

# **DBench (Dependability Benchmarking)**

(IST project: References, web address)

Jean Arlat<sup>\*</sup>, Diamantino J.G. Costa<sup>\*\*</sup>, Mario DalCin<sup>\*\*\*</sup>, Pedro Gil<sup>+</sup>, Karama Kanoun<sup>\*</sup>,  
Jean-Claude Laprie<sup>\*</sup>, Henrique Madeira<sup>++</sup> and Neeraj Suri<sup>+++</sup>

## **Partners**

- \* LAAS-CNRS, Toulouse, France (Coordinator)
- +++ Chalmers University of Technology, Sweden
- \*\* Critical Software, Coimbra, Portugal
- ++ University of Coimbra, Portugal
- \*\*\* Friedrich Alexander University, Erlangen-Nürnberg, Germany
- + Polytechnical University of Valencia, Spain

## **Sponsor**

Microsoft, Cambridge, UK

## **Industrial Advisory Board**

Astrium (France), Caldera (Germany), CMU (USA), INDRA (Spain), Oracle (Portugal),  
Saab Ericsson Space (Sweden), Thales (France).

## **Contact**

Karama Kanoun  
LAAS-CNRS, 7, Avenue du Colonel roche  
31077 Toulouse Cedex-4, France  
kanoun@laas.fr

**Starting date:** January 2001

**Project duration:** 36 months

## **Abstract**

The **DBench** (Dependability Benchmarking) project aims at defining a conceptual framework and an experimental environment for benchmarking the dependability of COTS and COTS-based systems. It will provide system developers and end-users with means for 1) characterising and evaluating the dependability of a component or a system, 2) identifying malfunctioning or less weak parts, requiring more attention, 3) tuning a particular component to enhance its dependability, and 4) comparing the dependability of alternative or competing solutions. The two final objectives that will be produced at the end of the project are a report presenting the concepts, specifications and guidelines for dependability benchmarking and a set of dependability benchmark prototype tools. The prototypes will be made widely available (e.g., through the web whenever possible) to promote their adoption by an audience as wide as possible.

## 1. Introduction

The objective of the DBench project is to define and validate dependability benchmarks for largely deployed computer systems. The pervasive use of commercial off-the-shelf (COTS)<sup>1</sup> components in a wide range of computer systems (e.g., embedded systems or database/web servers based on COTS operating systems and platforms), and the increasing dependence of our society on such systems, make it urgent to address the dependability of COTS components and COTS-based systems. In addition, the use of COTS is extended to systems targeted for high dependability environments (i.e., life-critical or business-critical applications). Although information about functionality and performance of COTS components is well characterized, there are no existing guidelines for characterizing the *behavior* of these components in the presence of internal and/or external faults. This not only constrains the developers ability to utilize COTS for designing dependable systems, it also comes to the situation that objectively evaluating the behavior of a COTS-based system is currently either an *ad hoc* process or essentially non-existent.

Dependability benchmark enables to quantify system survivability (its ability to resist to faults), through well-specified measures. Weak parts can thus be identified and emphasis may be put on enhancing their ability to resist to faults (by tuning the system to enhance its dependability). Hence, knowing the behavior of systems in presence of faults will allow construction of dependable systems

Another important objective of DBench is to help the computer industry to accelerate the improvement of computer's (including hardware and software) resilience to faults, even for general-purpose systems with no special fault tolerance mechanisms. Moreover, the results of DBench can constitute useful inputs for the standardization bodies to define guidelines for the certification of safety-critical systems incorporating COTS components.

Given the complexity and cross-cutting issues (hardware, operating system, middleware, application software) involved in the design of a *computer system*, the use of pure modeling alone or analytical techniques is not adequate, and we propose developing an experimental benchmarking approach. Our work focuses on COTS based on Windows and Linux. We consider systems widely used in i) general purpose computing (Windows, Linux), ii) database and webs servers (Oracle DBMS), and in embedded systems (refereed to as Windows-E/Linux-E). The proposed approach will provide means for: i) assessing the dependability of a component or a system, ii) identifying malfunctioning or weak parts, requiring more attention, iii) tuning a particular component to enhance its dependability, and iv) comparatively assessing the dependability of alternative or competing solutions.

