ISSN 2754-1118

Evaluating Software Execution as a Bernoulli Process

Peter Bernard Ladkin

Causalis Ingenieurgesellschaft mbH, Bielefeld, Germany

Abstract

Daniels and Tudor consider evaluating software statistically through modelling execution as a Bernoulli Process (Daniels and Tudor 2022). They give examples and adduce considerations which they claim show that such modelling is "flawed". However, neither the examples they give, nor the considerations they adduce, follow the constraints necessary for modelling any process as a Bernoulli Process. I show that here.

1 Introduction

1.1 Preamble

The statistical evaluation of safety-critical software is an important subject. I and colleagues have experienced over the last decade that this is one of the most misunderstood topics in safety-critical software evaluation. It also seems to be one of the most controversial; maybe these two phenomena are correlated. In a recent article, Daniels and Tudor claim that evaluating software execution using modelling as a Bernoulli Process is "flawed" (Daniels and Tudor 2022). However, neither their examples, nor the considerations they adduce, satisfy the constraints required for any Bernoulli Process modelling to succeed, as I show here. The appropriate conclusion would be, not that Bernoulli Process modelling is flawed, but that it is important to follow the constraints if you want trustworthy results.

The basic premiss of statistical evaluation is that one can observe a phenomenon occurring, note its characteristics, and analyse those data to see if there are any regularities. Those phenomena which do not exhibit obvious regularities can be said to behave stochastically. Even so, stochastic processes exhibit properties which can be characterised, such as the number of occurrences of a discrete phenomenon, or the relative frequency of values of some quantity. For many stochastic processes, there is an underlying probability distribution, as it is called, in which values of the characteristic can be taken to occur with a given probability.

Some probability distributions follow from very general, common characteristics of stochastic processes, and so occur ubiquitously; for example, the binomial distribution for discrete-valued events, and the normal (Gaussian) and Poisson distributions for continuously-valued events; there are others.

1.2 The Binomial Distribution in Mathematics and Engineering

The binomial distribution might underly a series of discrete events with exactly two discrete outcomes, which we can call success and failure. The binomial distribution occurs when

there are repeated discrete events with one of two outcomes, in which the outcome of a later event is probabilistically independent of the outcomes of any earlier events, and in which the outcome of any event is given by a certain, fixed, probability. There are infinitely many binomial distributions, but when I specify:

- the probability *p* of an individual event being a success; and
- the number of events *n* I observe;

then I can say, through mathematical analysis, what proportion of the *n* events I expect to result in success. I can also say what the chances are that some completely different proportion will be exhibited in my observations from that which I expect. Key to this is, of course, that each event indeed occurs with a given probability *p*. If this value varies over the events, then no dice; the events do not form a binomial distribution and I have to try something else.

Given that I have good qualitative reason to think that a certain kind of probability distribution is exhibited by a class of events, which is usually arrived at through prior consideration of how those events occur, then what I generally won't know beforehand are the values of the parameters. For a stochastic process, which I can argue qualitatively follows a binomial distribution, I can observe it for an appropriate number of events and attempt to estimate what I think the values of those parameters might be. The important parameter for the binomial distribution is the *fixed* probability of success. I need to be aware that what I observe may not be typical: stochastic processes are not deterministically law-governed, and even though I might expect a repeated coin toss (a prototypical example of a stochastic process taken to follow a binomial distribution) to yield me a roughly equal number of heads and tails on a long series of tosses, it just might give me all heads.

If all-heads is what I see, then it makes sense to ask:

- what is the possibility that I have an evenly-balanced coin, but that I just happened to get a run of all heads?
- what is the possibility that my coin is not at all evenly-balanced?

These are typically expressed as *confidence*[1]:

- Given the results, what is my confidence that the coin is evenly-balanced?
- Given the results, what is my confidence that the coin has a specific bias?

The even balance of a coin is represented by a probability *p* of 0.5; the bias of a coin by some other value of *p* between 0 and 1. Given my results, it turns out I can calculate my confidence in each of those circumstances (in each of the specific values of *p*).

