POLITECNICO DI TORINO
Repository ISTITUZIONALE

The Art of Fault Injection

Original
The Art of Fault Injection / Benso, Alfredo; DI CARLO, Stefano. - In: CONTROL ENGINEERING AND APPLIED INFORMATICS. - ISSN 1454-8658. - STAMPA. - 13:4(2011), pp. 9-18.

Availability:
This version is available at: 11583/2424120 since:

Publisher:
SRAIT

Published
DOI:

Terms of use:

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright

(Article begins on next page)

02 May 2024

The Art of Fault Injection

Alfredo Benso * , **Stefano Di Carlo** *

* *Politecnico di Torino, Control and Computer Engineering Department, Torino, Italy (e-mail: alfredo.benso@polito.it,stefano.dicarlo@polito.it).*

Abstract: In this paper we relate the foremost virtues of temperance, justice, courage, and prudence to the correct methodological approaches that researchers should follow when setting up a fault injection experiment. With this work we try to understand where good and bad practices lie, in order to highlight those common methodological errors that deeply influence the coherency and the meaningfulness of fault injection experiments. Fault injection is like an art, where the success of the experiments depends on a very delicate balance between modeling, creativity, statistics, and patience.

Keywords: Fault Injection, Computer Fault Tolerance, Computer Fault Diagnosis.

1. INTRODUCTION

"Therefore, as with any other four things, if we were looking for any one of them and recognized it first, that would be enough for us, but if we recognized the other three first, this itself would be sufficient to enable us to recognize what we are looking for. Clearly it couldn't be anything but what's left over... Therefore, since there are four virtues, mustn't we look for them in the same way? Clearly".

Plato, the Republic

The use of digital systems pervades all areas of our lives from common house appliances such as microwave ovens and washing machines, to complex applications like automotive, transportation, and medical control systems. These digital systems provide higher productivity and greater flexibility, but it is also accepted that it is not easy to guarantee that they are fault free. Some faults may be attributed to flaws during the development, while others can stem from external causes such as process defects or environmental stress. Moreover, as devices geometry decreases and clock frequency increases, the incidence of transient errors due to external stress (e.g., radiation, EMI, etc.) increases, and consequently, the dependability of the systems decreases. High availability is therefore a requirement for every digital system whose correct functionality is connected to human safety or economic investments. In this context, the evaluation of the dependability of a system plays a critical role. Unlike performance, dependability cannot be evaluated using benchmark programs and standard test methodologies. It requires observing the system behavior after the appearance of a failure. However, since the Mean-Time-Between-Failures (MTBF) in a dependable system can be of the order of years, the fault occurrence must be artificially accelerated in order to analyze the system reaction to a fault without waiting for its natural appearance.

Fault injection (Arlat et al., 1993; Benso et al., 1998b; Carreira et al., 1998; Clark and Pradhan, 1995; Choi and Iyer, 1992; Guthoff and Sieh, 1995; Mei et al., 1997; Jenn et al., 1994; Karlsson et al., 1994; Benso et al., 1998a; Benso and Prinetto, 2003; et al.., 2006) emerged as a suitable solution, and it has been deeply investigated by both academia and industry. Fault injection techniques are continuously evolving and improving. In this paper we will mostly refer to the ones that helped building the foundations of this topic.

In general, fault injection techniques can be grouped in hardware-implemented (Karlsson et al., 1994), simulation-based (Jenn et al., 1994), software-implemented (Benso et al., 1998b; Carreira et al., 1998; Choi and Iyer, 1992), and hybrid (Guthoff and Sieh, 1995; Benso et al., 1998a; Velazco et al., 2000) fault injection. The process of setting up a fault injection environment requires different choices that can deeply influence the coherency and the meaningfulness of the final results. In this paper we analyze these choices to point out where, in our experience, they can be more easily affected by methodological errors and therefore lead to non-reliable dependability evaluations of the target system. In the paper we will use the analogy with the four Cardinal Virtues to discuss the four most important aspects of a fault injection environment: the choice of the fault model and fault list (temperance), the choice of the workload and inputs to apply to the target system (justice), the outputs to be chosen as readouts points (prudence), and the way of understanding and interpreting the experimental results (courage). This paper does not intend to propose a new fault injection methodology. The intention is to review existing techniques, discuss the most critical methodological experimental issues and to propose some guidelines to help designing robust fault injection environments.

The use of the four cardinal virtues is solely intended as a metaphor to create a different perspective to the realities of fault injection. We do not wish, in any way, to offend any of the readers' religious sensibilities. This paper was written imagining setting up a fault injection environment in an ideal situation with an unlimited amount of resources,

time, and money. We understand that ideal situations are uncommon and that outside factors such as Time-To-Market or economic resources may influence the decision making process.

Finally, we would like to be the first to admit that we, the authors, in our research have incurred in most of the cited methodological errors. Our intention is to discuss the problems and some possible solutions, not to blame those who commit them, who, in this paper, will be referenced to with [omitted].

The paper is organized as follows: Section 2 presents the typical architecture of a fault injection environment. In Sections 3 we discuss the four cardinal virtues of fault injection. Finally, Section 4 concludes the paper.

