Some Practical Issues in Statistically Evaluating Critical Software

P. B. Ladkin

University of Bielefeld and Causalis Limited, ladkin@rvs.uni-bielefeld.de

This paper was presented at the IET System Safety and Cyber Security 2015 conference, 20-22 October, Bristol, UK and appears in the Proceedings

Abstract

In 2010, the author was approached with a query from industry concerning the application of IEC 61508-7:2010 Annex D, on the statistical evaluation of software. We realised that Annex D is not a helpful guide for a number of reasons. We discuss some common assessment scenarios and their quandaries and requirements for the application of statistical methods based on Bernoulli/Poisson mathematics.

1 Introduction

IEC 61508-7:2010 Annex D (henceforth Annex D) gives what it calls initial guidelines on a statistical approach to determining software safety integrity for pre-developed software based on operational experience. The information included is much less than basic course material [13], but is in our experience insufficient to enable nonexperts to apply the methods successfully. Also, there may be more to determining software safety integrity in an IEC 61508-conformant system development than purely operational reliability, which is the characteristic assessed by the statistical methods considered.

There is consensus that statistical evaluation of operational reliability may be used for software which performs functions whose failures over time can be construed as a Bernoulli process or a Poisson process (renewal process). The mathematics of these processes may be found in standard texts such as [13]. We restrict our comments here to these, but note that expert colleagues have successfully applied extensions of these methods in an industrial context [1, 9].

To evaluate software using Bernoulli- or Poisson-process mathematics, it must be established that

- the failures during operation of the software constitute a Bernoulli process, respectively a Poisson process
- the conditions for construing the failures during execution of the software as such a process are rigorously fulfilled
- There is sufficient operational history to enable the statistical evaluation

These conditions result in documentation requirements additional to the operational history alone. Any safety case including such an evaluation should therefore include documentation establishing the validity of the conditions.

IEC 61508:2010 assesses the reliability of safety-function execution in terms of probabilities of failure per hour, pfh, as the appropriate measure for software with continuous function. For software which operates on demand, the appropriate reliability measure is probability of failure per demand, pfd.

Statistical evaluation does not provide certainty about a software property C, e.g., its failure rate (for continuously operating systems), or its probability of failure on demand (for on-demand systems). It provides a likelihood that the property pertains, stated as a level of confidence or confidence level. Typical levels of confidence are 95% and 99%, which represent an assessment that there is a more than 19-in-20, respectively 99-in-100 chance that C pertains, and equally a less than 1-in-20, respectively 1-in-100, chance that C does not pertain. If there is a (derived) safety requirement that C shall pertain, then statistical evaluation alone is insufficient, since certainty is not achieved. Statistical evaluation could also be used to increase confidence in failure-free operation of software over an anticipated operational lifetime, but it is hard to do so [8, 2].

Consider the number of tests N required to draw a conclusion, at a reasonable confidence level Y, say 95% or 99%, that, on the given distribution of inputs, the software is reliable to a pfh of 10^{-x} or above. N is many multiples of 10^x hours; and mutatis mutandis for on-demand functions. See Tables 1 and 3. Tables 2 and 4 lay the numbers out explicitly for the SIL requirements on safety functions[1]

[1] It should be noted that IEC 61508 makes no explicit numerical requirements of this sort for software. The numerical reliability requirements are for safety functions, which cannot be implemented without some hardware, and they concern so-called random failures. Software failures are deemed by IEC 61508 to be systematic rather than random. We do not necessarily support such a distinction but do not discuss it further here.

Acceptable probability of failure to perform design function on command	$< 10^{-x}$
Number of observed demands without failure for a confidence level of 95%	3×10^{x}
Number of observed demands without failure for a confidence level of 99%	4.6×10^{x}

Table 1: Operational-history requirement for on-demand functions

SIL	Acceptable probability of failure	Observed demands for CL 95%	Observed demands for CL 99%
SIL 1	$< 10^{-1}$	3×10^{1}	4.6×10^{1}
SIL 2	$< 10^{-2}$	3×10^{2}	4.6×10^{2}
SIL 1	$< 10^{-3}$	3×10^{3}	4.6×10^{3}
SIL 1	$< 10^{-4}$	3×10^{4}	4.6×10^{4}

