer, it is exposed to space radiation, which may cause Single-Event Effects (SEEs) like the Single Event Upset (SEU) and Multiple Bit Upset (MBU). A single bit flip in a digital device is an example of SEU. When several memory bits are upset during the passage of the same particle it is a MBU [10].

The SWPDC also implements error detection mechanisms for Single and Double memory errors, which are "soft" bit errors, in that a reset or rewriting of the device causes normal behavior thereafter.

To detect processor errors the SWPDC is linked to a Watchdog circuit. A watchdog circuit is a computer hardware-timing device that indicates a problem if the

software neglects to regularly reset the circuit. Exception handling mechanisms exist to treat communication faults. No complex action to treat such errors is required; however, all errors that occur are reported via housekeeping data transmitted to the Ground System.

3 Test Environment

For the sake of validation, the SWPDC was treated as a black-box whose interactions with the test environment are only through Points of Control and Observation (PCOs). Figure 2 illustrates the test environment where the circles around the SWPDC box indicate the PCO's. The external inputs were all simulated. The dashed arrow from the Watchdog Circuit Simulation and to the Watchdog Error Simulation denotes the SWPDC did not send the watchdog timer signal within the expected period of time. A special circuit triggers Simple and Double memory errors, while another circuit controls the temperature. The EEP Simulator generates the scientific data and the QSEE-TAS (*Automatic Software Testing*) tool [17] simulates the OBDH.

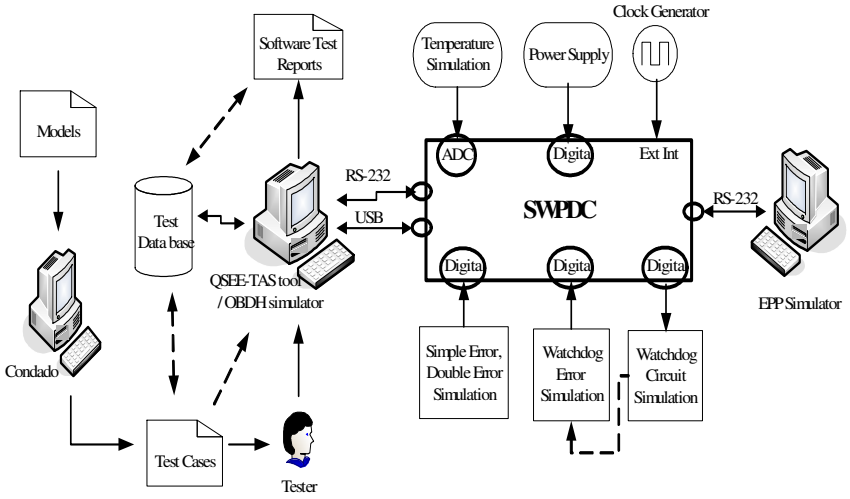

Fig. 2. The Test Environment. Legend: Ext Int = External Interruption; USB = Universal Serial Bus; ADC = Analog-to-Digital Converter.

The QSEE-TAS tool also includes facilities for test configuration, execution, reports, management of the test cases produced by Condado or produced manually, and SWIFI mechanisms that accelerate the occurrence of communication faults in commands produced by the OBDH. This mechanism assigns unspecified and/or incorrect values to fields of the commands to corrupt messages, repeat or delay

commands. So far, injection of memory and processor faults has not been automated, so the tester manually interfered in the respective PCO to trigger these types of faults.

The Condado tool [14] automatically derives test cases from state-based models. This tool is based on a theoretical approach of graphs and implements the switch-cover algorithm [7]. Since Condado generates all test cases in the same format: "`senddata (pco,input1) recdata(pco, output1) senddata (pco,input2) recdata()…`", a converter that takes *specific inputs* (indicating faults to be injected) of the fault cases and produces pre-defined faults was built, thereby permitting QSEE-TAS to execute the test cases produced by Condado directly.

4 CoFI Testing Methodology Applied to the SWPDC Software

CoFI systematizes the creation of partial models of IUT behavior that are employed in automated test methods to generate test cases. In other words, instead of designing a very complex model of software behavior under normal and faulty inputs, which could lead to an explosion of the number of test cases produced from this model, several simpler models are built. The behavior of an IUT is modeled for each service the IUT provides. Scenarios for normal and exceptional behavior are mapped into several state-based models [2], taking into account the *fault types* (or the *fault model*, the term used by the Fault Tolerance community), which describe the way the hardware or software component can fail, an important step for fault injection purposes.