2. TYPICAL STRUCTURE OF A FAULT INJECTION ENVIRONMENT

An effective way to characterize a fault injection environment is the FARM model, proposed by Arlat in (Arlat et al., 1993). The four FARM attributes are:

- the set of Faults $"F"$ to be deliberately injected into the system. F is also called the experiment *fault list*. Each fault is characterized by a *model* (e.g., stuck-at, bit-flip, short, etc.), a *location* (e.g., a flip-flop, a memory address, a pin, etc.), and an *injection time* (e.g., at a given clock cycle, after the execution of a certain instruction, after a given time, etc.). The size of the fault space is therefore $M \times L \times T$, where M is the set of possible fault models, L is the set of possible fault locations, and T the set of possible fault injection instants corresponding to the duration of each experiment. Contrary to traditional hardware testing where the size of the fault space is $M \times L$, in a fault injection experiment the size of the fault space is often assumed to be infinite, thus making impossible to work with exhaustive fault lists. The main problem in defining the target fault list F is therefore to select a subset of the entire fault space that can be injected in a reasonable time but still able to provide statistically significant results;
- the set of Activation trajectories $"A"$ that specifies how the system is functionally exercised during the experiment. It constitutes the set of functional inputs applied to the system during each experiment. The choice of A directly influences the length of every single experiment, and consequently, the size of the fault space ($M \times L \times T$) and target fault list. Often this model is extended to include the set of Workloads $"W"$ (e.g., a set of software benchmarks in the case of a microprocessor-based system);
- the set of Readouts $"R"$ that corresponds to the logged behavior of the system. Data recorded in R can strongly depend on the target system and on the mechanisms used to observe the system's behavior. For example, in a microprocessor-based system, recorded data may include memory accesses, the final application results, or the system exceptions. At the same time it may also include the waveform at the output of the microprocessor's pins. The quality and detail of the logged data can deeply affect the significance of the final fault injection results and also the experiment duration. The choice of R must therefore be a prudent trade-off between precision and time/memory overheads;
- the set of Measures $"M"$ obtained analyzing and elaborating the Readouts obtained during the experiment required to compute the final dependability estimation of the system.

Given the FARM model, a fault injection campaign is a collection of experiments, each requiring the injection of a fault f from the set F while the system is exercised with an activation trajectory a selected from A in a workload w from W. The set of measures M is obtained elaborating the set of readouts R gathered during each experiment. Fig. 1 presents the typical structure of a fault injection environment.

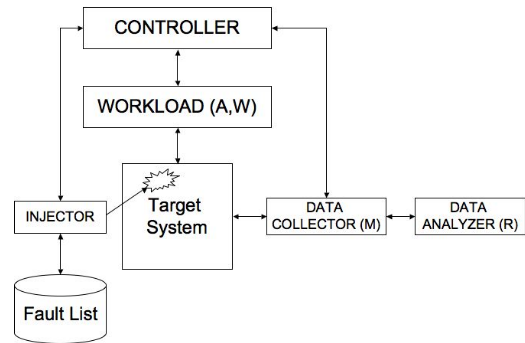

Fig. 1. A typical fault injection environment

The actual implementation of each functional block depends on the type of fault injection mechanism. There are basically four different approaches to fault injection:

- *hardware-implemented fault injection*: faults are injected in the actual system by changing its internal state using logic analyzers, heavy-ion radiations, FPGAs, or laser beams. Depending on the chosen injection method, only a limited set of locations can be corrupted. Hardware-implemented fault injectors are usually very expensive to setup, but they guarantee very realistic experiments;
- *simulation-based fault injection*: faults are injected into a model of the system (e.g., a VHDL or Verilog description) by modifying the model or using simulation scripts. This feature allows to theoretically injecting faults in any part of the system. The behavior of the system is then analyzed using a simulator. Simulation-based fault injectors are very low-cost solutions but the meaningfulness of their results strongly depend on the accuracy and abstraction level of the model of the target system;
- *software-implemented fault injection*: faults are injected into the actual target system using software procedures. Faults can therefore be inserted in any software accessible location. Software-implemented fault injectors are usually very flexible but can be used only in microprocessor-based systems;
- *hybrid fault injection*: hardware, simulation, and software approaches are all applied in the same environment to optimize performance and accuracy in the experiments.

In the following sections we will more deeply analyze the FARM model, highlighting the most common methodological errors that can badly affect the setup of a fault injection environment.

3. VIRTUES AND FAULT INJECTION

"I' mi volsi a man destra, e puosi mente
A l'altro polo, e vidi quattro stelle
non viste mai fuor ch'a la prima gente"
"To the right hand I turned, and fixed my mind
Upon the other people, and saw four stars
Ne'er seen before save by the primal people"

[Dante Alighieri, Purgatory, Chant I, 1-3]

The four stars that Virgilio sees in the Purgatory are the four cardinal virtues. They are called cardinal (Latin: cardo, hinge) virtues because they are hinges on which all moral virtues depend. We chose the analogy with the four cardinal virtues because they are supposed to be "natural" and they can be achieved through human effort.

Temperance is the virtue suggested in choosing an effective fault model F (see Section 2). The need of trading-off among several constraints requires finding the best "middle ground" or compromise between budget, time constraints and the representativeness of the fault list.

Justice is necessary when choosing the set of activations A (see Section 2). They have to strictly reflect the real working conditions, and cannot be chosen superficially "just to make the experiment work". They have consider both the goal of the experiment (validating a fault tolerant mechanism is different from evaluating the system response to faults), and the fault list, since it is most of the times recommended to inject faults only in the "active" parts of the system.

Prudence is essential in the choice of the readout points R (see Section 2). Logging the system behavior can be a very time and memory consuming task that directly affects the duration and the accuracy of the experiments.

