Collecting, Analyzing and Archiving Results from Fault Injection Experiments

Jean Arlat[1,2]
[1] CNRS; LAAS; 7, avenue du Colonel Roche
[2] Université de Toulouse; UPS, INSA, INP, ISAE,
UT1, UTM, LAAS; F-31077 Toulouse Cedex 4, France
jean.arlat@laas.fr

Regina Moraes
School of Technology, UNICAMP
1888, Rua Paschoal Marmo, Jardim Nova Itália
13484-332 Limeira, SP, Brazil
regina@ft.unicamp.br

***Abstract* — This paper addresses the issue of the identification of the suitable level of observation (readouts and measurements) to characterize fault injection experiments. In practice, several outcomes can be observed in an experiment, but it is not rare for experimenters to consider only one viewpoint or to rely on the first event observed, in order to diagnose the experiment. In addition, there is not always a single way to assert the faulty behaviors as distinct viewpoints might be considered. Accordingly, an elaborate reflection on the types of readouts and measurements to be collected and recorded is an essential dimension for analyzing the faulty behavior of a target system. Another key aspect concerns the need for archiving the experimental data in a suitable way (featuring sufficient details, still in an exploitable format), so that they can be also useful for extended or alternative analyses. With that in mind, the paper sketches some simple guidelines towards the sharing of experimental results via an open data repository.**

Keywords — Dependability Assessment; Controlled Experiments; Fault Injection; Collecting Outcomes; Archiving Results.

I. INTRODUCTION

For several decades, fault injection has been recognized as a pragmatic and useful approach to study the dependability and characterize the behavior of complex fault-tolerant computer systems [1, 2] and components [3, 4], in the presence of faults. Most likely, due to the rapid development and evolution of computer technologies, the vast majority of the conducted experiments have been successively considering distinct target systems (e.g., new architectures, components, configurations, etc.). Studies have seldom been using the very same target system in order to allow comparison and consolidation of results.

Except for very few studies (e.g., see [5]), such a tendency to address an "ever moving target" is a strong characteristics of the research in fault injection experiments. In fact, this is rather in opposition with most experimental scientific research studies, where it is rather common to replicate experiments to confirm the results obtained. Repeatability is definitely a key feature in that context.

Such a state of affair can be easily understood when the conducted experiments are meant to assess the fault tolerance features of a very specific computer system. However, when experiments are rather carried out for the purpose of dependability benchmarking [6], it is then important that the experimental context and results be openly disclosed. This way, it is possible to conduct similar experiments either on the same target system (for confirmation) or on an alternative candidate system (for comparison). Such a trend has actually been concretized under the form of the Data Repository recently developed in the frame of the AMBER project [7]. Nevertheless, some practical issues have to be dealt with for the archived data to be fully exploitable.

When disclosing the benchmark conditions (which is required for objectively interpreting its results), the complete set of attributes is to be included. In the case of dependability benchmarking, these attributes refer to: a) the input domain (workload and faultload), b) the output domain (readouts and measurements). In this paper, we emphasize the output domain.

Indeed, a lot of works have been reported that have addressed the input domain. In particular, beyond early investigations aimed at proposing efficient fault injection techniques and supporting tools (e.g., see [8, 9]), several studies have been concerned with the identification of: a) faultloads representative of specific fault sets [10-12], b) a suitable workload — and most likely in combination with the faultload — aimed at maximizing the significance of the conducted experiments (e.g., for targeting the specific fault tolerance mechanisms to be tested [13-15], for avoiding experiments where the injected fault would not be activated as an error [16], etc.).

For what concerns the output domain, a lot of work has addressed the experimental evaluation of the coverage of fault tolerance mechanisms and of failure modes (e.g., see [2], [3]). Fewer studies have focused on a detailed examination of the issues attached to the observation and analysis of experimental outcomes. Among these, one has to consider the work reported in [17-19] where the authors have investigated various ways to analyze the outcomes (error detection, delivery of an erroneous results) occurring during a single experiment. Another set of works has considered various end-user viewpoints to perform the analysis of the behaviors observed in presence of faults (e.g., see [20, 21]). Another recent trend has been promoting the application of metrology concepts to dependability evaluation based on fault injection experiments [22].