The paper is organized in seven sections. Section 2 gives a global overview of the work that will be performed within DBench. More details about this work are given in sections 3 to 6. Section 7 addresses results evaluation and relevance.

## 2. Description of work

For dependability benchmarks, although private discussions with large system provider companies and some published work reveals that some relevant work is undertaken in a sparse manner, the current state-of-the-art is still immature, lacks conformity or approval across companies and is basically globally non-existent. We utilize the experience from the performance benchmarking community, especially the issues developed in that area on *what* to measure and *how* to measure attributes. Typically, a performance benchmark is a workload to be run on a system and the output is a performance measure [9].

Analogously, we define a dependability benchmark as a set of **workload** and **faultload** to be executed on the system to ascertain dependability measures or indicators on the ability of a system to cope with both internal and external faults. Typical measures that will be **directly observable on the target**

---

<sup>1</sup> Indeed, our work will address more precisely off-the-shelf (including open source software) and commercial off-the-shelf components. However, we will indifferently use the term COTS to design either of them.

system will be identified during the project. Our starting point will be to consider classical measures already established in the *dependability arena*, (i.e., error detection efficiency, error detection latency, time to diagnosis, failure modes, recovery factors) and metrics adapted from the *performance benchmark arena* such as system response time or number of transactions, in presence of faults. We will also evaluate **system level measures**, such as reliability, availability and safety, obtained from models (incorporating the experimental measures as parameters as well as parameters provided from elsewhere). The results of dependability benchmarks will characterize the property of a system such that reliance can be justifiably placed on the service it delivers.

To facilitate project control and monitoring, the work is structured in four workpackages in such a way that their sequence reflects the project progress:

- WP1: Definition of the conceptual framework for system benchmarking.
- WP2: Identification and evaluation of the enabling technologies.
- WP3: Benchmark definition and application to pilot experiments in order to develop, experiment and validate benchmark prototypes.
- WP4: Consolidation of the conceptual framework with the experimental results.

The **conceptual framework** will settle the foundations of dependability benchmarks. It addresses several important aspects such as the need for an overall and global system viewpoint to correctly select the measures to be evaluated and interpret the results. To put into practice the conceptual framework, **enabling technologies** will be investigated and adapted in some respects. They encompass: measurements to be performed on the target system, fault representativeness and workload and faultload selection. **Experiments** will be performed to achieve a comprehensive analysis: different dependability benchmark prototypes will be defined and developed for two major application-areas (embedded and transactional applications). The benchmark prototypes will actually be implemented in two different families of COTS operating systems (Windows and Linux), allowing a cross evaluation of the concepts and the enabling technologies in a true dependability benchmark context. The final goal of the experiments is the validation of the dependability benchmark prototypes, in the sense of assuring that the benchmarks results represent a practical and meaningful characterization of the dependability properties of the target systems, both from the end-user and the system developer's point of view. Finally, the **consolidation** of the whole set of results will allow to push further the enabling technologies and to finalize the benchmark concepts and prototypes.

### 3. WP1: Conceptual Framework

The conceptual framework will address the most relevant issues involved in dependability benchmarking. It is a fundamental part of the project: the studies and experiments that follow will abide to common guidelines. Building upon relevant advances in performance and dependability evaluation, WP1 will be decomposed into three main items related respectively to i) assessing the state-of-the-art in performance benchmarking and dependability characterization, ii) identifying and investigating the concepts for dependability benchmarking, iii) setting up the foundations for the project development. In the sequel, we concentrate on the detailed description of the activities concerning the second item including: benchmark objectives and utilization, properties, measures and conduct.

For what concerns **benchmark objectives and utilization**, special attention will be devoted to distinguish the commonalities and specificities attached to the various benchmarking attributes. An important dimension is related to the point of view considered: an end-user has usually a perspective that is different from a system developer (or provider) perspective. In addition, the system boundaries can vary for different end-users and, for systems composed of systems. Moreover, very different levels of, for example, accessibility, controllability, and knowledge of the system are to be expected. Thus, different kinds of users of dependability benchmarks have different needs and requirements. Benchmarks can be used for instance to:

- *Assess* the dependability of a component or a system, requiring the benchmarks to produce just global aggregate numbers, but with comprehensive fault models.