Courage is required when elaborating the final results and deriving dependability measures M (see Section 2). It is very easy to jump to easy conclusions not supported by a statistical background. Real and useful results require analyzing the raw data with courage, without being afraid of having to add experiments or modify the methodology to obtain significant results.

3.1 Temperance (Faults)

"Temperance, called mastery of self, is really the mastery of the better over the baser qualities .... Temperance would seem to lie in the harmonious inter-relation of the different classes".

Plato, the Republic

Fault injection techniques mainly target one or more of the following goals:

(1) understanding the system's behavior under the effect of real faults;
(2) evaluating the fault tolerance mechanisms embedded in the target system;
(3) forecasting the faulty behavior of the target system and, in particular, encompassing a measurement of the efficiency (coverage) provided by the embedded fault tolerance mechanisms.

In all three contexts, one of the most critical phases is the definition of the fault list that most accurately represents the type of faults that can actually appear in the system under analysis. This is a delicate task since the type of possible real faults depends on different factors, such as the target system technology or its environmental working conditions. Differently from traditional testing techniques, in fault injection the fault space is made up of three dimensions: M, L, and T (see Section 2). They respectively are the fault model, the fault locations and the injection times. We can therefore divide the fault list generation process in three different steps:

- the choice of the fault model;
- the choice of the fault locations;
- the generation of the actual target fault list;

Temperance means "finding the middle ground", "avoiding excesses", and "offsetting an extreme". These are exactly the attitudes we need when defining a fault list where many parameters like budget and technical/time constraints, have to be traded-off with the reliability and confidence in the fault list. The choice of a bad fault list will completely invalidate the results of the experiments; either because the number of experiments is not statistically meaningful, or because the injected faults are not representative of the actual potential faults that may appear into the system.

Fault model The choice of the fault model is critical since real faults depend on different factors including the target system's technology and its environmental working conditions (e.g., in space applications many faults are caused by ion radiations that are less likely to appear at ground level). The common practice is to assume that the chosen fault model (whatever it is) is sufficient, justifying this assumption with experimental data, historical data, or results published in the literature. We agree that the problem of demonstrating a tight relationship between real faults and injected models is not trivial. However, in many cases, the choice of the target fault model is simply guided by the available fault injection environment rather than a real effort to model the realistic faults. For example, a very well known fault model is the Single Event Upset, or SEU (Normand, 1996), i.e., a bit of a memory element of the system is flipped. Almost all fault injection environments use this model as their primary target fault. This wide use of the SEU model is sometimes the result of a "virtuous" decision and sometimes that of a "sinful" one. The "virtuous" reason is that the advance in integration technologies is leading to a situation where SEUs will be very common causes of failures in commercial digital systems used at ground level (Titus et al., 1998). It is therefore true that SEUs are a growing concern not only for the dependability community, but also for the manufacturers of large production ICs. The "sinful" reason is that the SEU is the easiest model to artificially inject. In general, it is enough to stop the execution of the system, inject the fault, and then let the system run until the end of the experiment. The injection of intermittent or even permanent faults is much more complicated, since after

reaching the correct fault injection time, it is necessary to keep the faulty state "active", preventing the system from overwriting its effects. This operation can cause a considerable performance overhead that can be not compatible with experiments on real-time systems. Even though many researchers claim that their fault injection systems are able to deal with different types of faults, a few of them presented convincing experimental results [omitted].

Fault locations The available fault locations should be selected considering the nature of the target system. We can identify the following situations:

- the target system is a digital system but not a microprocessor. In this case the fault locations should include all possible memory elements (e.g., flip-flops, registers, memory cells). This can create a problem in the choice of the injection mechanism, that, for example, must be able to inject faults in all the flip-flops of a circuit. This operation can be very difficult or even impossible when the target system is not an abstract model (e.g., a VHDL model) but a real circuit with a limited set of "controllable" and "observable" locations;
- the target system is a microprocessor. When using a software-based approach, the only possible fault locations are the ones directly or indirectly reachable by an assembly instruction. Usually, cache memories and many internal microprocessor registers which are not directly addressable by assembly instructions can only be injected in VHDL models or using radiation-based mechanisms. A possible extension to the available set of fault locations can be offered in some microprocessors by particular resources such as the debug ports available in the 68k and PowerPC family of microprocessors;
- the target system is a software application executed on a microprocessor. In this case the goal is not the evaluation of the hardware platform, but that of the application running on it. Faults must be injected only in those locations (RAM, CPU registers, external devices) that are or can be directly or indirectly involved in the execution of the application. For example, if the application does not use the floating point unit, it is useless to inject in its registers, since they will never be activated during the experiments.

Obviously, the nature of the target system is not enough to select the target fault locations. To be injected, locations must be reachable, and the "reachability" of a location depends mostly on the adopted fault injection mechanism. In hardware-implemented fault injection (see Section 1) only a limited set of locations can be controlled or even observed, due to the difficulty of acting directly on the system's hardware using logic analyzers, heavy-ion radiations, laser beams, or ad-hoc hardware designs (Arlat et al., 1993). When using simulation-based environments (see Section 1) based on a soft model (i.e, a VHDL or Verilog description) every location is usually reachable and can therefore be used as a potential fault location. Finally, if running software-implemented experiments (see Section 1), any software accessible hardware location can be controlled using software instructions and therefore used as a candidate fault location. Another issue that needs to be taken into account is the physical dependency among locations. Equivalent fault locations, i.e., locations that, if corrupted, will by construction cause the same effect, mast be carefully identified and considered. For example in the case of a stack the question is: should all words in the stack injected with faults or it is enough to inject in the top of the stack, since all faults in the other words will be "activated" only when the corrupted word is popped out of the stack? The problem in considering these types of problems is that it can be very hard to "automate" the process of selecting the best fault locations , and, at the same time, a manual approach would be too time-consuming to be effective. Again, the solution probably lies in finding the best set of locations that can effectively represent the location space and that correctly represent the type of results we are looking for. Once all possible fault locations have been selected it is possible to generate the final fault list.