We advocate that several other aspects are also relevant in that context. They concern the various capabilities and opportunities to fully exploit and possibly reuse a set of experimental results, provided that they are made available in a form that is suitable for an end-user to perform a posteriori alternative analyses based upon additional insights (components, environment, requirements, etc.) he/she is aware of.

This means that in order to be considered as fully successful and useful, the end-product of a dependability benchmarking trial should go beyond the provision of an accurate set of figures. As previously pointed out, it should also disclose the major underlying results upon which the benchmark measures have been obtained. For a rationale, let us simply refer to the form of benchmark that customers are used to exploit when assessing the respective merits of a car. Data for various (quantitative or qualitative) criteria are provided to the user so that he/she can perform a well-informed assessment that is taking into account the actual importance of each of these criteria on his/her driving inclinations and expectations.

In this paper, we illustrate these ideas via some examples derived from actual fault injection experiments. Accordingly, it is specifically targeting people who are involved in setting up and conducting experimental work based on fault injection. We first describe how different viewpoints can be accommodated when referring to sufficiently detailed experimental results (Section II). In Section III, we then propose some guidelines indicating the most suitable way to conduct experiments and capture experimental data for inclusion into a data repository, so that researchers are able to make the most of the recorded data. Finally, Section IV concludes the paper.

II. Experimental Data and User Viewpoints

Conducting a fault injection campaign is a tedious, time consuming and often difficult exercise. Accordingly, it is rather natural to try to get the most of it in terms of insights about the failure modes and the robustness features of the target system. As already pointed out, here, we focus on the exploitation of available readouts and measurements.

In the subsequent paragraphs, we provide examples of possible extensions and inferences that can be achieved when considering the data obtained in a fault injection campaign.

A. Weighting of experimental results

A fault injection experiment simply provides insights about the behavior of a target system in presence of injected faults. This means that the measures that can be directly derived are only conditional dependability measures. For example, the fault occurrence process — the rate and distribution of faults into the target system — is usually not accounted for in the experimental results obtained.

Let us consider the results obtained during a fault injection campaign carried out on a target system composed of three modules as described in Table I. We assume that, the series of fault injection experiments were organized in three runs (one for each module) featuring each the same number of experiments.

The Table indicates, for each module, the number of faults injected and their impact on the target system-wide error detection mechanism (EDM). The last line gives the resulting estimate for the coverage factor in each case. At this stage, based on the experiments carried out, a potential estimate for the error detection coverage for the whole target system is the figure given at the bottom right of the Table.

TABLE I. Experimental results per module

Module	**1**	**2**	**3**	**Total**
#Faults	150	150	150	450
#Detections	100	110	140	350
Coverage (%)	67	73	93	78

An alternative clustering of the same data can be performed according to a different dimension, so as to help obtaining additional insights. For example, Table II displays the data from the same experiment, this time sorted with respect to (a sample of) the types of software faults that were injected. In this case only 51% of WEC were detected which noticeably indicates that detection mechanisms should be improved in order to deal with this type of fault (undetected error is disdained). The same overall detection coverage is obtained.

TABLE II. Experimental results per type of faults

Fault Type	**MFC**	**MIFS**	**WEC**	**Total**
#Faults	252	145	53	450
#Detections	198	125	27	350
Coverage (%)	79	86	51	78

MFC – Missing Function Call; MIFS – Missing IF Construct + Statement; WEC – Wrong Extended Class

When looking at the campaign from this perspective, one may object that this figure is influenced by the fact that different numbers of experiments were conducted for each fault type.

This is to be expected, as the number of each fault type clearly depends on the specific structure of the source code and this is just right, provided that the respective proportion of faults is representative of the related statements.

Further on that, let us come back to Table I and assume that after having conducted the experiments, some additional insights are made available. To illustrate this, consider that the respective failure rates[1] for each module in Table I are know. Let us denote λ_i each of these rates. Table III provides this additional information.