4.1 Creating the SWPDC Models

In this study, we identified the SWPDC inputs that could be executed in the test environment as the commands that characterize the IUT's services. Inputs that could not be executed were not considered, such as duplication and delay in commands coming from an EPP. Next, we defined a syntax for the inputs and outputs used in the models. An **input** carries information on command, channel (the physical representation of the PCO), and faults. Inputs preceded by *Cmd* indicate commands arriving from the OBDH, so the PCO was defined implicitly. The symbol {*badcks*} indicates the injection of a checksum error, while {*sup*} indicates the suppression of a field from the command. Inputs with no faults are all the commands of the PDC-OBDH and PDC-EPP communication protocols (see Figure 1).

Specific inputs indicate the faults to be triggered by the QSEE-TAS tool. The following information may be obtained from such inputs: a) channel-identification; b) number of times the command is repeated; c) delay time (in milliseconds) to wait before sending the command; d) special processing (to calculate checksum or to suppress command fields). Table 1 presents all the fault types accounted for in SWPDC; and sample inputs are also described for each fault type.

Eleven services were identified for the construction of the state-based models. The SWPDC service behavior was represented in scenarios for normal situations (Norm);

specified-exceptional situations (SExc); sneak paths (SPat)1; the presence of the communication faults such as command corruption, truncated and delay/early commands (Com); and the presence of memory and processor faults (M&Pr).

Table. 1. Fault types covered by the SWPDC and sample of specific inputs.

	Fault Type	Examples of specific inputs	Input description
Communication	Corrupted data field values	CmdTurnOnEPP2,CKS{badcks}	The *Turn On EPP2* command will have an error in checksum field.
		CmdPrepMemoryDumpData,Mem,18,EndI,8000,EndF,FFFF	The *Prepare Memory Dump* command will have an error in the address field.
	Repeated command	CmdTransTestData_2X	The *Transmit Test Data* command will be received twice.Indicates a duplication error.
	Out-of-order commands	-	Commands are sent in an unexpected sequence.
	Truncated – command fields are missing	CmdTurnOffEPP1,NU,{sup}	The third field in *Turn Off EPP1* command will be suppressed.
	Delay/Early – command arriving after/before the specified time	ObsEndT	T time-units will expire. This input is preceded by an action to start a timer in T time-units.
Memory	Simple error	ObsSingleError	A Single Event Upset will occur
	Double error	ObsDoubleError	A Multiple (double) Event Upset will occur
Process	First occurrence of Process-fault	ObsErrorProc1	First indication of the watchdog
	Second occurrence of Process-fault	ObsErrorProc2	Second consecutive indication of the watchdog

[1] A sneak path [4] is a path in the model that contains unlikely inputs for a given state. To help identify sneak path scenarios the tester creates a state table and completes it with the valid inputs against all states, then, create models that represent out-of-order and duplicated commands, which are two common types of communication faults.

Table 2 lists the services and the distribution of the 97 models by services and by scenario type. In general, each set of faults of the same type was mapped in a distinct model, except for memory and processor faults. The grey columns indicate the models that produced fault cases.

Table. 2. Services x models.

Services		Models					Total
		Norm	**SExc**	**SPat**	**Com**	**M&Pr**	
S1	Initialization	2	1	1	1	1	6
S2	Scientific data	2	2	1	1	1	7
S3	Housekeeping	3	3	3	1	1	11
S4	Test data	2	4	4	1	1	12
S5	Diagnostics	2	4	4	2	2	14
S6	Memory dump	5	3	5	2	1	16
S7	Change operat mode	1	0	0	0	1	2
S8	Load&execute program	1	5	4	3	2	15
S9	OBDH msg syntax	1	0	0	1	0	2
S10	EPP msg syntax	1	0	0	1	0	2
S11	Special commands	4	0	0	2	4	10
Total		24	24	22	13	14	97

The model of the single scenario for M&Pro of the S4-Test Data service is illustrated in Figure 3. This model shows that under the presence of one memory error (represented by the specific input *ObsSingleError*) the SWPDC reacts by correcting the error (represented by the *ObsCorrectError* output), reporting the event in housekeeping reports (*ObsWriteHkReport*) and remaining in the same state. But, under a double memory error (*ObsDoubleError*), where SWPDC is not required to correct the error, only a report is produced. In the presence of the first occurrence of a processor error (*ObsErroProc1*), it reports the failure, but in the second occurrence (*ObsErroProc2*), a reset makes the SWPDC return to its initial state (*Standby*).

5 The Fault Injection Experiments

Each model was submitted to the Condado tool. In the IUT models a transition represented an input and the expected corresponding output produced in reaction to that input [11]. This means that the test cases generated by Condado are ordered sets of inputs and outputs, comprising a path from the initial state to a final state. The set of test cases, therefore, covered all branches of each model at least once.

External faults added to the set of inputs normally accepted by the SWPDC define the generation of *fault cases*, which have specific inputs and input data that characterize the fault to be injected, so each fault case is considered a fault injection experiment.