- **Identify** malfunctioning or less robust parts, requiring more attention and perhaps necessitating a change at the architectural level.
- **Tune** a particular component to enhance its dependability (using wrapping), or tune a system architecture (by adding fault tolerance mechanisms or spare units, for example) to ensure an appropriate dependability level.
- **Compare**, grade or rank the dependability of alternative or competitive solutions.

Several **properties of dependability benchmarks** are relevant. Examples of such properties:

- Portability, an important requirement but quite hard to attain due to the intricate nature of faults and their consequences.
- Modularity, for ease of adaptability and expandability.
- Ease-of-use and simplicity, for wide adoption.
- Non-interference and non-damaging for simple use.
- Repeatability and reproducibility for high confidence in the results.

The extent to which each of these, and other, properties can be attained will be clarified. Most likely, it will not be possible to satisfy all of them, and a trade-off shall be made according to the measures to be evaluated.

Concerning **benchmark measures**, emphasis will be put on identifying and defining meaningful measures for the several users of a dependability benchmark (including system developers and end-users). Some qualitative or statistical measures can be obtained directly by measurements (by executing a benchmark on the target system) and some others can be derived from these results with the help of modeling.

The measures obtained by measurements characterize the reaction of the system to faults, given the fact that faults are present. We will consider both classical dependability measures and measures adapted from the performance area. Examples of measures that can be obtained directly by measurements and processing are: error detection efficiency, error detection latency, time to diagnosis, failure modes, recovery factors, time to initialize or restart the system, as well as system response time or number of transactions, in presence of faults.

In addition to the reaction of the system to these faults, the second set of measures, derived via modeling, accounts also for other processes, such as the occurrence of faults and maintenance policies. When fed to probabilistic models, the statistical measurements allow the evaluation of probabilistic dependability measures like reliability, availability and safety. Our previous work shows the powerfulness of modeling for evaluating the dependability of complex real-life systems (see e. g., [13, 14]). Additional relevant measures may be identified during the project.

For **conducting a benchmark**, users need explicit guidelines. In the same way, guidelines are needed for understanding and correctly interpreting the results. A typical benchmark will consist of a series of runs applied to the target system(s). Each run will include several phases: i) initialization, ii) workload submission without fault injection, iii) workload submission with fault injection (using the results of the previous phase as a reference), iv) observation of the target system reactions, v) integrity checking before the beginning of the subsequent run, etc. As already pointed out, it is anticipated that the proposed benchmarks have to be adapted to the specific target system(s) and also according to the users requirements. These phases and their co-ordination have to be carefully designed and evaluated. Tools are needed that support and automate large parts of this development. One of the goals of this activity is, therefore, to develop a comprehensive and tractable model on “how to conduct benchmarking” that delivers i) guidelines for proven best practices to set-up benchmarks and apply them efficiently, and ii) assistant tools for conducting the benchmark execution. The development of such a comprehensive supporting framework is highly desirable both to serve as a unifying basis to carry out the various experiments within the project and also to facilitate technology transfer and take-up initiatives at the end of the project. For example, a knowledge database can help specify the scope and objectives of an assessment, define success criteria or estimate time and costs. The database should be configurable and may identify several phases and iteration cycles for benchmark activities.

There are several available tools that allow for a precise definition, design and evaluation of such a framework (e.g., see [11]).

#### **4. WP2: Enabling Technologies Identification and Evaluation**

In addition to a sound conceptual and experimental foundation, developing dependability benchmarks requires also appropriate enabling technologies so as to derive meaningful results.