Thus, a more accurate (failure-rate aware) estimate of the detection coverage (c_D) can be obtained via a weighted average of the module coverage (c_{DMi} for i = 1…3) and the

[1] Actually, the failure of a module corresponds to a fault occurring into the system composed of these three modules.

relative probability for a module to be faulty ($pf_{Mi} = \lambda_i / \sum_i \lambda_i$) shown in the last row of Table III as follows [23]:

$$c_D = \sum_i c_{DMi} \times pf_{Mi} = 69\%$$

TABLE III. FAILURE RATES FOR THE TARGET SYSTEM MODULES

Module	**1**	**2**	**3**	**Total**
Failure rate λ_i (per hour)	$80\ 10^{-6}$	$15\ 10^{-6}$	$5\ 10^{-6}$	10^{-4}
Pr[fault in mod] pf_{Mi} (%)	80	15	5	100

Two remarks can be made:

1. Clearly, this coverage figure is quite differing from the global estimate in Table I, which implies a significant impact on dependability (e.g., see [24]).
2. Such an adjustment was possible thanks to the disclosure of module-based experimental results in addition to the global estimate.

Such a post-processing of the experimental results is quite common in practice[2]. It might also refer to insights concerning the workload (e.g., see [25]) carried out on each modules (a more active module is much prone to exercise faults as errors) or even the risk induced by the non detections on the application controlled by the target system (e.g., see [26]), etc.

B. Ordering and severity of the outcomes

Classically, the outcome of an experiment is classified as detected (D) if the first event observed is the activation of an EDM. Conversely, when a failure occurs (e.g., the delivery of wrong results to the controlled application) before any error detection, the system is considered to have failed (F). For some applications, where the controlled process features slow dynamics, it might be sufficient that the EDM signal the error within a given temporal window so that the system is not to be considered as failed — e.g., see the flight control system of an aircraft studied in [27]. In practice, when considering such a temporal window as equivalent to the duration of a fault injection experiment, the ordering of the observed events might be less important.

Table IV provides examples of some typical syndromes when multiple outcomes can be distinguished as the consequence of a fault injection experiment. A "1" indicates the observation of the event in the experiment. Usually, not all combinations of the events are possible: in particular, Workload Abort dominates Workload Incorrect, i.e., no WI outcome can be observed when a WA outcome is diagnosed first.

[2] This would also apply to the data clustering illustrated in Table II.

TABLE IV. SYNDROMES FOR A FAULT INJECTION EXPERIMENT

	Notification		**WL Failure**		**First**	**Priority to**		
	EC	**XC**	**WA**	**WI**	**event?**	**1st event**	**Notif.**	**Failure**
1	0	0	0	0	N/A	N/A	N/A	N/A
2	1	0	0	0	EC	D	D	D
3	0	0	0	1	WI	F	F	F
4	1	0	0	1	EC	D	D	F
5	0	1	0	1	WI	F	D	F
6	0	0	1	1	WI	F	F	F
7	1	1	0	0	EC	D	D	D
8	1	0	1	0	EC	D	D	F

EC: Error Code; XC: Exception; WA: Workload Aborted; WI: Workload Incorrect

The first row depicts the case when none of the considered events has been observed. This is a classical issue in testing and experimental studies. It may result from several alternatives (fault not activated, error masked, etc.). Rows 2 an 3 identify cases when a single event is observed, in which case the characterization of the outcome is straightforward and steady both with respect to the identification of the first event and the diagnosis (D or F), irrespective of the priority assumed for the analysis. The subsequent rows depict more interesting cases:

- **Row 4 (resp. 5)**: An error notification — error code (resp. exception raised) — is observed prior to (resp. after) the delivery of a wrong result, which leads to distinct diagnosis (whether priority is given to notification of failure) with respect to the first event-based analysis.
- **Row 6 (resp. 7)**: More than one event is observed in each main categories: notification (resp. failure). Further detailed analyses can be conducted by considering the usefulness of the error detection with respect to a potential recovery action: an error code is likely to be more exploitable than an exception (resp. the impact of the failure mode on the workload (a WA or a WI) might have distinct impact depending on the application process being considered.
- **Row 8**: Based on the comments for rows 6 and 7, this syndrome would correspond to the less severe case when both a notification and failure are observed.

Furthermore, it is interesting to stress that row 2 could be interpreted as a "false positive" with respect to the viewpoint of error detection (the error did had any impact on the WL) and row 3 a "false negative" with respect to fault tolerance.

As a final comment, for such detailed and alternative analyses to be performed, comprehensive and precise information about the various events that occurred during each experiment is clearly needed.