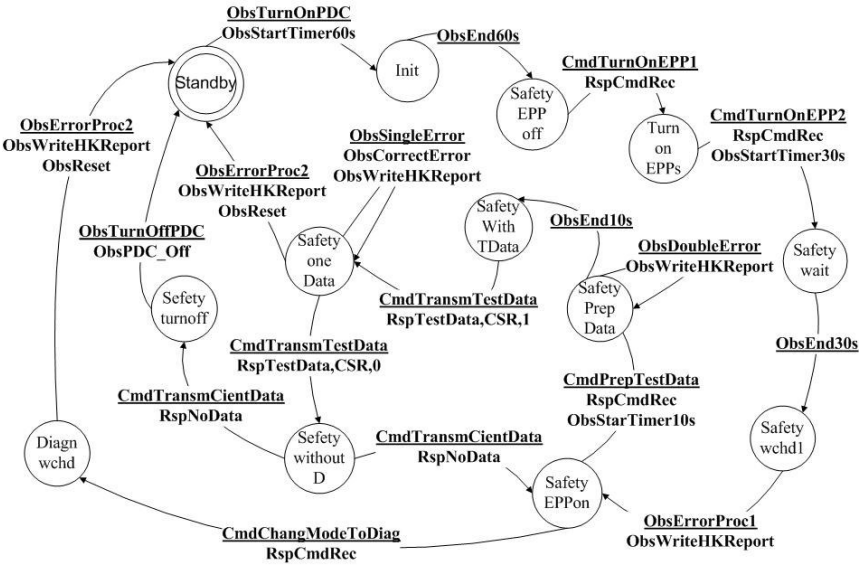

Fig. 3. State-based model representing processor and memory faults.

Table 3 shows the distribution of the errors detected by the fault cases in one campaign. In total, 451 fault cases were generated in 770 test cases produced from the models, resulting in 39 detected errors. Processor and memory faults were modeled in a single model (M&Pr), but fault cases and the errors detected were computed separately. There were 2 more processor errors than memory errors detected. The fault cases of communication were derived from the SPat and the Com models. The disproportionate number of communication fault cases reflects the research priority for identifying communication errors.

Table. 3. Fault type x detected errors

Fault Type	Fault injection experiments	Detected Errors
Communication	283	31
Processor	80	5
Memory	88	3
Total	451	39

The other 319 test cases generated from Norm and SExc models resulted in only 12 errors. Thus, CoFI was able to identify a significant number of errors in a relatively small number of fault cases. This suggests the CoFI methodology identified likely errors successfully when pre-defining which faults to inject, confirming the advantages of deterministic fault injection methods [9].

6 Conclusions

Considering that we were going to validate software supplied by a competent team from a prominent Brazilian Software industry to INPE, which was developed under rigorous quality assurance rules, we expected to find very few errors. The results surprised us as 51 errors were yet found.

Since the models reflected the software behavior based on information obtained from the textual documents such as protocol specification, technical specification, software design, and manual, all non-conformances between code and document were computed as a detected error. These errors were classified as 45% only code errors 33% only document non-conformance errors and 22% code and document non-conformance errors.

The results pointed out that focusing on the faults is more effective than on the normal behavior for validation purposes. Since models were grouped by fault types, the set of automatically generated fault cases were distinguishable by their fault types and statistical calculations on the tests were facilitated.

The INPE test team worked independently of the industrial development team to create the models. Obviously, the greater effort to create the models was compensated by the superior test organization CoFI achieved in comparison with previous ad hoc test designs. The models served as guides to focus the tester's attention to the faults and exceptions that could occur during the software's operation, leading to the design of situations the developers had not thought of. One example is for OBDH to require data during SWPDC initialization service.

Future work is required to verify whether other types of errors are identified when test models are combined. In order to make the CoFI testing methodology applicable to any space application, the adoption of a standardized test language to represent the inputs and outputs seems to be important as well.

Acknowledgments

The authors acknowledge the financial support from Financiadora de Estudos e Projetos (FINEP) to the QSEE research project and all those involved. The authors also thank the reviewers for their insightful comments and constructive suggestions.

References

1. Ambrosio, A.M. "CoFI: uma abordagem combinando teste de conformidade e injeção de falhas para validação de software em aplicações espaciais," INPE-13264-TDI/1031. Instituto Nacional de Pesquisas Espaciais - INPE, (2005)
2. Ambrosio, A.M., Martins E., Vijaykumar N. L., Carvalho, S.V., "A Methodology for Designing Fault Injection Experiments as an Addition to Communication Systems Conformance Testing," Proceedings of the 1st Workshop on Dependable Software - Tools and Methods in the IEEE Conference on Dependable System and Network, 28 June – 1 July 2005, Yokohama, Japan (2005)

