Models, Proofs and the Engineering of Computer-Based Systems: A Reality Check

Gérard Le Lann
INRIA - BP 105 - 78153 Le Chesnay Cedex, France
Internet: Gerard.Le_Lann@inria.fr

Abstract

The need for drastic reductions in budgeting and durations of projects that involve computer-based systems, as well as the increasing pressure to field systems that function satisfactorily, have fostered the need for more rigorous system engineering methods. Via case studies, we report on why models and proofs, which are at the core of emerging methods, are indeed instrumental in slashing costs, by avoiding project setbacks, operational mishaps or failures, and by permitting correct diagnoses when necessary. The perspectives offered by such methods help to understand better what is missing with existing COTS products, as well as what is the borderline between software engineering and system engineering.

INTRODUCTION

As is the case with mature engineering disciplines, the engineering of computer-based systems (CBSs) is bound to be based on models and proofs. Models and proofs are needed to provide ourselves with generic, understandable, repeatable processes. Models and proofs are instrumental in avoiding system engineering faults: A design and/or a physical dimensioning of a CBS can be proven correct, before construction or fielding is undertaken. This results in huge savings in project durations and system development and maintenance costs, as well as in higher levels of confidence into deployed systems, given that, according to recent studies, lack of rigor in SE methods or processes has seemingly become the major source of problems.

First, we briefly introduce those basic features of proof-based system engineering (SE) that are needed to report on case studies. Then, issues raised with COTS technology are put into perspective. Finally, we explore whether blaming "the software" whenever a problem arises is a surrogate for more serious analyses. Could it be that many "S/W problems" and the so-called S/W crisis would be (very much?) due to lack of rigor in SE methods/processes?

WHAT IS PROOF-BASED SE

For a detailed presentation of the basic principles of proof-based SE as well as the real problems that have been addressed over the last 4 years with the TRDF method, we refer the reader to (LL98).

Notation <N> (respectively, [N]) refers to a specification of a problem (resp., a solution). A specification is a self-contained and unambiguous set of statements, expressed in some human language, or via some formalized notation, or in some formal language. A problem, a solution, includes a set of requirements, i.e. properties, and a set of assumptions, i.e. models. Therefore, <N> is pair {<m.N>, <p.N>}, where m stands for models and p stands for properties. A specification usually contains variables, some of which are assigned a (range of) numerical value(s), while others are left unvalued. Notation <n> (resp., [n]) refers to the specification that contains those variables of <N> (resp., [N]) that are left unvalued.

The essential goal of proof-based SE is as follows: starting from AP, some initial and informal description of end users/customers application requirements and assumptions (e.g., an invitation-to-tender), to produce <AP>, a specification of the application problem under consideration, <X>, a specification of the computer science problem that matches <AP>, [S], a global, modular specification of a CBS, along with the proofs that [S] does solve problem <X>.

Proof-based SE builds upon Computer Science and model-based SE - see, e.g., (L97), (SRMB98), (SK97). Examples of models and properties, with well defined semantics, that are commonly considered with CBSs - some being used in the sequel - are given in the Appendix. Note that many of the models involved with the engineering of CBSs are not usually considered in S/W Engineering.

Phases of a project life-cycle which fall within the scope of proof-based SE, and which precede phases covered by other Engineering disciplines (e.g., S/W Engineering), are now surveyed and illustrated.

CAPTURE OF AN APPLICATION PROBLEM

AP ➡ <AP> ➡ <X>

A capture phase serves to translate AP into <AP> and <X>. This is done by resorting to properties (resp., models) in order to specify what is commonly called requirements (resp., environmental assumptions).

An example in Modular Avionics

Below is an excerpt from specifications we have established for and with Dassault Aviation, within the framework of a DGA/DSP (French DARPA) project, related to the NATO Allied Standard Avionics Architecture Council program.

AP: Set of functions. Pilot functions U, Y and Z are critical. Environment is aggressive (details given).