III. MANAGING EXPERIMENT OUTCOMES

This section discusses some suitable ways to capture experimental data and record them for exploitation in a repository, so that the research community is able to exploit

the recorded data. Many published works exist that provide useful hints concerning the management of large sets of data in many experimental contexts (e.g., see [28]). Here we focus on fault injection experiments. Still, our proposal is far from providing a comprehensive reference; rather, it should be considered as a "bootstrap" or an initial incentive that can be extended including by other researchers.

One important aspect that singularizes fault injection experiments is related to the fact that these are "controlled experiments". This means that by definition, the experimenter is explicitly acting upon the input domain (at least via the injected faults), which has obviously an impact on the output domain (the observed outcomes). Accordingly, a strong interplay exists between the observability and controllability dimensions.

Among potential people interested, we believe that students involved into experimental work would be the prime target for such guidelines. Indeed, it is not rare, during student research development, for the time spent with the experiments to be jeopardized or at least impaired by a poor planning, conduct and too restrictive recording of the experiment outcomes. Such guidelines should also provide a helpful instrument for advisors to convey relevant practices, as it is not always easy to verbalize all details involved, and to increase the chances that the students catch up with the related knowledge and skills.

Moreover, such guidelines should allow for the experiment results collected to be more easily and efficiently exploited by interested researchers when made available in data repositories: someone's else data can only be useful if the data provider is careful to collect, document and store these data in a comprehensive way.

The selected recommendations that follow are based on our own experiences, but also on exchanges with PhD students and other researchers in the field.

A. Planning the Experiments

Most of the concerns that have been verbalized by the students and researchers consulted are related to this phase of the experiments.

The primary objective is to translate the hypothesis to be demonstrated / proved or the type of measures to be derived, into a sound fault injection campaign. This has definitely an impact on the whole sets of attributes (workload and faultload, but also, readouts and measurements), as well as on the way the experiments are conducted (testbed configuration) and the results analyzed (e.g., statistical analysis).

Accordingly, whenever available, existing suitable tools have to be used, or else, novel experimental techniques are to be devised and related enabling technologies developed.

As previously explained, we focus on the impact of the planning of a fault injection campaign on the outcomes of the experiments. Two ways for planning a campaign (i.e., the injection of a series of faults) can be distinguished:

- The campaign is made up of a series of independent experiments, in which a fault pattern (possibly involving multiple faults) is injected and the target system is observed during a specific timeframe (e.g., the execution of a specific application task) for observing and collecting relevant outcomes induced by the injected fault. In practice, it is necessary to check the target system for possible residual errors. Usually, a special integrity test is run for that purpose and to purge the target system from latent errors.
- Here also, a series of faults are injected during the execution of the workload. However, in that case, the end of each experiment is characterized by the occurrence of a specific failure event (e.g., a crash). The same reset of the target system is to be performed before launching subsequent campaigns.

The main differences are as follows: while in the first case, the individual effect of a single fault can be assessed and thus recorded, in the second case only the combined effect of the injected faults can be reported. However, in the latter, the overall availability of the target system and the impact of the injected faults (e.g., number of faults injected before a crash occurs) can be more thoroughly estimated.

B. Collecting the Outcomes

There are different concerns during the conduct of experiments related to the observation and collection of experiment outcomes. Among them, the main preoccupation is to avoid undesired interferences with the target system. Such a worry is very much exacerbated when detailed timing measurement is to be performed as part of the experiment.

Besides the usual timing parameters that are essential to characterize a fault injection experiment (start time, fault injection time, end time, etc.) the record of timing information for relevant events (detection, recovery, crash, etc.) is often essential to fully characterize the faulty behavior of a target system.

Moreover, compared to the case of performance benchmarking, dependability assessment requires much more attention to be paid to the target system and its operational profile.

First, as it is routinely done for failure data analysis studies (e.g., see [29]), in the case of fault injection experiments, it is also important to record data characterizing the target system itself, the production process and the use environment. For example, for software systems, important information include: language and program size, version being tested, development tools and compilers, workload type, etc. [6].

Second, the observation of timing information about specific activity concerning the workload being executed (e.g., time of occurrence of a request) is very often mandatory, to be able to diagnose the status of the target system. Also, the characterization of fault tolerance

mechanisms frequently requires that related timing information (error detection time, recovery duration, etc.) be measured. Then, it is essential that either the same time scale be used or that the various time scales be synchronized with sufficient precision.