One major extension of dependability benchmarking with respect to classical performance benchmarking concerns the characterization of the behavior of a target system in the presence of a specific faultload (in addition to the workload). Thus, fault injection will play a central role for developing dependability benchmarks. However, the technologies that were classically used in fault injection experiments (mainly aimed at assessing the fault tolerance mechanisms of a particular system) are not directly usable for benchmarking purpose. Indeed, to be adopted and so as to provide meaningful quantitative results, benchmarking calls for the proposal of a new set of easy-to-handle, yet reliable, technologies. We will address this fundamental issue by attempting to answer the following specific questions:

- What are the relevant measurements to perform on the target system in order to derive meaningful measures.
- How to assess the representativeness of the faults to be injected.
- What are the relevant workload and faultload to consider.

##### ***4.1 Measurements***

It is essential to identify the events to be observed as well as how to observe them and when to measure their effects. This will define the measurements to be performed on the target system. Typical measurement results will be, among others, failure modes (hang, crash, etc.), time to failure or indications about possible weak points in the system design. Our experience relative to processing of real software failure data [12] will be helpful.

Both distributed and non-distributed systems will be targeted. A central question when considering a distributed system, is how to make coherent distributed observations and how to co-ordinate them; for example, data in various nodes should be checked for consistency.

Specific set of benchmarks will be defined, each tailored to the most relevant measures for the identified users (system developer and end-user). This will include the analysis of various usage profiles. Also, to make the benchmarks easily exploitable and usable, a graphic user interface will be developed. For example, to achieve high coverage the visual representation of execution and error propagation paths can be used to identify locations where test cases will effectively stimulate the system.

##### ***4.2 Fault representativeness***

As many fault injection experiments revealed a wide discrepancy among the behaviors caused by several candidate fault injection techniques (e.g., see [15, 19]), fault representativeness is indeed a key issue when considering benchmarking. Besides some recent efforts towards these ends (e.g., [7, 8, 24]), the results currently available are still very limited. Accordingly, work is needed to identify and validate the nature of the erroneous behaviors that are induced by various classes of faults. For example, the characteristics of an error can be established by exploiting the execution trace of a faulty system.

As it is expected that similar error patterns may often originate from various distinct causes (faults), for cost-efficiency, we advocate that, in the case of benchmarking, fault injection should rather aim at producing directly such error patterns rather than focusing on their potential multiple causes.

Elaborating on our previous experience (e.g., see [5, 7, 15, 16]), we will carry out specific experiments using tools already developed by the partners of the consortium, encompassing physical fault injection [17], software-implemented fault injection (SWIFI) [1, 2, 6] and software mutation [4]. We will also

elaborate on previous relevant related work (e.g., see [18] [3, 8, 10, 21]). The aim is to investigate what are the types of faults to be injected. Specific aspects to deal with include: distribution among the observed error patterns and their multiplicity, dynamics and timing characteristics of these error patterns. As already pointed out when dealing with repeatability, instead of exact matching between the many faults considered and the erroneous situations observed, that is indeed hardly achievable — and in fact not necessary in our context — we will rather found our investigation of the representativeness issue on a statistical/probabilistic basis.

#### **4.3 Workload and faultload selection**

Considering the **workload**, as a first basis, we will be using the type of workloads that conform to the well-identified performance benchmarks. It is anticipated that adaptations or enhancements will concern the definition of workloads primarily aimed at selectively activating specific services of the target OSs. In particular, in that case we will consider also workloads to test the nominal behavior as well as exceptional behavior provided by the services. The main innovative work associated to this task will be devoted to the investigation of how to combine the faultload and the workload according to the objectives of the benchmark on a given target system: either evaluation of dependability measures, or testing of its behavior in the presence of faults.

Considering the **faultload**, it is well known that probabilistic selection of the faults to be injected is very much needed when fault injection experiments are meant to rate the behavior of a target system in the presence of faults. For example for evaluating a coverage parameter (efficiency) of a fault tolerance mechanism, a fault tolerance property or dependability attributes. On the other hand, tailored and focused test cases are mandatory when the fault injection experiments are to reveal design flaws and weaknesses (e.g., in fault tolerance mechanisms). Still, no clear inclination exists for either of these two alternatives when dealing with dependability benchmarking. Indeed, the set of faults used in a benchmark should ideally be selected such that each test case generates a unique and significant error pattern; this way, the test cases would exercise as many features of the target system as possible. However, such an involvement might impair the portability and ease-of-use properties that should be fulfilled by a benchmark.