<p.AP>: At all times, functions U, Y and Z must be executable and must terminate within prescribed time bounds. Time bound for function Y is known. Time bounds for other functions: TBD (to-be-defined).

<m.AP>: Processors can fail (number of failures without repair is TBD). External events (via sensors) that trigger function Y or Z occur according to unknown laws (TBD). External events that trigger function U are sampled periodically, period is known.

<p.X>: Timeliness property required for U, Y and Z is termination within a strict deadline. Respective deadlines, relative to (unknown) times of request for activation, are d_U , 1 ms, d_Z. Absolute availability is the dependability property required for U, Y and Z.

<m.X>: Up to f processors can fail, according to the stop failure model. Model for external events that trigger function Y (resp. Z) is the unimodal arbitrary arrival model, with parameters $\{w, arr, sp\}_Y$ (resp. $\{w, arr, sp\}_Z$). Model for external events that trigger function U is the periodic model, period = 20 ms.

<p.x>: d_U , $d_{Z,}$
<m.x>: f, $\{w, arr, sp\}_Y$, $\{w, arr, sp\}_Z$.

Note: It was also required that there should be no restriction w.r.t. programming models or CBS configurability, which raises serializability issues.

An example with the Ariane program

This example is related to the European Ariane program (satellite launchers). Some analyses of the failure of Ariane 5 flight 501 (ESA96) conclude that the failure has been caused by S/W Engineering errors, in line with the Inquiry Board findings. Other analyses, including ours (LL96), (LL97), establish that the failure was caused by SE faults. See (RISKS) and (SCS) for contributions and discussions on this topic, as well as the subsection on "The failure of Ariane 4.5". In this paper, we give a detailed, albeit incomplete, presentation of what could have (and what has) been done. A correct capture would have led to the following specifications:

<m.X> (excerpt):
◆ EXT = set of external variables, i.e. variables shared between the CBS and other launcher subsystems. (For example, launcher horizontal velocity HV is in EXT).
◆ For every variable var in EXT, R(var) is the range of possible values.
◆ For every operational mode, which set of functions need be invoked. (For instance, functions needed/not needed after lift-off are listed).
◆ Every function, considered in isolation from the others, satisfies global invariants I (consistency constraints) defined over values of mission-dependent variables and variables in EXT. (For instance, values taken by variable "nominal trajectory" and by HV are not independent).
◆ Processors can fail according to the stop model.

<p.X> (excerpt):
◆ Serializability: Any concurrent activation of any number of functions should satisfy invariants I.
◆ Timeliness: For every event ev that updates a variable in EXT, latency between occurrence of ev (sensors reading) and elaboration of related outputs (actuators command) is upperly bounded.
◆ Availability/reliability of the CBS must be higher than $1 - 10^{-\alpha}$, $\alpha > 0$ known.

<m.x> (excerpt):
◆ R(HV).
◆ Boolean "alignment function" AF. AF = true means that alignment of the inertial platform must be operative after lift-off (false otherwise).
◆ Set of invariants I.

<p.x> (excerpt):
◆ Deadlines $d(ev_1)$, $d(ev_2)$,...

A capture phase involves iterative customer/provider interactions. It is efficient to proceed by identifying generic problems and subproblems, and by reusing corresponding generic specifications (J_REQ). Set {<AP>, <ap>, <X>, <x>} is the technical contract established between a customer and a CBS provider.

SYSTEM DESIGN

$<X> \Rightarrow \{[S], [DTool]\}$

This phase, which is under the responsibility of the CBS provider, has <X> as an input, and covers all the design stages needed to arrive at [S], a modular and partially valued specification of a CBS, the completion of each design stage being conditioned on fulfilling design correctness proof obligations (LL98). Proofs are established by resorting to models such as those shown in the Appendix, to algorithms, protocols, and appropriate reasoning techniques - see, e.g., (L96) - or such disciplines as Game, Decision or Scheduling theories.