Fault list As already introduced, in a fault injection experiment the target fault list cannot be an exhaustive set since the fault space is usually very large or even assumed to be infinite. Therefore, location and injection time of each injected fault are usually sampled from the set F of all possible fault locations and time. Sampling according to an uniform distribution of fault locations and injection time is an easy choice, but its effects should be carefully considered. The main risk is that, in the final results, each fault location will contribute with the same weight to the dependability figures. As an example, let us consider a software application containing a function repeatedly executed. Choosing a uniform distribution of faults on the application code means assigning the same weight to all instructions of the code, regardless how many times each of them is executed. This can lead to misleading results, since the function that is executed many times during the experiment will probably be more critical than a piece of code executed only once. A possible solution to obtain more realistic results is to relate the distribution of the selected fault locations to the workload or usage statistics of the parts of the system to which they belong. One of the most important goals is to avoid redundant experiments, i.e., injected faults that lead to equivalent sensitizations of the fault-tolerant algorithm or mechanism, which can lead to a great waste of work and resources. The problem is that the number of fault injection experiments needed to estimate the coverage with a reasonable confidence interval can still be extremely large. The question is: what is the alternative (or right balance) between considering a small set of elaborate and focused test cases (deterministic selection) versus the simple reliance on a statistical approach (probabilistic selection)? It is well known that probabilistic selection of the faults is required when fault injection is used to rate the behavior of a target system in the presence of faults. On the other hand, tailored and focused test cases are mandatory when fault injection experiments aim at revealing design flaws. Still, no clear inclination exists for both of these two alternatives when dealing with dependability benchmarking. An interesting approach to reduce the complexity of the fault list is the one suggested in (Benso et al., 1998c; Titus et al., 1998) and (Berrojo

et al., 2002), where the authors propose a set of collapsing rules based on the analysis of a fault free run of the system. With this approach it is then possible to reduce the number of faults to inject into the system without decreasing the accuracy of the results. The basic idea is to avoid the injection of faults whose behavior can be foreseen, like those faults that will be certainly detected by at least one of the system's error detection mechanisms, those belonging to certain fault equivalence classes, or those that will produce a predictable effect on the system's behavior (e.g., injecting a temporary fault on a register before a write operation on the same register is useless because the fault will be overwritten and never activated).

THE ADVICES TO APPLY TEMPERANCE IN FAULT INJECTION EXPERIMENTS ARE

- Choose a realistic fault model, considering the actual target system and, if necessary, its operating conditions. Do not choose a fault model only because it is easy to inject or it is the default model on the available tools;
- always take into account the real fault occurrence probability of the chosen model;
- compute the cardinality of your fault space. Claiming that the fault space is infinite will make your results totally useless because it will not be possible to evaluate how the injected set of faults is statistically representative of the whole fault space;
- apply meaningful fault sampling methods instead of resorting to simple uniform selection:
  1. by a statistical sampling, if the fault space distribution is known;
  2. taking into account the specific structure of the hardware and software under investigation:
     - (a) faults should never be injected in unused parts of the system;
     - (b) injection should focus on the most critical/stressed parts of the system;
  3. if the faults to be injected are randomly sampled from a larger list, the experiment should always be repeated with different fault lists until the required level of confidence is reached;
- there are always faults whose effect is known a-priori. These faults must always be identified, considered in the fault list, but never actually injected;
- there are always equivalent faults in terms of time (same location but different injection time) and/or space (different fault location but same effect on the system). These faults must always be identified using fault-collapsing techniques to minimize the number of actual injections.

3.2 *Justice (Activations)*

" ... and justice is the virtue of the soul as a whole; of each part never failing to perform its own function and that alone. To ask, now, whether justice or injustice is the more profitable becomes ridiculous".

Plato, the Republic

As we know, in a canonical fault injection setting like the one depicted by the FARM model of Section 2, high relevance is attributed not only to the fault list but also to the input stimuli that must ideally exercise the system with respect to all possible faulty conditions (the relationship between workload and error rates has been often highlighted (Chillarege and Bowen, 1989)). Since it is often impossible to exhaustively activate all possible operating modes in a system or application, the so-called activation set (A) must be chosen paying attention to two main requirements:

- representing the actual inputs the system will get during its operational lifetime;
- focusing on the particular goals of the fault injection experiment.

This is a typical dualism also faced in the field of hardware/software test generation. In the case of software, for example, this opposition between "functional" and "extensive" testing is visible in two of the mainstream approaches for software validation (Kuball et al., 2002), namely statistical software testing and code coverage. While the former aims at individuating (through a probabilistic model) a proper subset of the input space that is representative enough of the typical application usage scenarios, a maximization of code coverage (amount of source code executed at least once by the chosen input sets) goes in the opposite direction of extensive testing. Failure in choosing an activation set able to simulate all the critical parts of a target system (or application) could result in two main consequences for the experiment:

- *incomplete results*: the gathered dependability figures are not representative of the whole application, but only of the particular execution flow(s) covered by the chosen inputs. It would be wrong and risky to infer information about the dependability of the application (or even worse, of the system) from the results obtained with an incomplete activation set;
- *no effect faults*: faults injected in those parts of the system or application that will not be activated during the experiment will result in "no effect" faults, i.e., they will not cause any misbehavior in the system. The fault injection experiment will therefore categorize them as harmless faults. This is a misleading result since the fault might be destructive when using a different activation set.