This is all just maths, but here comes the engineering. Suppose a company is interested in coin-tossing. It manufactures coins for tossing. It claims to produce unbiased coins for tossing and it wishes to persuade its customers that it is very good at what it does. Then it can do a lot of tossing of each of the various coins it has produced, and, given the results, it can argue to customers:

[1] "*Confidence*" is a technical term in statistical evaluation, usually expressed as a percentage from 0% to 100%, also known as "confidence level". This derives from the notions of "confidence interval" and "confidence set" (a confidence interval is a confidence set which consists of a continuous interval of numbers, for the case in which the value being estimated is a number). The notion of confidence set is explained in, e.g., (Siegrist, Chapter 7 Set Estimation, Section 1, Introduction); (Bedford and Cooke 2001, Section 4.3.3).

- You can have confidence *X* that a coin you bought from us is unbiased with an error of at most ε[2]

A customer can respond, "A*ll well and good, but your tests were all performed in summer, during business hours (0900-1700 UTC) and inside in windless conditions, and we are going to want to use your coin in winter during bar happy-hours of 1800-2000 outside in the wind in the beer garden. How should this affect the confidence?*" It is an important question, answered for example as follows:

- the time of day does not causally affect a coin toss in any plausible way;
- the season *per se* does not causally affect a coin toss, but the temperature in which the toss is performed modifies the coin (which is larger when it is warmer) as well as the atmosphere (which is less dense when it is warmer);
- the humidity of the atmosphere might affect the toss (humid air is less dense);
- wind may well affect a toss, as far as we can see.

Then there are possible causal effects which have not been introduced, for example the mechanical manner of the toss: if the translational velocity and rate of rotation are not stochastic but rather uniform, then such uniformity might causally affect the result.

You can assess these possible causal effects by performing the coin toss in the circumstances described and seeing what happens. The most the manufacturer can say is: "*Here are the circumstances (believed to be all the causal circumstances) in which we performed our tosses; if you perform your tosses in these circumstances, you will get the results with the confidence we said".*

We call these circumstances the *environment*, alternatively the *operational environment* or *operational profile*, of a stochastic process[3].

1.3 An Example from Safety Engineering

Here is an actual example of this reasoning in safety engineering. UK civil-nuclear regulations require that emergency systems are statistically evaluated. A SCRAM device is such a system; it causes an emergency shutdown of a nuclear reactor. A SCRAM device consists of various pieces of software-controlled hardware, plus communications, also to a large extent software-controlled nowadays. To a very high (specified) level of confidence, the SCRAM device must exhibit a very, very low (specified) failure rate (very, very low estimated value of *p*), in a given environment (specified). Through decades of experience and analysis, regulators, manufacturers, and operators (let us call these the operational stakeholders) believe they understand the environment very well.

Not only that, but the operational stakeholders believe they understand the characteristics of the hardware, and the characteristics of the communication hardware + software being used, so they can include these characteristics, if they wish, in the operational profile, and thereby apply the above estimation to the software alone.

Let us consider a theoretical SCRAM device. Actual SCRAM devices are similar to this, but somewhat more complicated in ways which for our purpose are uninteresting (that is, not causal to what we care about).

---

[2] "*Unbiased with an error of at most* ε" translates as "*p lies within the interval* $(\frac{1}{2} - \varepsilon, \frac{1}{2} + \varepsilon)$"

[3] Why all these different terms? Historically, different constituencies interested in statistical evaluation have developed their own terms, and it happens that these three are common in engineering.

Sole input to the SCRAM device is a line which represents an "abnormal state" in some analogue fashion; if it remains "high" for a specific period of time, a SCRAM Command — a single high bit — is to be issued. The device is thus an arbiter. It has internal memory. After a SCRAM Command is issued, the device is completely reinitialised before being activated again: all registers and memory are reset to zero.

The device may fail: all arbiters may fail (Lamport 2012).

Can we model this circumstance with a binomial distribution, as above? I think so (as does the nuclear industry in the UK, and many professional statisticians). Given the operational profile, starting in the initial state (all registers and memory and anything internally computationally causal are set to the specified initial state). The device is run until it provides an output, and the output recorded.

1.4 Trials, Bernoulli Trials and Bernoulli Processes

What does it mean that "*the device is run*"? Recall the two outputs: "success" and "failure". Does this describe what the device does? Not completely. There it is working in its operational environment. Suppose the line does not remain "*high for a specific period of time*". Then we would not want output to be a single high bit raised on the line. So there has to be some analysis of the device to develop confidence that this does not happen (to the given level of confidence). We don't follow this issue further here, because it is not best modelled as a discrete process — it is a continuous process-in-time with discrete occurrences of failure (when the output bit goes high without having experienced the trigger input). We are interested in the phenomenon that, if there is indeed an input trigger ("*line remains high for a specific period of time*") an output (high bit) is generated (*success*) or not (*failure*). That is a discrete process, with three components:

- A given starting state;
- The occurrence of a trigger input; and
- The output (success or failure).