A system design phase has a tree structure. At every node, starting from root problem <X>, rewritten $<X_1>$, the problem under consideration is decomposed into independent subproblems. The design stage corresponding to tree node problem r of depth k - denoted $<X_{k,r}>$ - is completed when:
- a global design (i.e. a solution) has been specified for the modular decomposition considered,
- proofs are given that this design endows the modular decomposition with global properties that are at least those specified in $<p.X_{k,r}>$, for assumptions that are at least those specified in $<m.X_{k,r}>$.

Let $[S_i]$ be leaf i of a z-leaf design tree. Specification $[s_i]$ contains those unvalued variables that appear in $[S_i]$. Among variables found in [s], the union of the $[s_i]$'s, one finds variables whose values directly depend on values assigned to variables found in <x>. For example, in the Ariane case, [s] should contain Q(BH), the buffering size needed to store the values of horizontal bias BH, which depends on R(HV). Other variables found in [s] are those that appear in the design proofs, such as timers, processors speeds, number of processors, sizes of waiting queues.

One stops designing along a branch whenever the specification $[S_i]$ arrived at is known to be implementable (e.g., via some cost-effective COTS technology, or as a customized H/W and/or S/W module - see the "COTS products issues" subsection). By the virtue of the uninterrupted tree of proofs that every design is correct, [S], the union of design tree leaves $[S_i]$, $i \in [1, z]$, provably solves initial problem <X>.

Proof-based SE eliminates the staggering complexity that inevitably plagues system verification and integration testing under current SE practices. Proofs entail establishing behavioral functions such as upper bounds on response times (B), lower bounds on redundancy degrees, and so on. Such functions serve to establish feasibility conditions (FCs) - a set of analytical constraints - under which it is guaranteed that properties <p.X> hold true, provided that the models are not violated. FCs delineate a set of scenarios that necessarily contains every possible worst-case scenario that can be deployed by "adversary" <m.X>. Establishing FCs does away with the need to explore exhaustively (if at all feasible) all possible future system states. This is how one eliminates the "state space explosion" problem commonly encountered with existing rigorous or formal S/W Engineering methods. FCs serve to specify a Dimensioning Tool for [S], noted DTool. In contrast with event-driven simulation, whereby expected values and standard deviations are estimated over large samples of simulated runs, DTool checks worst-case scenarios deterministically, and computes guaranteed bounds (such as B).

An example in Modular Avionics

The solution that we have specified and that has been implemented by Dassault Aviation (CHLL99) is based on a distributed client-server architecture, under the control of a combination of real-time scheduling, fault detection-and-recovery, and distributed consensus algorithms. We had to prove the worst-case scenarios ("concentration" of event arrivals and processor failures). That was the most difficult part. Contrary to "traditional" scheduling (e.g., the Rate-Monotonic or the Time-Triggered approaches), tasks are modeled as graphs (i.e. with internal dependencies) and the event arrival model that was selected by Dassault Aviation is the unimodal arbitrary model. The high degree of genericity/portability embodied within such models complicates the proof exercise. For instance, traditional schedulability reasoning falls apart, partly because there is no commutativity property with graphs (w.r.t. processor busy periods).

Having established functions B_U, B_Y and B_Z, those FCs under which deadlines d_U, d_Y and d_Z are always met are simply obtained by writing the following:

$$B_U < d_U \qquad B_Y < 1 \text{ ms} \qquad B_Z < d_Z .$$

An example with Ariane 5/flight 501

One of the design faults is as follows. The prime contractor has split initial problem $<X_1>$ into a number of subproblems, considering a modular decomposition that included, among others, the Inertial Reference System (SRI) module (problem $<X_{2,sri}>$) and the On-Board Computer (OBC) module (problem $<X_{2,obc}>$). Analysis of design stages 1 and 2 reveals that the processor stop failure model that had to appear in <m.$X_{2,sri}$> has been violated during flight 501. Rather than "committing suicide" when they diagnosed an error (value of BH higher than implicitly assumed), SRI processors sent an error message to OBC processors, which was interpreted as correct flight data, which led to the failure.