3. Arlat, J.; Aguera, M.; Amat, L.; Crouzet,Y.; Fabre, J.-C.; Laprie, J.-C.; Martins, E.; Powell,D. Fault Injection for Dependability Validation: A Methodology and Some Applications. IEEE Tr on SE, v. 16, n.2, p. 166-182 (1990)
4. Binder, R. Testing Object-Oriented Systems-Models, Patterns and Tools – Addison-Wesley (2000)
5. Cavalli, A.; Gervy, C.; Prokopenko, S. "New Approaches for Passive Testing using Extended Finite State Machine Specification". In: WTCS, Canada (2001)
6. Chandra, R.; Lefever, R.M.; Cukier, M; Sanders, W.H. A global-state triggered fault injector for distributed system evaluation. IEEE Transaction on Parallel and Distributed Systems –p.593-605 v.15 n.7 (2004)
7. Chow, T.S. "Testing software design modeled by finite state machines". IEEE Trans on Sw Engineering (TSE), vol.SE-4, nr. 3, pp178-187 (1978)
8. Dssouli, H.; Salek, K.; Aboulhamid, E; En-Nouaary, A ; Bourhfir, C. Test Development for Comm. Protocols: Towards Automation. Computer Networks, n. 31, p. 1835-1872 (1999)
9. Echtle, K; Chen, Y. Evaluation of Deterministic Fault Injection for Fault-Tolerant Protocol Testing. IEEE 21th Annual International Symposium on Fault-Tolerant Computing, Montreal, p.418-425 (1991)
10. Goddard Space Flight Center (GSFC). Available at: <http://radhome.gsfc.nasa.gov/radhome/papers/seeca1.htm>. Accessed in: March 2007
11. Holzmann, G. J. Design and validation of computer protocols. Prentice Hall, 1990
12. International Organization for Standardization ISO/IEC– IS9646 International standard conformance testing methodology and framework. Geneve (1991)
13. Martins, E.; Mattiello-Francisco, F. A Tool for Fault Injection and Conformance Testing of Distributed Systems. Lecture Notes in Computer Science, v. 2847/2003, pp. 282-302, Brazil (2003)
14. Martins, E.; Sabião, S.B.; Ambrosio, A. M. ConData: a Tool for Automating Specification-based Test Case Generation for Communication Systems. Software Quality Journal, v.8, n. 4, p. 303-319 (1999)
15. Mattiello-Francisco, M.F.; Santiago V.A., Costa, R., Jogaib, L.. Verificação e Validação na terceirização de software embarcado em aplicações espaciais. Simpósio Brasiliero de Qualidade de Software - SBQS2006, pp. 368-375, Villa Velha, ES, Brazil (2006)
16. Santiago, V.; Mattiello-Francisco, F.; Costa, R.; Silva, W.P.; Ambrosio, A.M. "QSEE Project: An Experience in Outsourcing Software Development for Space Applications". In The Nineteenth International Conference on Software Engineering and Knowledge Engineering (SEKE'07), Boston, EUA (2007)
17. Silva, W. P. et al, QSEE-TAS: Uma Ferramenta para Execução e Relato Automatizados de Testes de Software para Aplicações Espaciais, XX Brazilian Symposium on Software Engineering-SBES (2006)

PUBLICAÇÕES TÉCNICO-CIENTÍFICAS EDITADAS PELO INPE

Teses e Dissertações (TDI)

Teses e Dissertações apresentadas nos Cursos de Pós-Graduação do INPE.

Manuais Técnicos (MAN)

São publicações de caráter técnico que incluem normas, procedimentos, instruções e orientações.

Notas Técnico-Científicas (NTC)

Incluem resultados preliminares de pesquisa, descrição de equipamentos, descrição e ou documentação de programa de computador, descrição de sistemas e experimentos, apresentação de testes, dados, atlas, e documentação de projetos de engenharia.

Relatórios de Pesquisa (RPQ)

Reportam resultados ou progressos de pesquisas tanto de natureza técnica quanto científica, cujo nível seja compatível com o de uma publicação em periódico nacional ou internacional.

Propostas e Relatórios de Projetos (PRP)

São propostas de projetos técnico-científicos e relatórios de acompanhamento de projetos, atividades e convênios.

Publicações Didáticas (PUD)

Incluem apostilas, notas de aula e manuais didáticos.

Publicações Seriadas

São os seriados técnico-científicos: boletins, periódicos, anuários e anais de eventos (simpósios e congressos). Constam destas publicações o Internacional Standard Serial Number (ISSN), que é um código único e definitivo para identificação de títulos de seriados.

Programas de Computador (PDC)

São a seqüência de instruções ou códigos, expressos em uma linguagem de programação compilada ou interpretada, a ser executada por um computador para alcançar um determinado objetivo. São aceitos tanto programas fonte quanto executáveis.

Pré-publicações (PRE)

Todos os artigos publicados em periódicos, anais e como capítulos de livros.