When we have these three, this process forms what we may call a *trial*. A trial is an event with a beginning and an end, and an outcome. A particular class of interest here is that of *Bernoulli trials*. A succession of Bernoulli trials is called a Bernoulli Process (Siegrist, Chapter 10, Bernoulli Trials) (Ladkin 2017, Chapter 1, Software, the Urn Model, and Failure) (Ladkin and Littlewood 2016). Not every trial is a Bernoulli trial, and not every repeated trial forms a Bernoulli Process.

A Bernoulli Process is nothing more, nor less, than a sequence of trials of a process governed by the binomial distribution. It follows that there is reason to attempt to model a situation as a Bernoulli Process when there is reason to think that the underlying probability distribution is or might be binomial. Accordingly, the chances of success (failure) in a trial in a Bernoulli process must be the same for any trial (that p in the binomial distribution); in particular, it must be independent of the history of the trials in the process. If you don't have reason to think that the underlying probability distribution can be represented as binomial, then you equally don't have reason to attempt to estimate the parameters of that distribution through treating the process as a Bernoulli Process.

1.5 A Trial Which is Not a Bernoulli Trial

Suppose the registers, memory, etc., are not reinitialised before performing the trial. The software may have some internal one-byte counter that is typically incremented three or four times in the course of a computation. If that counter starts at zero, the computation proceeds as one would expect. But suppose the counter has a value of 255. Then incrementing it will cause it to overflow and (let us suppose) trigger an exception condition. So, if the counter starts with a value of 255, then the chances of failure are 100% ($p = 0$). The probability of success of a trial with counter starting at zero is different from the probability of success of a trial with counter starting at 255. This situation does not fit a binomial distribution; such trials are not Bernoulli, and a sequence of them is not a Bernoulli Process.

2 Some Claimed Difficulties with Interpreting Software Execution as Bernoulli Trials

2.1 Preamble

If you are trying to model a situation as a Bernoulli Process, you first need to define what you take to be a trial, the events whose underlying distribution you are attempting to estimate; and, second, you need to argue that these trials are Bernoulli, that the underlying distribution is binomial. The probability concerned is the probability of that Bernoulli trial being successful.

Consider the way in which Daniels and Tudor introduce their topic. They speak of the "*probability of Success* [being] *the same every time the software is executed*" (Daniels and Tudor 2022, Sub-section 3.3.1). They don't define a trial; in that they don't define a starting state. The end state they consider is presumably the output "*Success*"; they don't address any concrete termination with an output of "*Failure*" — this leaves the possibility that software they are considering evaluating might not terminate. If that is the case, then there will be no trial. As they note, and as we have seen above, the interpretation of the probability (as imprecisely described above) taken at a snapshot in time can be very different, depending on the time and the internal state of the software at that time.

I see no plausibility in a suggestion that, taking executing software at a random point in its execution, and asking whether the computation will result in success, you can model this situation as a binomial distribution. What is "flawed" here is such a modelling attempt.

If you want software execution to be modelled as a Bernoulli Process, then you consider it as executing a series of Bernoulli trials, and it is the probability of success of the Bernoulli trial which is constant. If you identify a trial, but can't successfully argue that repeated trials of this sort are Bernoulli, then you are out of luck. Ladkin (2017, Chapter 1) shows that, very often, you can; but you do have to pay attention to the constraints of interpretation.

2.2 A Wrap-around Example

The example considered in Sub-section 1.5 above is isomorphic to the example considered in Daniels & Tudor (2022, Sub-section 3.3.2), with its counter wrap-around (in their example from 65535 to 0). Daniels and Tudor assert correctly that such a situation is not modelled as a Bernoulli Process in this way (Daniels and Tudor 2022). However, they do

not define the situation they are modelling: they do not specify what is to count as a trial, let alone attempt to determine if it is Bernoulli.