However, the more detailed the timing information, the more sophisticated and intricate the related observation devices. Indeed, both the intrusiveness (e.g., the temporal interference or overhead induced by the measurement mechanisms) and the accuracy (e.g., the instrumentation requirements and related cost) are to be carefully balanced. In the extreme case when no overhead at all is tolerable, a radical approach exposed in [19] was to "freeze" the progression of time when the events are observed[3]. This is more and more acceptable when considering the trend towards the development of generic virtual execution platforms featuring extensive simulation capabilities encompassing hardware (Hardware-in-the-Loop) and real time kernel functionalities (e.g., see [30]).

C. *Archiving the Experimental Data*

In accordance with [28], which states that "The data have then to be organized into an appropriate form for analysis (often in different ways, depending on the analysis)", we advocate that specific care needs to be taken in managing and storing the outcomes of fault injection experiments. In practice, a good balance is to be made between two extreme cases: a) storing the complete set of raw data and b) simply archiving the dependability measures of interest. Indeed, when considering the first option, one has to consider the fact that in many cases the raw data is very large, and more importantly, very difficult to exploit without some supporting parsing tools. For example, when considering experimental studies targeting operating systems, the basic raw data may correspond to execution traces of considered operating systems when subjected to faults. Clearly, such a raw material is virtually useless as is, for most end-users, unless dedicated scripts are provided to help parse it.

Accordingly, the raw data collected usually requires the application of some specific filtering technique, so as to keep the data relevant for the viewpoint of the experiment being conducted and focus on the suitable level of details (e.g., see, [29]). Still, the filtering process should be carefully tuned for the stored data to be useful for research studies related to the main topic.

A popular format for recording and analyzing data sets corresponds to the use of spreadsheet tables. In practice, several tables are produced that may correspond to different part of the campaign (e.g., records concerning the input set workload, faultload on one side and readouts and outcomes on the other side). Such a potential disconnection might cause problems when trying to relate such data sets, for example, in order to perform a detailed cause and effect analysis when singular or unexpected behaviors are revealed by the observations made. Such a shortcoming might not only impair the pertinence of the overall assessment, but also the confidence in the experimental data produced. More generally, a strict control of the data versions should be enforced in order to precisely identify which data is related to the various runs and experiments.

All this calls for the support of a database, and especially a relational database, so that the complete data set can be archived in a coherent format and procedures to exploit it can be explicitly attached to it.

[3] This requires the hardware clock to be managed by the fault injection tool so as to disable and resume its counting.

IV. Concluding Remarks

The paper provides some insights about collecting, analyzing, and archiving results obtained in fault injection experiments. As we have illustrated by means of examples in Section II, for experimental outcomes to be readily reusable, it is important to have sufficiently detailed data disclosed.

In some scientific fields, for example biology, depositing such a data in a formal repository can be a prerequisite for publication. Experimental research in computer science does not have such a constraint, and while this might be a too strong requirement, clearly, there is room for improvement! Accordingly, as an attempt to bootstrap such a trend, we have discussed a few hints for collecting and archiving experimental data.

Moreover, to ensure that data survive the interests of original researchers, it would be highly desirable that such repositories be managed by perennial organizations (possibly professional societies). Also, to get the most of the data stored in the repository, it is essential that these data be accompanied by documents and procedures that make raw data actually parsable and exploitable by most end-users.

Sharing data via relational databases would allow for generic storage and advanced processing. This would mean that with such data, you can do all sorts of things, you can analyze it, you can combine it with other data sources, you can reach new conclusions and strike down old ones.

Such a trend should make it possible to researchers to objectively say: "I have evidence that it is (or is not) working well", instead of simply stating: "I have the feeling that it is (or is not) working well". This should be a significant improvement for all experimental research in the dependability assessment.

Acknowledgment

The study reported herein was initiated during Jean Arlat's one-month stay at the School of Technology, UNICAMP, supported in part by UNICAMP Visiting Professor Program and FAEPEX. Regina Moraes work is partially supported by the REVVIS project and linked to *Engenharia de Informação e Sistemas* research group. This work was also supported in part by the CAPES-COFECUB project RobustWeb.