In that context, one important practical issue is the level of synchronization of the faultload with respect to the workload. Indeed, synchronization allows for specific test cases to be applied, but at the price of some necessary knowledge and detailed analysis of the target system(s). On the other side, “random” injection is easier to implement, but at the expense and risk of slow convergence or possible bias of the results. Along these lines, another dimension of the same problem concerns the impact on the results of the clustering of the faultload into classes that are prone to lead to distinct test cases. 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 argument (probabilistic selection).

Elaborating on our previous work [18], we will specifically investigate and validate means (e.g., bias reduction techniques and confidence interval statistics) to support a sensible assessment and to identify objective compromises. We will also address the repeatability of experimental assessment. As deterministic repeatability cannot be achieved when dealing with complex systems, we will investigate to what extent statistical fault injection and probabilistic interpretation of the results can offer rational means to compensate this lack of repeatability. In order to support these assessments, we propose to investigate several techniques for identifying test cases with high coverage. These techniques will make use of methods, including formal methods based approaches, to conduct pre-injection analysis of the target system [20, 22].

### **5. WP3: Benchmark Definition, Experimentation and Validation**

In order to achieve a comprehensive work, different dependability benchmark prototypes will be defined and specified for two major application areas: embedded and transactional applications. Additionally, the benchmark prototypes will actually be developed for two different families of COTS operating systems (Windows and Linux), allowing a cross evaluation of the concepts (WP1) and the

enabling technologies (WP2) in a true dependability benchmark context. The final goal of this comprehensive set of experiments is the validation of the dependability benchmark prototypes in the sense of assuring that the benchmarks results represent a practical and meaningful characterization of the dependability properties of the target systems, both from the end-user and the system developer's points of view.

The proposed experimental set-up comprises the cross-cutting issues involved in the characterization of the large variety of today's computer systems: representative application areas (embedded and transactional applications), widely used COTS operating systems (Windows and Linux), and available in different versions (Windows/Windows-E and Linux/Linux-E) for a large range of hardware platforms. The combination and the cross exploitation of these multiple dimensions in the planned experiments do represent a realistic portrait of the computer systems industry.

This workpackage is organized in three major tasks, comprising the logical steps of benchmark definition, experimentation, and validation.

### **5.1 Benchmark Definition**

The definition and specification of dependability benchmark prototypes for embedded and transactional applications will bring together for the first time the concepts and techniques developed in previous workpackages. The specification of the prototypes must include all the details required to implement and use the benchmarks for different real systems. We anticipate that these specifications will include the following major issues: i) workload, ii) faultload, iii) measurements, and iv) benchmark conduct.

**Representative workloads** have been selected for each application area. A control and monitoring application has been selected to represent a typical embedded application and the TPC-C benchmark workload from the Transaction Processing Performance Council (TPC) has been selected as a typical transactional application [23]. It is worth noting that these workloads represent well the typical workload nature and the different degrees of criticality that can be found in each application area. In fact, the control and monitoring application comprises both non-critical and mission critical functions, while the TPC-C workload is used to represent a comprehensive range of database applications, including high demanding business-critical database applications. Additionally, the use of a workload from an established performance benchmark (the TPC-C) will provide useful insights from this very successful benchmark.

The **faultload** describes the set of faults that are going to be inserted in the target system. It is defined according to the fault representativeness and fault selection criteria established in WP2. The means (tools and techniques) that must be used to apply the faultload must also be described for a complete benchmark specification.

The **sets of measurements** to be performed on the target system and the corresponding measures are defined from WP1 and WP2. The benchmark results, representing both the end-user and the system developer's perspectives, combine functional (performance) and dependability measures. While for the end-user's perspective the quantitative characterization of the performance of the global system in the presence of faults could be enough, the system developer will be interested in a more detailed characterization of the dependability attributes of the system or system components.