Table 2: Operational-history requirement for on-demand functions correlated with IEC 61508 SIL requirements

Acceptable probability of failure to perform design function per hour of operation	$< 10^{-x}$
Number of observed hours of operation without failure for a confidence level of 95%	3×10^{x}
Number of observed hours of operation without failure for a confidence level of 99%	4.6×10^{x}

Table 3: Operational-history requirement for continuous functions

SIL	Acceptable probability of failure per hour	Observed hours for CL 95%	Observed hours for CL 99%
SIL 1	$< 10^{-5}$	3×10^{5}	4.6×10^{5}
SIL 2	$< 10^{-6}$	3×10^{6}	4.6×10^{6}
SIL 3	$< 10^{-7}$	3×10^{7}	4.6×10^{7}
SIL 4	$< 10^{-8}$	3×10^{8}	4.6×10^{8}

Table 4: Operational-history requirement for continuous functions correlated with IEC 61508 SIL requirements

2 Some Difficulties

Statistical evaluation of very-high-reliability software-based functions remains an art at the time of writing. In order for the conclusions of a statistical evaluation to be valid, not only must the conditions under which the mathematics of Bernoulli, resp. Poisson processes is valid rigorously pertain, but there must be very high assurance in the absence of confounding factors. We give two illustrative examples from amongst many.

First example. One requirement is that the distribution of inputs in the intended future use of the software must be identical to the distribution of inputs in the historical operational records.

Some software used in critical applications has a debug or maintenance mode (DMM) which allows a user access to internal data structures in the software. Giving the software input while in DMM results in output of interest to the maintainer, which will rarely be values appropriate for the critical function of the software in other words, this critical function will routinely fail when the software is in DMM. The software is switched into DMM by a specific combination of input values known to the developers/maintainers (henceforth maintainer), but not necessarily by the engineer wishing to use the software in a critical application and evaluating its use statistically (henceforth client)[2] [3].

Operational history provided by maintainer to client will usually not contain examples of operation in DMM. Suppose that the client can ensure exactly this distribution of inputs in hisher application, with the exception of an infrequent but not rare input which is exactly the combination to send the software into DMM. The software will fail, infrequently but not rarely, in the new application. The statistical evaluation will not have been an accurate guide to the future behavior[3]. However, the input distribution will have been very similar to the historical distribution provided for evaluation indeed identical in all but one point! It follows that the requirement that the distribution of past inputs in the operational history should be identical to the distribution of inputs in the future intended use must be taken rigorously.

A second example. Software for a moderately critical application was being assessed for environmental dependencies. The developer assured the assessors that the software was not at all dependent on GPS signals: it had no function that would require location information; no such dependency had been deliberately implemented; indeed, an attempt had been made explicitly to avoid it. The software did not use library or other external functions that were known to rely on GPS. The assessors brought in a GPS jammer and activated it. The software soon ceased to operate as intended because of the jamming [15, 12].

These two examples illustrate how dealing with the rigor of the mathematical conditions for evaluation and careful analysis to determine the absence of confounding factors are required

[2]How to deal effectively with the evaluation and operation of critical software with DMM is beyond the scope of this note. It will necessarily involve both safety and security considerations. See [5].

[3]Note that IEC 61508-1:2010 Clause 7.4.2.3 requires security to be considered [4]. If the information on DMM is available to a malicious agent with access, say an insider with malicious intent, then it is easy to see how such an agent could cause almost-continual loss of intended function, known in computer-security terminology as denial of service (DoS). It is beyond the scope of this note to consider such security issues.

skills for accurately evaluating software with statistical methods.

A common assessment scenario is that Client A proposes to use a real-time version of an operating system, RTOS, to run critical software executing, say, a SIL 3 safety function F, for which the probability of failure per hour should be less than 10^{-7}. A claims that RTOS has more than enough hours without failure to satisfy the SIL 3 condition for F. In particular, F is continuous and A has detailed logs of the order of 10^8 failure-free (for F) operating hours on the software, more than required by Table 2 above.