We will not discuss here whether specification <m.$X_{2,sri}$> did state or, respectively, did not state the stop failure model. In other words, we will not discuss whether that design fault has been made by the contractor in charge of the SRI module or, respectively, by the prime contractor.

Design correctness proofs would have revealed the inconsistency, which would have led someone to question the underlying assumption that there is no common mode failure (of the SRI module).

COTS products issues

A design phase can also be conducted bottom-up, the case whenever COTS or customized products must be reused, such as, e.g., those which minimize cost and risk criteria. Knowing ahead of design time that specific products must be part of a CBS is tantamount to "freezing" some of the design tree leaves a priori. Correctness proof obligations still apply. FCs are needed as well (see Mars PathFinder mishaps).

Imagine that every COTS product of interest would be accompanied with a specification [S_i], along with FCs, or with the specification of which problem <X_i> this product solves. It would then be trivial to decide whether a COTS product can be part of a global design solution. Why should the Systems Engineering community keep bearing the burden of "looking inside" COTS boxes, of exercising them, and the like, in order to infer such specifications? This work is under the responsibility of COTS technology vendors.

Specifications akin to [S_i] , FCs or <X_i> are available with refrigerators. Why not with real-time monitors or middleware? Success with projects involving complex and/or critical CBSs partially or fully built out of COTS products will keep resting on heroic efforts and luck - not a very exciting perspective - until the Systems Engineering community makes it clear to the COTS vendors that they must change their habits and make contractual commitments on such specifications as {[S_i], [s_i]}, or FCs, or {<X_i>, <x_i>}, correctness proofs being disclosed or not, which has significant legal and commercial implications.

SYSTEM DIMENSIONING

Before an implementation can be undertaken, or finalized in cases where existing modules are reused, the physical dimensioning of every module must be specified. This entails finding a correct valuation V([s]) for those variables that appear in [s], the specification that contains those variables of [S] left unvalued. Of course, V([s]) must match V(<x>), a customer provided valuation of problem <X>. Many different quantifications V(<x>) may be contemplated by a customer or a prime contractor, for various CBS releases, without invalidating design work.

Finding a correct dimensioning of a CBS is conditioned on fulfilling correctness proof obligations, which are enforced by DTool, i.e. by checking the FCs established during a design phase. V(<x>) is an input to DTool; V([s]) is an output from DTool.

An example in Modular Avionics

In our work, [s] contains such variables as, e.g., t, the invocation period of the distributed scheduling/consensus algorithm, speed(k), speed of processor k, Q(k), the size of the task scheduler waiting queue local to processor k. V(<p.x>) - resp. V(<m.x>) - serves to assign numerical values to d_U and d_Z - resp. to f, $\{w, arr, sp\}_Y$, and $\{w, arr, sp\}_Z$.

By iterating on values of t, of speed(k), and by computing the FCs, DTool either returns a negative answer - meaning V(<x>) is not feasible (given design [S]) - or declares feasibility, in which case values of t and speed(k) found to be sufficient are returned, as well as the highest value attained for each of the Q(k)'s, i.e. the memory capacity needed to avoid an overflow of waiting queues. In case different assignments V([s]) meet a given V(<x>), that V([s]) which minimizes some criterion (e.g., cost) is selected.

Checking whether FCs are met rather than proceeding with statistical analysis, as is the case with event-driven simulation, has an impact on running times. Performance speed-up ratio of DTool compared to an event-driven simulator developed by Dassault Aviation is in the order of 25.

An example with the Ariane program

Assuming the capture phase would have been conducted as described previously, <u>and assuming that the Ariane 4 CBS design and software are reused untouched for Ariane 5</u>, the outputs of the respective dimensioning phases would have been as shown below.

Recall that the dimensioning of unsigned integer BH, noted Q(BH), is a known (design-dependent) function F of the dimensioning of HV, which dimensioning is fully and exclusively determined by those space engineers who had to decide on a particular launcher technology for Ariane 4 first, and then for Ariane 5 (not a S/W issue). It turns out that Ariane 5's HV can be as much as 5 times higher than Ariane 4's HV.