The **benchmark conduct** specifies all the detailed steps required to correctly implement and apply the benchmark. This is very important to assure a uniform (standard) use of the benchmark and guarantee that the benchmark results are meaningful and can be used to compare alternative solutions from the dependability point of view.

### **5.2 Benchmark Experimentation**

The benchmarks defined for each application area will be developed for two different OS families, resulting in a comprehensive set of experiments over COTS-based systems. We will thus develop dependability benchmark for: i) embedded applications running over Windows-E and Linux-E, and for ii) transactional applications based on Oracle running over Windows and Linux.

Naturally, the need for performing these major benchmark experiments provides us with a comprehensive experimental environment that can be exploited in many directions, allowing a complete and detailed evaluation of concepts and techniques. This cross exploitation will be done within the same system family (to emphasize the difference and similarities between the two application areas) and between the two families (Windows and Linux) considered in the project. Following are some examples of the analyses that are allowed:

- Evaluation of dependability features of the target systems, using synthetic workloads. The idea is to benchmark the operating systems “alone” and to evaluate the dependability features of specific system components. The use of this information to accelerate the benchmarking process at the system level will be investigated;
- Comparison between “black-box” and “white-box” benchmarking. Although the needed dependability benchmarks must be designed for systems for which the source code is not available (black-box benchmarking), it is very interesting to investigate specific features dependent on the knowledge of the source code;
- Analysis of the differences and similarities between benchmarking the full versions of the OS and the reduced version designed for embedded world. This is particularly interesting as it is a practical way to isolate the influence of specific components of the OS that are present in one version and absent in the other;
- Analysis of the common dependability characteristics and differences of the two OS families;
- Comparison of the benchmark results obtained for each benchmark (i.e., the benchmark for embedded and the benchmark for transactional applications) running on top of different OSs.

### **5.3 Benchmark Validation**

The most difficult part of the validation is to assure that the benchmark results do represent an accurate characterization of the dependability properties of the tested systems and thus can be used to compare alternative solutions. In fact, this validation is tightly related to the evaluation of aspects such as fault representativeness that will be performed in WP2. Complementary approaches based on theoretical, modeling, and experimental techniques will be used to acquire confidence in the benchmark results.

The benchmark validation also consists of assuring that the benchmark prototypes satisfy a set of properties required to derive practical, useful, and meaningful dependability benchmarks. Examples of such properties are portability, scalability, usability, non-interference, non-damaging, and repeatability for high confidence in the results. The actual implementation of the planned benchmark prototypes in the rich and diverse experimental environment allows for a comprehensive evaluation and validation of these benchmark properties. In addition, the large number of experiments will help validate results repeatability and benchmark usability.

The validation of the benchmark prototypes is a pragmatic way of validating the concept of dependability benchmarking, as defined, specified, and implemented in the DBench project. In fact, the validation of the benchmark prototypes also represents the actual validation of the conceptual framework (WP1) and of the different enabling technologies (WP2) in an actual benchmarking environment.

## **6. WP4: Consolidation**

This Workpackage is the final step of the project, where all advances and developments made in the project are summed up, discussed and put into their final format. It will give the full description of the proposed Framework for Dependability Benchmarking, which will take into account all the methods, insights and tools produced or studied in the project

The Workpackage is expected to clarify the benchmark objective and utilization, as well as the benchmark measures and properties. Significant advances are needed in all enabling technologies (conceptual and experimental foundations, fault representativeness, fault injection, measurements, experiment conduct). These advances are expected to be made possible by the experiments performed

in WP2 and WP3, and by the detailed studies made in WP1. Some final experiments with the existing set-ups might still be performed at this stage to clarify some points that may need it, probably in issues like fault representativeness and measurements.

The practical outcomes of this workpackage are i) a set of guidelines and recommendations for the benchmark users and ii) a set of benchmark prototypes.

Finally, a special effort will be devoted to the dissemination and exploitation of the results during this period. The prototype tools will be widely disseminated (through the web, whenever possible) for the community to become acquainted with the technology, and convinced of its usefulness.