The validation proposed conceives of failures in RTOS operation as a Poisson process. The *time to next failure* is a random variable which is exponentially distributed (that is, its probability distribution is the exponential distribution [13][Chapter 13, Section 2]. This satisfies the property that

For any t and s,
Prob(F fails in time interval (t, t+m) given that
it hasn't failed up to t)
=
Prob(F fails in time interval (s, s+m) given that
it hasn't failed up to s)[4]

(This property has a name - the exponential distribution is said to be "memoryless" [11, 13]. In fact, the exponential is the only continuous distribution which is memoryless in this sense, *op. cit.*) Suppose RTOS has a failure sequence FSeq, and that T is a time at which RTOS is well within FSeq, within a few microseconds, say s, of inevitable failure. Then

Prob(F fails in time interval (T, T+s) given that
it hash't failed up to T) = 1

Consider now the operation of RTOS within the same few microseconds s of boot-up. RTOS is very unlikely to fail, because failure within s of boot-up would have caused it to be unusable.

Prob(F fails in time interval (0, 0+s)) = approx. 0

(Since the process is started at time 0, and it cannot fail before it is started, this probability is unconditional.) The memoryless property of the time to next failure is prima facie not fulfilled. This means that its distribution cannot be the exponential distribution and thus that the overall failure process is not prima facie a Poisson process[5].

[4]These probabilities are technically speaking conditional probabilities: *Prob(F fails in time interval (t, t+m) given that it hasn't failed up to t)* is usually written *Prob(F fails in time interval (t, t+m) | F hasn't failed up to t)* and this is defined equal to *Prob(F fails in time interval (t, t+m)) ÷ Prob (F hasn't failed up to t)*

[5]We are aware that any simple counterexample such as this likely can be plausibly finessed somehow, but there are deeper characteristics that need to be considered to construe the failure behaviour as a Poisson process. This simple counterexample is not the end of the story.

A plausible construal of RTOS execution as a memoryless process is that one run of the process is defined by the boot-up of the RTOS and concluded with its shut-down. In that case, it is plausible that whether the RTOS succeeds or "fails" (however this might be defined for an entire boot-up–shutdown sequence) is statistically independent of whether the previous boot-up–shutdown sequence succeeded or failed. The statistical process would be discrete, a Bernoulli process rather than a Poisson process. To evaluate this Bernoulli process, all input to the RTOS from boot-up to shutdown must have been logged, and that sequence of consolidated inputs constitutes one mathematical input i to the single run of the Bernoulli process. That will usually be a very large amount of time-stamped data.

The distribution of that input data must be determined. If the history of RTOS is like the history of most OSs of the author's acquaintance, it is very unlikely that an exact sequence of inputs from boot-up to shut-down, as well as the relative-timing relations for time-dependent functions, will have occurred more than once. That means each mathematical input value as defined above has occurred precisely once. Thus the distribution will be a subset of the entire possible input space. It is also combinatorially infeasible for all possible inputs to the RTOS to have been observed: the subset will be sparse. (Relative timing of various inputs will be important. We omit this consideration here.)

The numbers for the Bernoulli process are valid only for the case in which the proposed future use of the RTOS has an identical input distribution to that cited in the operational history from which the numbers are derived. That is, exactly this sparse subset of all possible inputs must occur in the future. Quite how this could validly be demonstrated in any practical case remains a mystery to us - given the combinatorics it seems implausible.

We conclude that establishing the reliability of RTOS practically using the Bernoulli/Poisson mathematics in this manner looks close to infeasible. Yet Annex D currently states in its second sentence "*This approach is considered particularly appropriate as part of the qualification of operating systems, [etc.]*" !

A nice twist was recently recounted by Bernd Sieker [14], who observed that some portion of the working memory of the Linux kernel is not initialised on boot-up, in order to serve as a seed to the random-number generator (RNG). There is a story that someone once observed that some part of memory was not initialised, and wrote code to zero it out. The RNG stopped working effectively and thereby also certain important kernel functions. The seed must be there. Since the (correct) OS thereby does not start from a defined initial state, a run from boot-up to shut-down cannot count as a run of a renewal process, since there is memory (both literally and figuratively) left over from a previous run. The mathematics of Poisson processes are prima facie inapplicable. (There are plausible ways of fixing this quandary which we do not consider here.)