In general, the effectiveness of a chosen activation set could be quantified with a measurement of the portion of fault list it is able to cover. To this purpose, different target systems could rely on different kinds of metrics: in a generic digital system the provided activation set should be integrated with a measure of the percentage of potential fault locations that such set is able to activate. In the particular case of a microprocessor, a possible metric to exploit is the so-called instruction set coverage, i.e., the percentage of assembly instructions of the instruction set that are executed during the experiment. Yet a uniform coverage of all the instructions in the instruction set does not guarantee a uniform activation of all the microprocessor functionalities, which would be possible only with a very deep knowledge of the microprocessor's architecture and microcode. If the target is a software application, all possible execution flows deriving from its design should be activated in order to have a global figure of the application dependability. When this is not possible, then it will be necessary to at least exercise all those critical or typical functionalities that the application will need during its operational lifetime. As mentioned above, a measure of the code coverage (Kuball et al., 2002; Garg, 1994) should be provided in order to demonstrate the effectiveness of the chosen activation set.

Operational profiles The problem of identifying correct test patterns for system dependability assessment has been tackled in many ways, producing several practical approaches based on the peculiar features of targeted systems (i.e., monitoring facilities in operating systems, for software testing). The only unified, system-independent methodology seems that of operational profiles (Musa, 1993). An operational profile is a collection of information about all relevant fault-free system activities. Typical traced information items are read/write activities associated with processor registers, address bus, data bus, and memory locations in the system under test. Moreover, they may also include higher level information like the most probable expected sets of inputs that the system or application will receive, or even concern other issues that influence development, like customer/user preferences.

This translates in a set of probability distributions that identify possible rates of occurrence for typical usage scenarios or execution flows within the considered application or system. Essentially, the purpose of an operational profile is to better understand the situation in which the system or the application will be used, and then analyze this information to ensure that only faults which will produce an error are selected during the fault list generation process.

Several approaches have been proposed to cope with the creation and optimization of operational profiles. A reasonable development approach in terms of quality/cost ratio (Juhlin, 1992) consists in separating profile components that depend on the peculiar workload exercised on the system by each potential user (usage profile(s)), from the ensemble of physical and logical features representing the unvarying setting upon which particular usage profiles may be introduced (configuration profile). It is also possible to manipulate such profiles to deal directly with the above-mentioned problem of "no-effect faults". Through the use of inverted profiles of a tested system (Voas et al., 1996), it is possible to isolate subsets of inputs that are unlikely to be activated in order to rebuild the activation set and test.

Scenario-based test generation The concept of operational profile has nowadays become quite a standard for the primary modeling of systems usage, almost independently from the particular architecture being targeted. This methodology has been integrated with various input generation approaches, each one trying to propose a different way to deal with its limitations. What follows is a brief summary of approaches available to deal with meaningful test pattern generation (and therefore fault activation), distinguishing between the two main categories of hardware systems and software systems.

3.2.2.1. Hardware systems Test generation for digital electronic systems is usually composed by three main steps: selecting a description method (model of the system), developing a fault model and generating tests to detect all the faults identified by the fault model. Therefore, the efficiency of test generation (in terms of quality and speed) is highly depending on the system description and fault models which have been chosen. Due to the increasing complexity of digital circuits, classical gate-level test generation has become impractical. Moreover, it has been shown that test generation for combinational circuits is an NP-complete problem (Fujiwara, 1985). Consequently, different methodologies to model test patterns have been progressively considered:

- *functional and behavioral test synthesis methods*: such methods rely on an outline of the expected system's behavior, trying to compose the activation set starting from all main functionalities expected by the system (Gupta and Armstrong, 1985; et al., 1983). However, as such methods do not use implementation data but mainly rely on design specifications, they cannot afford good test quality measured in terms of gate-level fault coverage;
- *hierarchical test synthesis methods*: in order to maintain a better correlation between test selection and physical coverage of the system, hierarchical test generation methods have evolved (Lee and Patel, 1991; Rao et al., 1993). Such methods take advantage of higher abstraction level information, while generating tests for the gate-level faults. The advantage gained by hierarchical methods over functional ones lies in the possibility of measuring the test coverage in gate-level stuck-at faults (SAF). On the other hand, it has been shown that exclusive SAF coverage cannot guarantee high quality of testing (Soden and Hawkins, 1993; Huisman, 1993). This is due to the fact that SAF has proven unfeasible in adequately representing the majority of real IC defects and failure mechanisms;
- *decision diagrams*: good results in gate-level test generation have been achieved with decision diagrams used as model of digital circuits (Corno et al., 1995) in order to replace abstract fault models like SAF with realistic defect models. In (Ubar and Raik, 2000) the authors propose a hierarchical test generation method where three levels of modeling are exploited: high-level decision diagrams for efficient high-level system constraints generation and high-level test planning (Ubar, 1996), low-level Boolean differential equations for logic-level exact fault activation, and medium-level structurally synthesized binary decision diagrams for local test pattern generation with respect to single system modules. By using such diagrams for different system levels (RTL and gate-level), this methodology manages to provide a uniform system representation model, and the application of consistent fault activation and transportation procedures throughout all the levels.