◆ **Ariane 4 (A4)**
V(<x>): AF = true, R(HV) = HV_4.
V([s]): Q(BH_4) = F(HV_4) = 15 bits.

◆ **Ariane 5 (A5)**
V(<x>): AF = false, R(HV) = $HV_5 \leq 5\ HV_4$.
(As a consequence, $BH_5 \leq 5\ BH_4$).
V([s]): Q(BH_5) = F(HV_5) = 18 bits.

The 501 failure entailed the destruction of the launcher and its satellites, a loss in the order of US$ 700 Million, setting to approximately Euro 210 Million the cost of each missing bit.

An example with Mars PathFinder
System shutdowns that were observed during testing at NASA/JPL were attributed to obscure H/W glitches, whose occurrence during mission was declared very much unlikely. That guess was wrong. Neither Wind River nor JPL felt it necessary to establish VxWorks-dependent FCs. Consequently, no dimensioning oracle (DTool) was built. Such FCs can be established - assuming a capture phase has been conducted correctly, given that VxWorks is a real-time monitor which schedules tasks according to a well known fixed priority-based algorithm.

Had DTool been built and run before launching the probe, JPL would have discovered that the 125 ms bound specified (via V(<p.x>)) for the cycle time of task bc_dist was violated - violation of this timeliness property led to the system shutdowns - with option PI (priority inheritance) turned off.

They could then have found that DTool returns a positive response when PI is turned on. PI had been arbitrarily set to off - without telling the customer - because Wind River engineers know that this shortens VxWorks execution time.

PI is an example of a CBS variable that should have been valued via V([s]). This is an illustration of what must change, in terms of contractual commitments, with COTS products.

SUMMARY

{[S], V([s])} is a modular and fully quantified specification of a CBS that provably solves quantified problem {<X>, V(<x>)}. Each of these specification modules can then be implemented as desired, in H/W, in S/W, or in any appropriate technology. In case some existing module is reused, its specification must be checked against the modules of {[S], V([s])}, which are technical contracts established between a prime contractor and co/subcontractors.

That SE issues must be addressed prior to addressing S/W Engineering issues is not acknowledged in certain circles. Yet, it does not help much to check or to prove S/W design or S/W implementation correctness for specifications that are incomplete or incorrect in the first place. Similarly, oversight of SE issues, or confusion between SE issues and S/W Engineering issues usually leads to projects setbacks, or to operational mishaps and/or failures, or to erroneous a posteriori diagnoses.

Many analyses and audits, in various areas, demonstrate that problems often are mistakenly believed to originate in S/W. Besides the Ariane 5 and the Mars PathFinder examples, this has been shown to be the case with, e.g., Therac-25 (LT93), the FAA AAS project (air traffic control), the US PSTN (K97), the Danone vs. C case (see further).

A PROBLEM? IT'S THE SOFTWARE!

The failure of Ariane 4.5
An Inquiry Board was formed to identify the cause(s) of the 501 failure. Their report concludes that causes are S/W design and S/W implementation errors (ESA96), a view which is disputed - see (L98) for example. In fact, it is almost straightforward to show that the unique cause of the failure is a SE fault.

The origin of the causal graph that leads to the 501 failure is a fault rooted in the launcher requirements capture phase. The alignment task was running, despite the fact that realignment of the inertial platform after lift-off, needed with A4, is useless in the case of A5. This task contains the conversion procedure that computes integer BH from HV.

What if someone had had the idea of disallowing the execution of this task after lift-off? The BH overflow could not have occurred, and the scenario which has led to the 501 failure could not have occurred either.