## 7. Relevance and evaluation of results

To ensure that the guidelines and benchmarks developed in DBench are relevant and representative for real applications, the project includes an SME as partner (Critical software) and Microsoft, as sponsor. Further advice and guide to the partners on actual utility and relevance of the project activities to industry, will be ensured by the **Industrial Advisory Board (IAB)**, where several companies are represented (listed in the front page). They represent both COTS providers and COTS end-users.

The Evaluation and Assessment of the project will heavily rely on inputs from the IAB members. In addition, it is worth mentioning the creation (by the IFIP WG10.4) of a Special Interest Group (SIG) on Dependability Benchmarking. All partners of DBench are members of this SIG. Many influential system developers are members of the SIG (AT&T, IBM, Compaq, Sun Microsystems) together with well known-academic institutes. The project will take advantage of the SIG meetings to present the results and solicit feedback. This feedback, discussed within the project, will be a part of the self-evaluation. The chairman of the SIG is in the IAB.

## References

- [1] J. C. Campelo, F. Rodríguez, P. J. Gil and J. J. Serrano, *Design and Validation of a Distributed Industrial Control System nodes*, Technical Univ. of Valencia, Spain, Research Report, 1999.
- [2] J. Carreira, H. Madeira and J. G. Silva, "Xception: A Technique for the Experimental Evaluation of Dependability in Modern Computers", *IEEE Trans. on Software Engineering*, 24 (2), pp.125-136, February 1998.
- [3] C. Constantinescu, "Validation of the Fault/Error Handling Mechanisms of the Teraflops Supercomputer", in *Proc. 28th Int. Symp. on Fault-Tolerant Computing (FTCS-28)*, (Munich, Germany), pp.382-389, IEEE CSP, June 1998.
- [4] Y. Crouzet, P. Thévenod-Fosse and H. Waeselynyck, "Validation of Software Testing by Fault Injection: The SESAME Tool", in *Proc. 11th Conf. on Reliability and Maintainability*, (Arcachon, France), pp.551-559, SEE, September 1998, *In French*.
- [5] M. Daran and P. Thévenod-Fosse, "Software Error Analysis: A Real Case Study Involving Real Faults and Mutations", in *Proc. Int. Symp. on Software Testing and Analysis (ISSTA'96)*, (S. J. Zeil, Ed.), (San Diego, CA, USA), pp.158-171, ACM Press, January 1996.
- [6] J.-C. Fabre, F. Salles, M. Rodríguez Moreno and J. Arlat, "Assessment of COTS Microkernels by Fault Injection", in *Dependable Computing for Critical Applications (Proc. 7th IFIP Working Conf. on Dependable Computing for Critical Applications: DCCA-7, San Jose, CA, USA, January 1999)*, (C. B. Weinstock and J. Rushby, Eds.), Dependable Computing and Fault-Tolerant Systems, 12, (A. Avizienis, H. Kopetz and J.-C. Laprie, Eds.), pp.25-44, IEEE CSP, Los Alamitos, CA, USA, 1999.
- [7] P. Folkesson, S. Svensson and J. Karlsson, "A Comparison of Simulation Based and Scan Chain Implemented Fault Injection", in *Proc. 28th Int. Symp. on Fault-Tolerant Computing (FTCS-28)*, (Munich, Germany), pp.284-293, IEEE CSP, June 1998.
- [8] E. Fuchs, "Validating the Fail-Silence of the MARS Architecture", in *Dependable Computing for Critical Applications (Proc. 6th IFIP Int. Working Conference on Dependable Computing for*