3.2.2.2. Software In the case of software, several analytic models have been proposed for the estimation of reliability.

- *time-domain models*: these models use the failure history obtained during testing to predict the field behavior of the program, under the assumption that testing is performed in accordance with a given operational profile. However, there are some fundamental difficulties with this approach including the saturation effect of the testing process (Chen et al., 1992), and the usual difficulty in obtaining an accurate operational profile;

- *statistical testing*: in statistical software testing, estimation of software dependability is based on the execution of a predetermined number of test cases generated as a statistical sample from the operational profile (Thevenod, 1991). Operational profiles are often built by partitioning the input space into a set of so-called bins (groups of inputs that share the same characteristics), each one retaining a different occurrence probability. Statistical testing does guarantee a better performance in uniform input selection since it is highly subjected to the quality of the operational profile (Garg, 1994);
- *coverage*: other than a completely distinct methodology, coverage is intended to integrate previously defined approaches by defining a uniform metric to quantify the fraction of software application (in terms of execution paths and code blocks, for example) that has been actually executed and evaluated during the testing process. The need for coverage measurements in the validation of software dependability has been widely demonstrated (Kahn and A.W., 1953; Benso et al., 1998b). However, although coverage is needed to define some sort of threshold that delimits a sufficient amount of testing of a software system, it does not hold a straightforward interpretation for reliability. In this connection, efforts to combine coverage with other test generation models are needed (Garg, 1994; Howden, 1997).

Another interesting approach that goes in the direction of providing high level fault activation for software systems can be obtained by shifting our attention from the input space to that of faults.

The stress- and path-based fault injection methods (Aidemark et al., 2001; Tsai et al., 1999) provide an effective solution for generating a fault list that correctly matches system's inputs, by analysis of the application under evaluation at different stages. This goes in the direction of generating a fault list that gets highly (if not completely) activated. The stress-based method monitors run-time workload activity at the system level in order to lead fault injection to the locations and times of greatest workload activity. This technique guarantees high activation rates for the fault list.

The path-based method analyzes the system under workload from an application standpoint during the development phase of the software application, by considering control-flow and resource usage information. It can therefore be used to individuate those portions of code that need improvement. However, it is very dependent on the peculiar instruction set and code format of the platform. It usually requires pre-injection analysis to be properly applied. Path-based analysis guarantees complete activation of the fault list.

The path-based approach is usually more costly than the stress-based one, both in terms of time and setup efforts. Such additional cost is fixed and quantified with the number of paths to evaluate. However, is demonstrated the existence of a threshold value for the number of paths, over which such overhead starts to get amortized, thus justifying the adoption of path-based over stress-based in particular case studies.

THE ADVICES TO APPLY JUSTICE IN FAULT INJECTION EXPERIMENTS ARE

- If the goal is to evaluate the hardware reliability, it is necessary to choose a workload able to activate the majority (possibly 100%) of the system functionalities:
  (1) it is possible to use end-of-production test patterns or verification patterns, if available;
- if the goal is to evaluate the hardware/software system, then the workload has to guarantee the highest possible coverage of the mission software functionalities by:
  (1) computing the code coverage guaranteed by the chosen workloads;
  (2) choosing a set of workloads representative of the actual mission. If available, useful knowledge can be extracted from historical information of previous systems working in the same conditions.

3.3 Prudence (Readouts)

"Prudence is practical wisdom and judgment regarding the choice and use of the best ways and means of doing good".

Plato, the Republic

Readouts strongly depend on the type of fault injection environment. Simulation-based environments usually allow more detailed readouts, whereas software- or hardware-implemented solutions in most of the cases allow recording only a limited amount of information (usually gathered from the system's primary outputs only). Moreover, since logging the system's behavior can be very expensive (in terms of execution time, hardware, or complexity), the choice of the readout mechanism can be constrained by the type of target system. For example, in case of a real-time system, it is necessary to choose readout mechanisms that do not influence the timing properties of the system itself.

The readouts obtained during a fault injection experiment allow therefore elaborating results that are intrinsically strongly related to the particular level of abstraction of the target system and fault injection setup. A lot of attention has therefore to be paid if one of the goals is to deduce dependability results for an abstraction level (for example the gate-level) different from the one used in the experiment (for example behavioral-level). Some papers tried to do this [omitted] but only a few of them gave a formal and complete statistical demonstration of the correctness of their final results.

THE ADVICES TO APPLY PRUDENCE IN FAULT INJECTION EXPERIMENTS ARE

- Whenever possible, implement or choose a fault injection tool able to work at the same level of abstraction at which I need to obtain results;
- observation points should be carefully selected to precisely represent the actual portion of the system that is critical for the experiment results.

3.4 Courage (Measures)

"... Courage we find in the soldiers; courage is the true estimation of danger, and that has been ingrained in them by their education.".

Plato, the Republic

Courage is maybe the most important virtue when trying to interpret the fault injection results obtained during the experiments.

The fault tolerance coverage estimations obtained through fault injection experiments are estimates of conditional probabilistic measures characterizing the dependability. Coverage is defined as the probability of system recovery given that a fault exits. Deriving coverage information from the collected readouts is not trivial and requires a good statistical background. Moreover it is necessary to carefully analyze how the data has been obtained. The first issue has to do with the *no-effect* faults that can occur. A no-effect fault does not cause any modification in the system's behavior. Example of no-effect faults are:

- a fault present in a memory location that is never accessed (e.g., a variable that will never be read or written again, or an instruction that the given activation set never executes);
- a transient fault overwritten by a write operation on its location;
- a fault masked by the system.