Now the argument. How could this someone know that this was the right thing to do? Obviously, only by correctly capturing the problem to be solved by the A5 CBS, i.e. by specifying the interface between this particular A5 subsystem and the A5 inertial platform subsystem - a SE issue (see above). The only way to do this is by asking an Ariane engineer: “Given A5 technology, do you need to have the strap-down inertial platform aligned after lift-off?”. Only then would have it been known that boolean AF had to be set to false. That knowledge, which relates to a specific launcher technology, is totally independent of the fact that the thing that, after lift-off, happens to be needed (A4), or not needed (A5), is implemented in hardware, in software, or in melloware, correctly or incorrectly. It has never been the intent of ESA, of CNES, or Arianespace, to embark on building or operating a launcher whose technology is A5’s technology and which needs inertial platform alignment after lift-off, a fictitious launcher that could be labelled Ariane 4.5, half-way between A4 and A5. End of the argument.

Therefore, stricto sensu, neither the work invested in “inspecting the code” and ironing out the “S/W errors” from the alignment task, nor the contributions (including ours) to the “Is the 501 failure due to S/W or System Engineering mistakes?” debate, apply to the Ariane 5 program. They are relevant for this fictitious Ariane 4.5 launcher, that will never be operated, and whose unique flight is labelled 501. The real qualification flights of A5 have been (successful) flights 502 and 503, which were conducted with the alignment task inhibited after lift-off.

And there is no “S/W error” involved with the design or the implementation of the alignment task, anyway. Calling a faulty dimensioning of a memory buffer a “S/W error” is as misleading as calling the choice of too slow a processor a “S/W error”. What if the procedure which correctly computes values of BH had been correctly implemented in H/W? Flight 501 would have failed as well - unless Q(BH) would have been correctly dimensioned. However, the Inquiry Board would have then diagnosed a “H/W design error”, an equally superficial conclusion.

Expertise required to be a “good” system researcher/ engineer is not the expertise required to be a “good” S/W researcher/engineer (and vice-versa). There is a particularly striking illustration of the above in the Board report. The Board found that "there is considerable redundancy at the equipment level" (p. 3 of the report). The Ariane CBS implements simple non diversified replication (2 identical SRI processors, 2 identical OBC processors). And this is the weakest kind of redundancy that can be imagined.

The Mars PathFinder shutdowns

Luck! Some JPL engineers had time to engage in heroic “code inspection”, patch the value of PI, to see that this would solve the shutdown problem. Hence the illusion that it was a S/W problem!

The VxWorks code is not flawed (at least for what is of concern here). It was run under some inappropriate (1 out of 2) option, a SE issue. It’s like driving your car in first gear all the time, to find out that your car is too slow in some occasions, and complain that “the engine has a problem”. Imagine that VxWorks would have been ASIC-implemented. The JPL engineers would have done “H/W inspection”. Then the conclusion would have been: It’s a H/W problem...

Overcoming the limitations of formal S/W Engineering methods

Models and properties that are tractable with proof-based SE methods match reality (e.g., the multimodal arbitrary arrival model, concurrent/distributed computations in the presence of shared persistent updatable variables). Some are even “worse than” reality (e.g., byzantine processors, asynchronous computational models). Models and properties that are tractable with formal S/W Engineering methods do not match reality very well (e.g., the sequential processor model vs. look-ahead techniques, the zero-time execution/infinite storage capacity assumptions).

Proof-based SE serves the purpose of capturing and reasoning about the “ugly” reality (technology, complexity, environment). In particular, design work is conducted until one shows how these “ideal” models currently accommodated with formal or rigorous S/W Engineering methods are indeed supported. Then, it is possible to reap the benefits that result from using such methods.

This has been our experience with the Modular Avionics project. The Dassault Aviation team had developed S/W components in Esterel, a programming language that permits to develop S/W correctness proofs for “simple” models. But how to prove that an entire embedded system is correct w.r.t. the specification of interest (see capture phase)? To make matters worse, “practical” constraints also have to be accounted for (COTS, the “cheaper, faster” motto, the reusability requirement, etc.). They acknowledged they had hit a wall: Esterel arsenal does not help in developing correctness proofs, once for ever, for any possible combination of S/W components drawn from a given set, which share persistent updatable variables, in the presence of concurrency and partial failures. Even less to develop timeliness proofs. Even less to specify correct system dimensionings. After 1 year of SE work and 1 year of implementation and experimentation, the Dassault team could see that proof-based SE (the TRDF method) had permitted to “bridge the gap”. In particular, with the DTool we have specified, they can dimension their CBS at will, for any possible assignment of any subset of tasks (modeled as directed finite graphs of Esterel components), of shared variables, over the processors.