The stipulated fixed probability p in the binomial distribution has a mathematical consequence that Daniels and Tudor ignore in their discussion. The result of a trial of the system in Sub-section 1.5 is stochastically dependent on the previous trials: previous trials raised the value of the counter to 255 and resulted in $p = 0$ for this trial. In a Bernoulli Process, all trials are stochastically independent of one another; if they aren't, the process is not a Bernoulli Process.

That there is no Bernoulli Process at work here is not a subtle observation. This situation is well known to safety-critical system software developers — indeed, to most software developers. When you program a discrete function in software and you want to control its behaviour, you start it in a known starting state, usually by reinitialising (setting to zero) all memory and registers. Why? Because then the course of its computation will not be dependent on values created in previous computations. That is a good thing for software discrete-function programming. It is also a condition which must be assured if you want to evaluate a succession of calculations as a Bernoulli Process.

2.3 Summary of Conditions for Regarding Trials as Bernoulli

The discussion so far has illuminated the misleading introduction Daniels and Tudor (2022) give in their Sub-section 3.3.1 to attempting to model software execution as a Bernoulli Process.

- First, you must define what you take to be a trial with two possible but definitive outcomes.
- Second, it is not the "*probability of Success every time the software is executed*" that is relevant but rather the probability of success of the trial.
- Third, a trial must be stochastically independent of previous trials. If you don't have that, then the trials are not Bernoulli trials, and modelling as a Bernoulli Process cannot work.

The key is whether the underlying process plausibly follows a binomial distribution. If it doesn't, then attempting to estimate its parameters as if it were is unlikely to be fruitful.

2.4 Stepping Back and Looking

It is worth stepping back from the specific example of software execution for a moment to look at general characteristics of statistical evaluation, because some of the phenomena being highlighted have nothing to do with software *per se*.

Example: Suppose you are a Caucasian male of 60 years of age living in Western Europe, with no known impediments to your health. You are standing up, about to take two strides forward. What are your chances of dying while doing so?

- If you are standing on the unprotected parapet of a 50-story building, your chances are likely 100%.
- If you are standing at a tram stop, facing the tracks, your chances are arguably the period of time during which a tram is approaching the station and unable to stop before the point at which you cross its track, divided by the rest-of-the-time.

- If you are standing in your garden with only your lawn before you, the chances are arguably those of a person of your age and medical condition suffering an anomalous medical event (heart failure, aneurysm, etc.) within that period of time.

The first point to note is that the original situation is very much underspecified if you want to draw informative conclusions. The second point is that, in certain more specified circumstances, a probability can reasonably be assigned; indeed, it can reasonably be assigned in all these three cases.

So it is with the statistical evaluation of software; if the software is in an unknown state at the beginning of your trial, then the probability of its resulting in Success is equally all over the place. If the circumstances of its operation are sufficiently constrained according to known modelling requirements, then regularities can likely be identified and estimated.

This is well known to operators of computers. Back in the 1970's and 1980's, reboots of computer systems would succeed or fail, with outcomes not known deterministically to their operators, but known by "rule of thumb". Every tenth reboot would hang. Or every sixth re-boot. Every operator "knew" the proportion. I was involved in such proceedings. When the proportion changed, people would suspect something else was going wrong, and start looking at possible hardware weaknesses and so on. None (or, very few) of these operators were trained statisticians. They generally didn't know what a Bernoulli Process is (although the situation they were evaluating was arguably so modelled). They were just people with an experienced eye for regularities. That is what statistical analysis is — having an eye for regularities. If you can't see a binomial distribution before you, don't try modelling the situation as a Bernoulli Process. If you can and your data fits the Bernoulli Process assumptions, by all means try it and see.

On one point, Daniels and Tudor are very much correct. If you are running software which produces "*Success*", "*Failure*" output, you can't just pull out Table D.1 of IEC 61508-7:2010 Annex D, look up numbers and suppose without further consideration that they apply to your software[4]. More work is required to ensure that you are modelling your software execution appropriately.

2.5 Stepping Outside the Operational Profile

In their Sub-section 3.3.3, Daniels and Tudor consider the execution of software which fails (necessarily) on a leap-year date. They say "*Assuming there were no other defects, had we started running the program on 1 March 2016, then by 28 February 2020 we would have observed 35,040 hours of failure free operation. We would have had a high degree of confidence that the software is correct, yet it would have failed the very next day, 29 February 2020.*"

I have no idea how Daniels and Tudor might have obtained such a "*very high degree of confidence*", and neither should any other reader. They have not attempted to explain what model they are using which renders it. I also doubt they could plausibly have arrived at any such high degree. Construing an execution as a Bernoulli Process, or Poisson Process, or any other statistical evaluative model does not usually enable you to conclude with any degree of confidence, let alone a "*high degree of confidence*", what the outcome will be when the software executes with a hitherto unseen input.