These faults reduce the efficiency of the assessment process because the fault injection experiments for these faults provide no useful information and yet require the maximum amount of resources. A lot of experiments involving no-effect faults must be therefore discarded.

As an example, let us consider an application with the control flow graph presented in Fig. 2. Let us assume to define a fault injection campaign that requires the injection of 1,000 random faults in the code segment of the application with an activation set that causes the application to execute the following flow: Begin-B-End. Approximately the same number of faults will be injected in A and in B. However, since the A block of instructions is never executed, at the end of the campaign we can expect all the faults injected in A (50%) to be classified as no-effect.

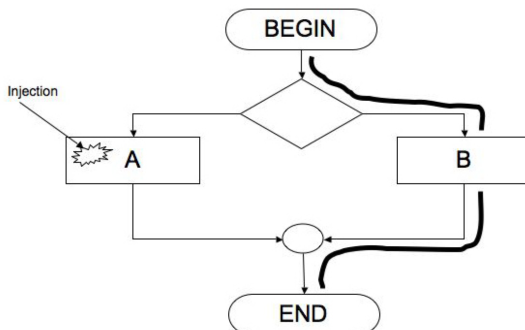

Fig. 2. Simulation Results

If the purpose of the experiment was to evaluate the dependability of the application with a particular set of inputs, then the result can be correct. On the other hand, if the goal was the evaluation of the application in general, the meaning of the collected results might be misunderstood since there are no results about the application behavior when faults are injected in A during its execution.

We analyzed before that avoiding a random generation of the target fault list can dramatically reduce the problem of having useless or predictable results (see Section 3.1.3). Nevertheless, when using collapsing techniques on the target fault list it is very important to remember that the collapsed (and therefore discarded) faults should still be considered in the final measures. If a fault has been discarded because its effect was predictable, than in the final results its contribution must be counted in the corresponding class. Explicit description of the statistics, computations and assumptions must always be included in the results of a fault injection experiment, because they are the only means to evaluate the confidence in the final results.

THE ADVICES TO APPLY COURAGE IN FAULT INJECTION EXPERIMENTS ARE

Use real statistics and not only the minimum amount to make the results look professional:

- never provide absolute results, but always couple them with an error and confidence estimation;
- always correlate the results with the operating conditions;
- be very careful in generalizing results;
- always take into account the actual fault occurrence probability. A very destructive fault with a very low occurrence probability may be not that critical after all;
- evaluate the results in a critical way in order to understand if they reflect the actual system behavior or they are biased by an error in the fault injection setup.

4. CONCLUSIONS

"Errare umanum est, perseverare diabolicum"

In this paper the authors discussed the most common methodological errors that can be committed (or have already been committed) in setting up and presenting the results of a fault injection campaign. The purpose is not to blame other researchers, but to summarize those malicious problems that can make fault injection results unreliable, and to propose some possible solutions. The four Cardinal Virtues of fault injection are: Temperance in the choice of the fault model and fault list, Justice in the selection of the activation set, Prudence in the selection of the readouts points, and Courage when elaborating and interpreting the final results. The authors believe that two of the most significant advances in fault injection techniques can be found in (i) the use of statistics in both the definition of the fault list and the interpretation of the results, and in (ii) the use of coverage metrics (as code coverage, instruction set coverage, etc.) to relate the final dependability results to the actual coverage of the system functionalities that the experiment allowed.

REFERENCES

Aidemark, J., Folkesson, P., and Karlsson, J. (2001). Path-based error coverage prediction. In *7th IEEE On-Line Testing Workshop, IOLTW'01*, 14–20.

Arlat, J., Costes, A., Crouzet, Y., Laprie, J., and Powell, D. (1993). Fault injection and dependability evaluation of fault-tolerant systems. *IEEE Trans. Comput.*, 42(8), 913–923.

Benso, A., Civera, P., Rebaudengo, M., and Sonza Reorda, M. (1998a). An integrated hw and sw fault injection environment for real-time systems. In *IEEE Defect and Fault Tolerance Symposium, DFT'98*, 117–122.

Benso, A. and Prinetto, P. (2003). *Fault Injection Techniques and Tools for Embedded Systems Reliability Evaluation*. Kluwer Academic Publishers.

Benso, A., Prinetto, P., Rebaudengo, M., and Sonza Reorda, M. (1998b). Exfi: a low cost fault injection system for embedded microprocessor-based boards. *ACM Transactions on Design Automation of Electronic Systems*, 3(4), 626–634.

Benso, A., Rebaudengo, M., Impagliazzo, L., and Marmo, P. (1998c). Fault-list collapsing for fault-injection experiments. In *Annual Reliability and Maintainability Symposium*, 383–388.

Berrojo, L., Gonzalez, I., Corno, F., Reorda, M., Squillero, G., Entrena, L., and Lopez, C. (2002). New techniques for speeding-up fault-injection campaigns. In *IEEE Design, Automation and Test in Europe Conference and Exhibition, DATE'02*, 847–852.

Carreira, J., Madeira, H., and Silva, J. (1998). Xception: a technique for the experimental evaluation of. *IEEE Trans. Softw. Eng.*, 24(2), 125–136.

Chen, M., Horhan, J.R., P., M.A., and J., R.V. (1992). A time/structure based model for estimating software reliability. Technical report, Purdue Univ. Technical Report SERC-TR-117-P.