This demonstrates that there are limitations to what can be achieved with S/W Engineering, that many of these limitations correspond to SE issues, and that is useless to address them as S/W Engineering issues.

A costly confusion
In the mid 80’s, a company - say C - was awarded a contract by Danone for delivering a CBS intended for operating an automated plant (dairy products). Acceptance tests demonstrated that the CBS was not functioning properly. This was acknowledged by C, which argued that problems were due to errors in the application S/W. Danone granted an extension of the contract. Months later, the CBS failed the acceptance tests again. Danone refused to take delivery of the CBS, and did not pay the final (big) check. This turned into a lawsuit, which lasted 2.5 years. <u>Official diagnosis and legal battle: Final CBS still plagued with application-level S/W errors</u>.

An analysis (a few days) of the application problem revealed that the CBS had to be designed as a distributed system, so as to meet extensibility and dependability requirements, which raises well known synchronization problems such as, e.g., distributed global snapshots, distributed mutual exclusion. These problems have algorithmic solutions documented in the open literature (some algorithms are even implemented in some COTS products). <u>Real cause: Lack of a distributed synchronization algorithm</u>.

An analysis revealed that the CBS was not equipped with a synchronization algorithm, a major SE mistake (C was unaware of the issue!). Hence, application programs were run interleaved in arbitrary manner. As is well known, even with perfect (bug-free) S/W, that CBS could not behave correctly anyway.

Lessons learned are:
(1) spending time and budgets for fixing S/W errors makes no sense when issues at stake are SE issues,
(2) the 2.5 year long litigation that focused on S/W issues was a waste of time and money.

The US Public Switched Telephone Network
A study conducted by NIST (K97) on those outages experienced by the US PSTN between 1992 and 1994 has led to interesting conclusions, which contradict widely held beliefs. Excerpts:

“An unexpected finding, given the complexity of the PSTN and its heavy reliance on S/W, was that S/W errors caused less system downtime (2%) than any other source of failure except vandalism”.
“S/W is not the weak link in the PSTN system’s dependability”.
"Overloads caused nearly half of all downtime (44%) in terms of outage minutes”.

Hence, the dominant cause of outages are SE faults. Overloads result from incorrect capture of environmental assumptions <m.X>, faulty designs, incomplete designs (such as lack of FCs), faulty dimensionings, even when S/W implementations are correct.

CONCLUSIONS

It is difficult to imagine how the engineering of CBSs could become a mature discipline without resting on rigor and scientific achievements. Via the investigation of real cases, we have tried to show the relevance of models and proofs. There are obvious technical advantages that result from embracing proof-based SE. Yet, we believe that widespread adoption of proof-based SE will occur when our community gets a correct picture of the significant economic and strategic advantages at stake.

REFERENCES

(CHLL99) P. Carrère, J.-F. Hermant, G. Le Lann, “In Pursuit of Correct Paradigms for Object-Oriented Real-Time Distributed Systems”, 2nd IEEE ISORC Symposium, May 1999, 9 p.

(ESA96) European Space Agency, "Ariane 5 - Flight 501 Failure", Board of Inquiry Report, 19 July 1996, 18 p. [http://www.esrin.esa.it/htdocs/tidc/Press/Press96/ariane5rep.html].

(J_REQ) The Requirements Engineering Journal, Springer-Verlag UK, ISSN 0947-3602.

(K97) D.R. Kuhn, “Sources of Failure in the Public Switched Telephone Network”, IEEE Computer, April 1997, 31-36.