The authors gratefully acknowledge useful discussions with colleagues, especially Yves Crouzet at LAAS-CNRS (France), and PhD students: Tânia Basso at UNICAMP (Brazil), Afonso Araujo Neto, Naaliel Vicente Mendes, Nuno Manuel dos Santos Antunes at the University of Coimbra (Portugal).

REFERENCES

[1] D. A. Rennels, "Some Past Experiments and Future Plans in Experimental Evaluations of Fault Tolerance," *Proc. Int. Symposium. on Mini and MicroComputers in Control and Measurement*, San Francisco, CA, USA, 1981, pp. 91-98.

[2] J. Arlat, M. Aguera, L. Amat, Y. Crouzet, J.-C. Fabre, J.-C. Laprie, E. Martins, D. Powell, "Fault Injection for Dependability Validation — A Methodology and Some Applications," *IEEE Trans. on Software Engineering*, vol. 16, no. 2, pp. 166-182, 1990.

[3] N. P. Kropp, P. J. Koopman, D. P. Siewiorek, "Automated Robustness Testing of Off-The-Shelf Software Components," *Proc. 28th IEEE Annual Symp. on Fault-Tolerant Computing (FTCS-28)*, Munich, Germany, 1998, pp. 230-239.

[4] K. Kanoun, Y. Crouzet, "Dependability Benchmarks for Operating Systems," *Int. Journal of Performability Engineering*, vol. 2, no. 3, pp. 275-287, 2006.

[5] J. Arlat, Y. Crouzet, J. Karlsson, P. Folkesson, E. Fuchs, G. H. Leber, "Comparison of Physical and Software-Implemented Fault Injection Techniques," *IEEE Trans. on Computers*, vol. 52, no. 9, pp. 1115-1133, 2003.

[6] K. Kanoun, L. Spainhower, *Dependability Benchmarking for Computer Systems*, *IEEE Computer Society Press and Wiley*, 2008, 362 p.

[7] M. Vieira, N. Mendes, J. Durães, "A Case Study on Using the AMBER Data Repository for Experimental Data Analysis," *Workshop on Sharing Field Data and Experiment Measurements on Resilience of Distributed Computing Systems (co-located with IEEE Symp. on Reliable Distributed Systems (SRDS-2008)*, Naples, Italy, 2008. See also: www.amber-project.eu.

[8] J. Carreira, H. Madeira, J. G. Silva, "Xception: A Technique for the Experimental Evaluation of Dependability in Modern Computers," *IEEE Trans on Software Engineering*, vol. 24, no. 2, pp. 125-136, 1998.

[9] E. Martins, M. de F. Mattiello Francisco, "A Tool for Fault Injection and Conformance Testing of Distributed Systems," *Proc. 1st Latin American Symposium on Dependable Computing (LADC-2003)*, São Paulo, Brazil, 2003, pp. 282-302.

[10] V. Sieh, O. Tschäche, F. Balbach, "Comparing Different Models Using VERIFY," *Proc. 6th International Working Conference on Dependable Computing for Critical Applications (DCCA-6)*, Grainau, Germany, 1997, pp. 59-76.

[11] J. Durães, H. Madeira, "Emulation of Software Faults: A Field Data Study and a Practical Approach," *IEEE Trans. on Software Engineering*, vol. 32, no. 11, pp. 849-867, 2006.

[12] R. Moraes, R. Barbosa, J. Durães, N. Mendes, E. Martins, H. Madeira, "Injection of Faults at Component Interfaces and Inside the Component Code: Are They Equivalent?," *Proc. European Dependable Computing Conference (EDCC-6)*, Coimbra, Portugal, 2006, pp. 53-64.

[13] J. Arlat, J. Boué, Y. Crouzet, "Validation-based Development of Dependable Systems," *IEEE Micro*, vol. 19, no. 4, pp. 66-79, 1999.

[14] F. Saad Khorchef, A. Rollet, R. Castanet, "A Framework and a Tool for Robustness Testing of Communicating Software," *Proc. ACM Symp. on Applied Computing*, Seoul, Korea, 2007, pp. 1461-1466.

[15] P. Costa, J. G. Silva, H. Madeira, "Dependability Benchmarking Using Software Faults - How to Create Practical and Representative Faulloads," *Proc. 15th IEEE Pacific Rim International Symp. on Dependable Computing (PRDC-15), Shanghai, China*, 2009, pp. 289-294.