Chillarege, R. and Bowen, N. (1989). Understanding large system failures-a fault injection experiment. In *19th IEEE International Symposium on Fault-Tolerant Computing, FTCS'89*, 356–363.

Choi, G. and Iyer, R. (1992). Focus: an experimental environment for fault sensitivity analysis. *IEEE Trans. Comput.*, 41(12), 1515–1526.

Clark, J. and Pradhan, D. (1995). Fault injection: a method for validating computer-system dependability. *IEEE Computer*, 28(6), 47–56.

Corno, F., Prinetto, P., Sonza Reorda, M., Glaser, U., and Vierhaus, H. (1995). Improving topological atpg with symbolic techniques. In *13th IEEE VLSI Test Symposium, VTS'95*, 338–343.

et al., J.S. (1983). Speed up behavioral atpc. In *30th IEEE/ACM Design Automation conference, DAC'83*, 92–96.

et al.., V.P. (2006). Reduced instrumentation and optimized fault injection control for dependability analysis. In *IEEE International Conference on Very Large Scale Integration*, 291–396.

Fujiwara, H. (1985). *Logic Testing and Design for Testability*. MIT (Cambridge).

Garg, P. (1994). Investigating coverage-reliability relationship and sensitivity of realiability to errors in the operational profile, software testing, reliability and quality assurance. In *First International Conference on Software Testing, Reliability and Quality Assurance*.

Gupta, A. and Armstrong, J. (1985). Functional fault modeling and simulation for vlsi devices. In *22nd ACM/IEEE Design Automation Conference, DAC'85*, 720–726.

Guthoff, J. and Sieh, V. (1995). Combining software-implemented and simulation-based fault injection into a single fault injection method. In *25th IEEE International Symposium on Fault-Tolerant Computing, FTCS'95*, 196–206.

Howden, W. (1997). Systems testing and statistical test data coverage prediction. In *20th Computer Software and Applications Conference, COMPSAC '97*, 500–504.

Huisman, L. (1993). Fault coverage and yield predictions: do we need more than 100%. In *3rd European Test Conference, ETC'93*, 180–187.

Jenn, E., Arlat, J., Rimen, M., Ohlsson, J., and Karlsson, J. (1994). Fault injection into vhdl models: the mefisto tool. In *24th IEEE International Symposium on Fault-Tolerant Computing, FTCS'94*, 66–75.

Juhlin, B. (1992). Implementing operational profiles to measure system reliability. In *3rd International Symposium on Software Reliability Engineering*, 286–295.

Kahn, H. and A.W., W. (1953). Methods of reducing sample in monte carlo computations. *Journal of the Operations Research Society of America*, 1(5), 263–278.

Karlsson, J., Folkesson, P., Arlat, J., Crouzet, Y., Leber, G., and Reisinger, J. (1994). Evaluation of the mars fault tolerance mechanisms using three physical fault injection techniques. In *3rd IEEE nternational Workshop on Integrating Error Models with Fault Injection*, 21–22.

Kuball, S., Hughes, G., and Gilchrist, I. (2002). Scenario,based unit testing for reliability. In *Annual Reliability and Maintainability Symposium*, 222–227.

Lee, J. and Patel, J. (1991). Artest: An architectural level test generator for data path faults and control faults. In *IEEE International Test Conference, ITC'91*, 729–738.

Mei, Tsai, T., and Iyer, R. (1997). Fault injection techniques and tools. *IEEE Computer*, 30(4), 75–82.

Musa, J. (1993). Operational profiles in software-reliability engineering. *IEEE Softw.*, 10(2), 14–32.

Normand, E. (1996). Single event upset at ground level. *IEEE Trans. Nucl. Sci.*, 43(6), 2742–2750.

Rao, S., Bi, and Armstrong, J. (1993). Hierarchical test generation for vhdl behavioral models. In *4th European Conference on Design Automation, EDAC'93*, 175–182.

Soden, J. and Hawkins, C. (1993). Quality testing requires quality thinking. In *IEEE International Test Conference, ITC,93*, 596.

Thevenod, P. (1991). From random testing of hardware to statistical testing of software. In *5th Annual European Computer Conference onAdvanced Computer Technology, Reliable Systems and Applications, CompEuro'91*, 200–207.

Titus, J., Combs, W., Turflinger, T., Krieg, J., Tausch, H., Brown, D., Pease, R., and Campbell, A. (1998). First observations of enhanced low dose rate sensitivity (eldrs) inspace: One part of the mptb experiment. *IEEE Trans. Nucl. Sci.*, 45(6).

Tsai, T., Mei, Zhao, H., Kalbarczyk, Z., and Iyer, R. (1999). Stress-based and path based fault injection. *IEEE Trans. Comput.*, 48(11), 1183–1201.

Ubar, R. (1996). Test synthesis with alternative graphs. *IEEE Des. Test. Comput.*, 13(1), 48–57.

Ubar, R. and Raik, J. (2000). Efficient hierarchical approach to test generation for digital systems. In *1st IEEE International Symposium on Quality Electronic Design, ISQED'00*, 189–195.

Velazco, R., Rezgui, S., and Ecoffet, R. (2000). Predicting error rate for microprocessor-based digital architectures through c.e.u. (code emulating upsets) injection. *IEEE Trans. Nucl. Sci.*, 47(6), 2405–2411.

Voas, J., Charron, F., and Miller, K. (1996). Investigating rare-event failure tolerance: reductions in future uncertanity. In *IEEE High-Assurance System Engineering Workshop*, 78–85.