[4] Anecdotes abound amongst safety-critical system engineering consultants of clients wanting to do this.

Since (by hypothesis) the software in their example crashes on a leap-year date, it follows that the date is a causal input to the software. Let us consider such software. It is obvious that all dates on which the software has been observed to run are less than or equal to today, *T*. Assuming you don't know anything else about it, running software to which date is an input on a sequence of dates $<T$ enables no useful conclusion to be drawn on how the software will behave when date inputs are $>T$. Any "*high degree of confidence*" is misplaced.

It is a commonplace remark that statistical evaluations and estimates of future reliability can only occur for the operational profile which has been observed during statistical evaluation. When date is an input, the future operational profile is *disjoint* from the observed profile. It is hard to imagine deriving any useful conclusion with much confidence if the behaviour on the observed profile is all you know about the software[5].

I also note that such an example is not appropriately modelled using a binomial distribution. Daniels and Tudor are considering numbers of hours of failure-free operation. There is no notion of elapsed time which can be brought into rendering the software execution as a Bernoulli Process. The closest one can come is a parameter which is the number of Bernoulli trials. But number of trials is not necessarily related to elapsed time; in, say, 3 hours of operation, there might occur 0, 1, or 500 Bernoulli trials. If Daniels and Tudor want to reason about hours of failure-free operation, they cannot plausibly use Bernoulli Processes. A more appropriate model would be a Poisson Process or some other renewal process.

2.6 The Problem of Unknown/Inadvertent Input

As in the leap-year case above, it may indeed be that a particular software has inadvertent inputs that have not been recognised by the designers or users. If that is so, then when statistical evaluation if performed on an operational profile, these unrecognised inputs will likely be omitted from the explicit observed operational profile. Reasoning about the demonstrated reliability of the software will thereby be misleading.

This is a practical problem with statistical evaluation that must be addressed during any evaluation. Often, such inadvertent dependencies can arise through the use of library functions. It is said that much software is dependent on the date/time as well as on the availability of the Global Positioning System (GPS), which has no necessity to be dependent on those parameters, simply through using library functions which include them. For example, Hobbs recounts an incident in which GPS signals were jammed in the North Sea and a ship taken through to see what would happen (Hobbs 2015, p19[6]). Amongst other things, the ship's radar stopped working. "*The experimental team contacted the company that manufactured the radar and was told that GPS was not used within it. They then contacted the manufacturers of components in the radar and found that one small component was using GPS timed signals."*

It is crucial that safety-critical system developers understand exactly what is input to their system. Building software which is dependent on unanalysed third-party functions from a library is generally unacceptable. Unanalysed dependency not only breaks accurate statistical evaluation, but much else besides.

[5] In such a situation, it is usual to attempt to factor the software operation architecturally into a part dependent on date, and a part which is not, and the effects dependent on date be treated static-analytically. Such factoring may or may not succeed. Any reasonable statistical evaluation, however, depends on it succeeding at least in part.

[6] Credit is given to Martyn Thomas.

2.7 Defect Clustering

The structure of this section parallels the sequence of examples in Daniels and Tudor Sub-section 3.3. In their Sub-section 3.3.4, Daniels and Tudor address software defect clustering; however, I don't see what that phenomenon might have to do with modelling software operation as a Bernoulli Process.

2.8 Non-operational Modes and Easter Eggs

In their Sub-section 3.3.5 Daniels and Tudor concern themselves with non-operational modes and Easter Eggs. Such modes may be undesirable for many reasons, but I don't see what problem they pose for statistical evaluation. Such modes have trigger input sequences. If the natural occurrence of such trigger input sequences is stochastic, then such executions indeed involve a fail of the functional intent of the software, but are thereby incorporated into any of the common statistical models (Bernoulli, or renewal). If the natural occurrence of the trigger sequences is not stochastic, then that indeed makes for difficulties in modelling.