[16] R. Barbosa, J. Vinter, P. Folkesson, J. Karlsson, "Experimental Dependability Evaluation of a Fail-Bounded Jet Engine Control System for Unmanned Aerial Vehicles," *Proc. European Dependable Computing Conf. (EDCC-5)*, Budapest, Hungary, 2005, pp. 246-262.

[17] P. Chevochot, I. Puaut, "Experimental Evaluation of the Fail-Silent Behavior of a Distributed Real-Time Run-Time Support Built from COTS Components," *Proc. Annual IEEE/IFIP International Conf. on Dependable Systems and Networks (DSN-2001), Göteborg, Sweden, 2001, pp. 304-313.*

[18] A. Steininger, C. Scherrer, "Identifying Efficient Combinations of Error Detecting Mechanisms Based on Results of Fault Injection Experiments," *IEEE Trans. on Computers*, vol. 51, no. 2, pp. 235-239, June-July 2002.

[19] M. Rodríguez, A. Albinet, J. Arlat, "MAFALDA-RT: A Tool for Dependability Assessment of Real Time Systems," *Proc. Annual IEEE/IFIP International Conf. on Dependable Systems and Networks (DSN-2002)*, Washington, DC, USA, 2002, pp. 267-272.

[20] J. Durães, H. Madeira, "Mutidimensional Characterization of the Impact of Faulty Drivers on the Operating Systems Behavior," *IEICE Trans. on Information and Systems*, vol. E86-D, no. 12, pp. 2563-2570, 2003.

[21] A. Albinet, J. Arlat, J.-C. Fabre, "Benchmarking the Impact of Faulty Drivers: Application to the Linux Kernel," see [6], pp. 285-310, 2008.

[22] A. Bondavalli, A. Ceccarelli, L. Falai, M. Vadursi, "Foundations of Measurement Theory Applied to the Evaluation of Dependability Attributes," *Proc. 37th Annual IEEE/IFIP International Conf. on Dependable Systems and Networks (DSN-2007)*, Edinburgh, UK, 2007, pp. 522-533.

[23] D. Powell, E. Martins, J. Arlat, Y. Crouzet, "Estimators for Fault Tolerance Coverage Evaluation," *IEEE Trans. on Computers*, vol. 44, no. 2, pp. 261-274, 1995.

[24] J. Arlat, A. Costes, Y. Crouzet, J.-C. Laprie, D. Powell, "Fault Injection and Dependability Evaluation of Fault-Tolerant Systems," *IEEE Trans. on Computers*, vol. 42, no. 8, pp. 913-923, 1993.

[25] X. Ju, H. Zou, "Operating System Robustness Forecast and Selection," *Proc. 19th IEEE International Symp. on Software Reliability Engineering (ISSRE-19)*, Seatlle, WA, USA, 2008, pp. 107-116.

[26] R. Moraes, J. Durães, R. Barbosa, E. Martins, H. Madeira, "Experimental Risk Assessment and Comparison Using Software Fault Injection," *Proc. 37th Annual IEEE/IFIP International Conf. on Dependable Systems and Networks (DSN-2007)*, Edinburgh, UK, 2007, pp. 512-521.

[27] A. Youssef, Y. Crouzet, A. de Bonneval, J. Arlat, J.-J. Aubert, P. Brot, "Communication Integrity in Networks for Critical Control Systems," *Proc. European Dependable Computing Conference (EDCC-6)*, Coimbra, Portugal, 2006, pp. 23-32.

[28] Statistical Services Centre, *Data Management Guidelines for Experimental Projects*, Department of Applied Statistics, The University of Reading, 1998, 20 p. (www.reading.ac.uk/ssc).

[29] K.Kanoun, M. Kaâniche, J-C. Laprie, "Qualitative and Quantitative Reliability Assessment," *IEEE Softwar*e, vol. 14, no. 2, pp. 77-87, 1997.

[30] M. Short, M. J. Pont, "Assessment of High-Integrity Embedded Automotive Control Systems using Hardware in the Loop Simulation," *Journal of Systems and Software*, vol. 81, no. 7, pp. 1163-1183, 2008.