3 Assessing Failure-Freeness

Suppose one has observed N invocations of an on-demand function without failure. One wants to know with a certain degree of confidence that the probability of failure, $(1 - p)$ of the function is sufficiently small, according, say, to the specification of the SIL level. Alternatively, that the probability of success of the function, p, is sufficiently large. The question is statistical: one has a certain series of observations, one has established for certain that one is observing a Bernoulli process in the failure behaviour and one wants to know what the likelihoods are that one is observing such a process with a satisfactory value of p. Similar considerations apply to continuous functions whose failure behaviour forms a Poisson process. The mathematics is given in [9].

In order for the mathematics of Bernoulli and Poisson processes to be applied, real failures must be considered, not just those failures which might have been observed. Thus

- There must be perfect failure detection: all failures of the software must be (have been) observed and recorded (this requirements has been noted above)
- In particular, failures may be masked by other failure phenomena. These masked failures must also somehow be observed and recorded[6]

Some assumptions that lead to the appropriate characterisation of the statistical behavior of software malfunction as a Poisson process are

1. That the behavior of the SW on a given input E is dependent only on the value E and not on the internal state of the SW at the time E is input
2. That the likelihood of a malfunction remains constant over time

These amount to plausible approximations of the mathematical memoryless condition. Condition 1 may prompt an observation that relatively little safety-related software, if any, fulfils it literally.

However, internal state is often used to record some history of inputs in order to determine a change which is significant for the function of the software, and in many cases a transformation may be effected to render the function memoryless. For example, a temperature-rise-control function *TC* monitors temperature and, if a temperature rises too fast, executes a mitigation. *TC* determines that the temperature is rising too fast through comparing a sequence of timestamped temperatures, delivered in suitably timely fashion. Internal state (memory) is used in *TC* to record the sequence of temperatures and times for determination of the temperature profile. *TC* does not literally fulfil Condition 1; indeed internal state is used essentially. Its behavior, however, is essentially memoryless, as follows. Factor the function which *TC* performs into two. The first function *DT* aggregates temperature over time, and determines the rise in temperature over discrete time periods: it is a discrete approximation to the first time derivative of temperature. When this value exceeds a threshold, it signals a protection/mitigation process *Prot* to start. *Prot* is literally memoryless and embodies a Bernoulli process. *TC* consists of the pipeline *(DT >> Prot)* and the Bernoulli-process assumptions and mathematics apply because *Prot* embodies literally a Bernoulli process (resp. a Poisson process if considered over time rather than through demand-invocations).

Other examples of such effectively memoryless processes are some protection systems in nuclear power plants. Such a system starts from a given initial state and reacts to parameters over a short time period, in order to execute its protection function. The time period over which input is aggregated is shorter than the time period between invocations of the software, and the software is reset to the initial state after execution of its function. Different invocations of this function, and thereby failure or success per invocation, are independent in the appropriate sense for Condition 1. Furthermore, there is some constant probability of failure, dependent on the exact sequence of inputs per invocation and the distribution of those inputs.

Generalising, software which implements on-demand functions and which is returned to a defined initial state (reinitialised) after each invocation, such as protection functions, often fulfil the above conditions. There are specific verification conditions which must be assured: that such SW starts each time in a pre-defined initial state must be proven (if it is always the case), or alternatively it must be recognised without exception when the SW is not in the initial state, and such exceptional invocations omitted from the statistics and the statistical evaluation of the software.

Another class of functions which fulfil the general conditions are cyclic functions: software functions which are reinitialised at predetermined times or at other points (say, dependent on internal state). The sequence of inputs from an initial state up to the point at which the software is reinitialised form one mathematical input, namely a sequence. Detailed reasoning is given in [6].

This short discussion should suffice to indicate that the restriction imposed by the condition of memorylessness is not as severe as it first appears. It does, however require a degree of experience to assess practical software execution as appropriately memoryless.

[6] A discussion of failure masking and resolution is beyond the scope of this note.