2.9 Duration

In Sub-section 3.3.6, Daniels and Tudor address a phenomenon they call "*duration*". What they say, in full, is:

> *From a statistical point of view, it does not matter whether we run one copy of the software for a million hours or a million copies of the software for an hour each; both experiments result in one million hours of operation. In the real world, these two experiments are not the same. Running a million copies of the software for an hour each does not give us confidence that the software will run for two hours, let alone for days, months or years. Why do we expect the environment's behaviour for one hour to be the same as the environment's continuous behaviour for a million hours?*

If one is talking about software which executes over a period of time which is being measured, then a Bernoulli Process model is not likely to be appropriate, but rather a continuous-time model, for the reasons mentioned in Sub-section 2.5 above. But, if one nevertheless insists on trying to do so, then the closest pertinent parameter would be the number of executions of Bernoulli trials.

Translating what Daniels and Tudor say to the language of Bernoulli Processes, they would be contrasting running one copy of the software for a million trials with running a million copies of the same software for one trial each.

Both these options seem to me to define the same, or very nearly the same, Bernoulli Process. Daniels and Tudor want to claim that "*in the real world*" these two circumstances are different. They provide no argument for this contention. I don't agree with it. I think these two trial situations are the same in the pertinent respects.

3 The Upshot

On the basis of the critique exhibited in their Sub-section 3.3, Daniels and Tudor claim, "*The use of a Bernoulli process to model software failures is fundamentally flawed*" (Sub-section

3.6) and, "*We have shown in this paper that software reliability models that assume that software execution is a Bernoulli process are flawed*" (Sub-section 5).

I disagree that they have "*shown*" this. They have introduced some examples of faulty modelling and faulty statistical inference, which I have considered above in detail. The conclusion to draw is surely not that the statistical approach to evaluating software is flawed, but rather that if you apply a technique in various flawed ways then you are unlikely to get worthwhile results.

4 Reasoning With and About Probabilities

In their Sub-section 3.3.1, Daniels and Tudor speak about, "*The probability of Success* [being] *the same every time the software is executed*". This suggests that they think about probability as some objective characteristic inherent in the situation, in this case attaching to executing software. In the coin toss, similarly, one might think of the inherent probability of the unbiased coin turning up heads, as ½. This could be called the Laplacian conception of probability, as a property of an object exhibited in a circumstance.

Nowadays, no one takes probabilities to be such entities as this. Statisticians speak of the probabilities of events, of something happening (Newby 2020). Inductive logicians speak of the probabilities of sentences or assertions (Hacking 2001) (Howson and Urbach 2006) (Adams 1998). The Boolean logic of sets and propositions is more or less the same (complements, unions and intersections correspond to negations, disjunctions and conjunctions of propositions describing those sets), so one can correlate event classes with propositions, and vice versa. Probabilities in general are nowadays taken to be any assignment of (say) numbers between 0 and 1 to a Boolean algebra of sets or propositions which satisfy the Kolmogorov axioms (Hacking 2001). The idea of probability as being something inherent in a situation has lost much of the flavour it might have had to Laplace and the other pioneers. Probability nowadays is more like a collection of numbers which may be imposed on a situation according to certain constraints. An interpretation, not an inherence. There can be different interpretations for the same situations.

In fact, there are many contemporary conceptions of what probability might be. The two most common are the *frequentist*, in which a probability is a proportion which arises when an event is repeated an unbounded number of times, and the *Bayesian*, in which it represents something like a person's best guess, given original circumstances modified by experience with experiments (often called *subjective probability*).

In his introductory book on statistical evaluation, Spiegelhalter lists classical probability, 'enumerative' probability, 'long-run frequency' probability, propensity or 'chance', and subjective or "personal" probability as five different conceptions (Spiegelhalter 2019, pp 217-8). Furthermore, he says, *"...I take the view that any numerical probability is essentially constructed according to what we know about the situation — indeed probability doesn't really 'exist' at all (except possibly at the subatomic level)*". Such a conception, that probability doesn't really exist at all, is *prima facie* hard to reconcile with such a concrete-seeming quantity as the "*probability of Success* [of] *software* [being] *executed*".

Spiegelhalter also has something to say about the phenomenon of misunderstanding in probability and statistics (Spiegelhalter 2019, p209):

> *"I am often asked why people tend to find probability a difficult and unintuitive idea, and I reply that, after forty years researching and teaching in this area, I have concluded that it is*

> *because probability really is a difficult and unintuitive idea. … Even after my decades as a statistician, when asked a basic school question using probability, I have to go away, sit in silence with a pen and paper, try it a few different ways, and finally announce what I hope is the correct answer".*

Yup — been there, done that.