- Critical Applications: DCCA-6, Grainau, Germany*, (M. Dal Cin, C. Meadows and W. H. Sanders, Eds.), Dependable Computing and Fault-Tolerant Systems, 11, (A. Avizienis, H. Kopetz and J.-C. Laprie, Eds.), pp.225-247, IEEE CSP, Los Vaqueros, CA, USA, 1998.
- [9] J. Gray (Ed.), *The Benchmark Handbook*, Morgan Kaufmann Publishers, San Francisco, CA, USA, 1993.
- [10] J. Güthoff and V. Sieh, "Combining Software-Implemented and Simulation-Based Fault Injection into a Single Fault Injection Method", in *Proc. 25th Int. Symp. on Fault-Tolerant Computing (FTCS-25)*, (Pasadena, CA, USA), pp.196-206, IEEE CSP, June 1995.
- [11] I. Jacobson, G. Booch and J. Rumbaugh, *The Unified Software Development Process*, Addison-Wesley, 1999.
- [12] K. Kanoun, M. Kaâniche and J.-C. Laprie, "Qualitative and Quantitative Reliability Assessment", *IEEE Software*, 14 (2), pp.77-86, march 1997.
- [13] K. Kanoun, M. Borrel, T. Morteveille and A. Peytavin "Availability of CAUTRA, a sub-set of the French Air Traffic Control System", *IEEE Transactions on Computers*, Vol. 48, No. 5, pp. 528-535, May 1999.
- [14] K. Kanoun and M. Borrel, "Dependability of Fault-tolerant Systems — Explicit Modeling of Hardware & Software Component-Interactions", *IEEE Transactions on Reliability*, to appear in September issue 2000.
- [15] J. Karlsson, P. Folkesson, J. Arlat, Y. Crouzet and G. Leber, "Integration and Comparison of Three Physical Fault Injection Techniques", in *Predictably Dependable Computing Systems*, (B. Randell, J.-C. Laprie, H. Kopetz and B. Littlewood, Eds.), ESPRIT Basic Research Series, pp.309-327, Springer, Berlin, Germany, 1995.
- [16] H. Madeira, D. Costa and M. Vieira, "On the Emulation of Software Faults by Software Fault Injection", in *Proc. Int. Conference on Dependable Systems and Networks (DSN'2000)*, (New York, NY, USA), IEEE Computer Society Press, pp.417-426, June 2000.
- [17] R. J. Martínez, P. J. Gil, G. Martín, C. Pérez and J. J. Serrano, "Experimental Validation of High-Speed Fault-Tolerant Systems Using Physical Fault Injection", in *Dependable Computing for Critical Applications (Proc. 7th IFIP Working Conf. on Dependable Computing for Critical Applications: DCCA-7, San Jose, CA, USA, January 1999)*, (C. B. Weinstock and J. Rushby, Eds.), Dependable Computing and Fault-Tolerant Systems, 12, (A. Avizienis, H. Kopetz and J.-C. Laprie, Eds.), pp.249-265, IEEE CSP, San Jose, CA, USA, 1999.
- [18] D. Powell, M. Cukier and J. Arlat, "On Stratified Sampling for High Coverage Estimations", in *Proc. 2nd European Dependable Computing Conf. (EDCC-2)*, (Taormina, Italy), pp.37-54, Springer-Verlag, October 1996.
- [19] V. Sieh, O. Tschäche and F. Balbach, "Comparing Different Models Using VERIFY", in *Proc. 6th IFIP Int. Working Conference on Dependable Computing for Critical Applications (DCCA-6)*, (Grainau, Germany), pp.59-76, March 1997.
- [20] P. Sinha and N. Suri, "Identification of Test Cases Using Formal Techniques", in *Proc. 29th Int. Symp. on Fault-Tolerant Computing (FTCS-29)*, (Madison, WI, USA), IEEE CSP, June 1999.
- [21] A. Steininger and H. Schweinzer, "A Model for the Analysis of the Fault Injection Process", in *Proc. 25th Int. Symp. on Fault-Tolerant Computing (FTCS-25)*, (Pasadena, CA, USA), pp.186-195, IEEE Computer Society Press, June 1995.
- [22] N. Suri and P. Sinha, "On the Use of Formal Techniques for Validation", in *Proc. 28th Int. Symp. on Fault-Tolerant Computing (FTCS-28)*, (Munich, Germany,1998), pp.390-399, IEEE CSP, June 1998.
- [23] *TPC Benchmark C, Standard Specification*, Revision 3.3, Transaction Processing Performance Consortium, San José, CA, USA, 1998.
- [24] C. R. Yount and D. P. Siewiorek, "A Methodology for the Rapid Injection of Transient Hardware Errors", *IEEE Trans. on Computers*, 45 (8), pp.881-891, August 1996.