4 Summary of Evaluation Conditions Discussed Above

- The failure behaviour during operation of the software must be shown to constitute a Bernoulli process, respectively a Poisson process
- The conditions under which the mathematics of Bernoulli, resp. Poisson processes is valid must rigorously pertain: in particular
  - the distribution of inputs in the operational history must be identical to the distribution in the intended further use
    - in particular, the operational conditions under which the statistical data has been gathered must be statistically identical to those in operation, and both have to be statistically identical to the future intended operational use
  - That such SW starts each time in a pre-defined initial state must be shown (if it is always the case), or
  - Alternatively, it must be recognised without exception when the SW is not in the initial state, and such exceptional invocations omitted from the statistics and the statistical evaluation of the software
  - The time to next failure of the real program, as binary machine code running on electronic hardware, must be shown to be exponentially distributed, equivalently to fulfil the memoryless property; that is:
    - At any point in time, the time to next failure must be independent of the previous history of the execution of the program
- There must be perfect failure detection: all failures of the software must be/have been observed and recorded
- In particular, failures may be masked by other failure phenomena. These masked failures must also be/have been observed and recorded
- There must be very high assurance in the absence of confounding factors, such as unremarked environmental parameters, in the operational-history logs

Acknowledgements

The author acknowledges the substantial contribution of Bev Littlewood in discussions of this material. The work reported here is part of a longer joint project to design a guide for system safety engineers to the statistical evaluation of critical software, one aim of which is to contribute to the modification or replacement of Part 7 Annex D in the next edition of IEC 61508.

References

[1] J. Braband. Personal communication, (2015)

[2] R. Butler, G. Finelli. "The Infeasibility of Quantifying the Reliability of Life-Critical Real-Time Software", *IEEE Transactions on Software Engineering* **19(1)** pp. 3-12, (1993)

[3] R. Faller. Personal communication, (2014)

[4] International Electrotechnical Commission. "IEC 61508, Functional safety of electrical/electronic/programmable electronic safety-related systems, Parts 1-7", (2010)

[5] P. B. Ladkin, "An example of a safety-critical element with deliberately unreliable function." *RVS White Paper 8*, (2015). Available from http://www.rvs.uni- bielefeld.de/publications /WhitePapers/LadkinFallerExample20150101.pdf, accessed 2015-06-24

[6] P. B. Ladkin, B. Littlewood. "Practical Statistical Evaluation of Critical Software", (2015). Available from http://www.rvs.uni-bielefeld.de/publications/Papers /LadLitt20150301.pdf, accessed 2015-06-24

[7] P. B. Ladkin, B. M. Sieker. *Safety of Computer-Based Systems*, draft textbook, Chapter 16 (in German), RVS Group, University of Bielefeld, (2011). Available from http://www.rvs.uni-bielefeld.de/publications/books /ComputerSafetyBook/index.html, accessed 2015-06-24

[8] B. Littlewood, L. Strigini. "Validation of Ultra-high Dependability for Software-based Systems", *Communications of the ACM* **36(11)**, pp. 69-80, (1993)

[9] B. Littlewood, L. Strigini. "Guidelines for Statistical Testing" *Report No. PASCON/WO6-CCN2/TN12*, City University London for ESA/ESTEC Project PASCON, (1997). Available from http://openaccess.city.ac.uk/254/, accessed 2015-06-24

[10] B. Littlewood, L. Strigini. " "Validation of ultra-high dependability" - 20 years on", *Safety Systems* **20(3)**, pp. 6-10, (2011)

[11] A. M. Ross, E. W. Weisstein. "Memoryless". From *MathWorld–A Wolfram Web Resource*. Available from http://mathworld.wolfram.com/Memoryless.html, accessed 2015-06-24

[12] Royal Academy of Engineering. *Global Navigation Space Systems: reliance and vulnerabilities*, (2011). Available from http://www.raeng.org.uk/publications/reports /global-navigation-space-systems, accessed 2015-06-25

[13] K. Siegrist, *Virtual Laboratories in Probability and Statistics*, University of Alabama at Huntsville, (1997-2014). Available from http://www.math.uah.edu/stat/, accessed 2015-06-25

[14] B. M. Sieker. Personal communication, (2015)

[15] M. Thomas. Personal communication, (2011)