5 An Issue Not Addressed

Engineers reading Daniels and Tudor (2022), and then my commentary, might well be tempted to think that, if statistical reasoning is as tricky as this, then why don't we just avoid it altogether? Then no one will be misled. One answer is that, if we are not prepared to gather reliable data on software-based system performance and analyse it, there are pressing industrial problems which cannot be solved.

Consider Michael's Problem[7]. Michael's company produces sensors for safety-critical applications, typically for measuring the basic parameters of chemical and other industrial processes. The sensors are software-based, and haven't failed for software-related reasons in decades. They were not developed according to IEC 61508:2010 because that standard didn't exist when they were designed and built. But IEC 61508:2010 says that you can't use a piece of software in a new safety-critical application unless it is developed according to its precepts. The sensible engineering approach is surely to use the company's sensors *as they are* in a new application because they have proved themselves in use. A much less sensible engineering approach, but the approach nominally required by the standard, is to reimplement the software according to the precepts of IEC 61508:2010 and then use the extant hardware with the reimplemented software. You would be thereby replacing something known for a long time to be reliable with something with low operational experience that might do as well, or, alternatively, it might have a bug or two which need discovery and mitigation.

Avoiding statistical evaluation, as Daniels and Tudor advocate, will not and, maybe, cannot solve Michael's Problem.

6 Conclusion

Statistical evaluation of systems in their operation and the software which partly controls that operation is a task which naturally arises in engineering, for example in Michael's Problem.

But it seems to be tricky for many engineers; there appear to be misunderstandings and pitfalls. Some of these lie in the nature of the subject, but some of them seem to lie in misconceptions of statistical techniques.

I believe that enumerating the historical ways in which engineers have misused/attempted to misuse statistical reasoning in high-reliability environments is a worthwhile informational and cautionary exercise. To this end, a clear discussion of the examples raised by Daniels and Tudor in their Sub-section 3.3 is worthwhile. I hope I have contributed this.

[7] Named after a colleague who has it.

Acknowledgments

Many thanks to Bev Littlewood, Martin Newby and Lorenzo Strigini for helpful comments.

References

Adams, E. W. (1998). *A Primer of Probability Logic.* CSLI Publications, 1998.

Bedford, T. and Cooke, R. M. (2001). *Probabilistic risk analysis: Foundations and methods*. Cambridge, UK: Cambridge University Press 2001.

Daniels, D. and Tudor, N. (2022). *Software Reliability and the Misuse of Statistics*, Safety-Critical Systems eJournal 1(1), SCSC-174, Safety-Critical Systems Club, January 2022. Available from https://scsc.uk/r174.3:1. Accessed 14th July 2022.

Hacking, I. (2001). *An Introduction to Probability and Inductive Logic*, Cambridge University Press, 2001.

Hobbs, C. (2015). *Embedded Software Development for Safety-Critical Systems*, CRC Press 2015.

Howson, C. and Urbach, P. (2006). *Scientific Reasoning: The Bayesian Approach*, Third Edition, Open Court Press, 2006.

Ladkin, P. B. (2017). *A Critical-System Assurance Manifesto*, RVS-BI, 2017. https://rvs-bi.de/publications/RVS-Bk-17-01.html. Accessed 20th July 2022.

Ladkin, P. B. and Littlewood, B. (2016). *Practical Statistical Evaluation of Critical Software*. Paper presented at the 24th Safety-Critical Systems Symposium, Brighton, UK. Available from https://scsc.uk/r131/8:1. Accessed 14th July 2022.

Lamport, L. (2012). *Buridan's Principle*, Foundations of Physics 42 (8), August 2012, 1056-1066. Available from https://lamport.azurewebsites.net/pubs/pubs.html#buridan. Accessed 14th July 2022.

Newby, M. (2020). *Private discussion on calculating probabilities of Covid-19 infection*, 2020.

Siegrist, K. (no date). *Random: Probability, Mathematical Statistics, Stochastic Processes*. Available at https://www.randomservices.org/random/. Accessed 23rd July 2022

Spiegelhalter, D. (2019). *The Art of Statistics: Learning from Data*, Pelican Books, Penguin Random House UK, 2019.