(L98) P. Ladkin, “The Ariane 5 Accident: A Programming Problem?”, Article RVS-J-98-02, Bielefeld University, Germany, March 1998 [http://www.rvs.uni-bielefeld.de].

(LL96) G. Le Lann, “The Ariane 5 Flight 501 Failure - A Case Study in System Engineering for Computing Systems”, INRIA Res. Report 3079, Dec. 1996, 26 p [http://www.inria.fr/RRRT/publications-fra.html].

(LL97) G. Le Lann, “An Analysis of the Ariane 5 Flight 501 Failure - A System Engineering Perspective”, 10th IEEE Intl. ECBS Conference, March 1997, 339-346.

(LL98) G. Le Lann, “Proof-Based System Engineering and Embedded Systems”, invited paper, Lecture Notes in Computer Science 1494, G. Rozenberg, F. Vaandrager Eds., Springer-Verlag Pub., Oct. 1998, 208-248.

(LT93) N.G. Leveson, C. Turner, “An investigation of the Therac-25 Accidents”, IEEE Computer, July 1993, 18-41.

(L97) H. Lykins, “A Framework for Research into Model-Driven System Design”, Proc. of the 7th Annual INCOSE Symposium, L. Hritz, E. Barker, Eds., Aug. 1997, 765-772.

(L96) N. Lynch, "*Distributed Algorithms*", Morgan Kaufmann Pub., ISBN 1-55860-348-4, 1996, 872 p.

(RISKS) The RISKS Forum [http://catless.ncl.ac.uk/Risks].

(SCS) Safety Critical Systems Mailing List [ftp.cs.york.ac.uk, directory hise_reports/sc.list].

(SRMB98) S. Schulz, J. Rozenblit, M. Mrva, K. Buchenrieder, “Model-Based Codesign”, IEEE Computer, Aug. 1998, 60-67.

(SK97) J. Sztipanovits, G. Karsai, “Model-Integrated Computing”, IEEE Computer, April 1997, 110-111.

APPENDIX

Examples of models

● Computational/system models. Used to specify knowledge relative to computing/communication delays. From synchronous (upper bounds exist, values are known), to asynchronous models (delays cannot be bounded).
● Data models. Used to specify external variables, i.e. those shared between a CBS module and its environment. Used to specify invariants to be satisfied by values taken by external variables and application-dependent variables.
● Task models. From finite directed graphs (nodes are sequential subtasks, edges represent dependencies between subtasks), to trees, and so on, to sequences. For each task, events or conditions under which it can/should be run/suspended/aborted.
● Event arrival models (for every event). Periodic, sporadic, aperiodic, unimodal or multimodal arbitrary. The unimodal arbitrary model serves to specify a bounded arrival density, via a time window of size w, an upper bound arr on the number of arrivals contained within any such window, and a minimum time separation sp between any two consecutive arrivals.
● Failure models. They apply to modules. The weakest model is the stop model (either behavior is correct or a permanent crash has occurred). The strongest model is the byzantine model (behavior is arbitrary).

Examples of properties

● Safety. Every possible system run (collective execution of tasks) satisfies some given invariants (e.g., mutual exclusion, serializability, data consistency).
● Liveness. Tasks make progress despite concurrency (e.g., no deadlocks).
●Timeliness. Timeliness constraints are defined for every task. Type: latest termination deadline, bounded termination jitter, earliest start time. Nature: constant, linear, non-linear, functions of system or environment parameters.
● Dependability: availability, reliability, security, confidentiality, are examples of classical measures.

AUTHOR BIOGRAPHY

G. Le Lann is director of INRIA’s Project REFLECS, Rocquencourt (F). He holds a PhD in Computer Science (Univ. of Rennes, F) and an Engineer Degree in Informatics (ENSEIHT, Toulouse, F). He has previously worked at CERN (Geneva, CH), University of Rennes (F), and Stanford University (USA). He also is and has been acting as a consultant for national and international companies and agencies.