ARTICLE:

Robustness of software

By **Peter Bernard Ladkin**

In the English civil court case *Bates v Post Office Limited* (Bates 2019),[1] the properties of the Post Office Horizon transaction-processing system were investigated and argued. On Day 1 of the proceedings, March 11, 2019, Anthony de Garr Robinson QC for the Post Office defined "robustness" of the Horizon software-based system (Bates, Day 1 Transcript, §87):

> "The concept of robustness is a concept which involves reducing to an appropriate low level of risk, the risk of problems in Horizon causing shortfalls which have a more than transient effect on branches. So it involves both measures to prevent bugs arising in the first place but those measures are never going to be perfect and it includes measures which operate once a bug has actually occurred and triggered a result. It is both aspects of the equation. I don't say that the word "robust" necessarily means "extremely low level of risk", but what we say is that if you have a robust system it produces a result in which the system works well in the overwhelming majority of cases and when it doesn't work well there are measures and controls in place to reduce to a very small level the risk of bugs causing non-transient lasting shortfalls in any given set of branch accounts."

The concept of robustness was at the core of the defendant's argument, which was that Horizon was "robust", if not infallible.

We shall see that the vocabulary deployed by Mr de Garr Robinson is not used in this way in computing, whether or not it is conceptually clear.

In computing science, the notion of "dependability" is defined by IFIP Working Group 10.4 as follows:

> the ability to deliver service that can justifiably be trusted

(AvLaRaLa 2004), with the added comment that the definition stresses the need for the justification of trust. The authors note an alternative definition as:

> the ability to avoid service failures that are more frequent and more severe than is acceptable.

Both of these definitions involve social concepts. One is the concept of (justifiable) trust, the other of acceptability (of rate and severity of failure). Both stem from the intuition that computer systems are engineered systems that accomplish something directly or indirectly desired by human users. A computerised system for controlling an aircraft (called "fly-by-wire") fulfils certain expectations, or not, of its pilots, passengers and operators (and air traffic control). It may do so by happenstance, or it may do so because of careful development which has paid attention to all the ways in which control actions can go right or wrong, and has paid careful attention to ensure that the designers' decisions as to these control actions are correctly implemented according to precise engineering descriptions of their behaviour. That second alternative, careful development using methods known to enhance SW quality, is what is referred to by the term "justifiable" in the definition. IFIP WG 10.4 defines "reliability" as

> continuity of correct service

This involves being able to tell what is "correct" and what "not correct" in an encapsulated series of operations (a "service"), and in assessing that this service is provided "correctly" over a continuous time period. An appropriate way in engineering to tell what is correct and not correct in a service is to write an exact "specification" of the service from which anyone can tell unambiguously, using inference and precise meanings of terms, whether a given behaviour constitutes "correct service" or not. The engineering-science of behavioural specification is most well-developed in computer science, but is increasingly applied to wider areas of engineering.

IFIP WG 10.4 has published no explicit definition of software-system robustness.

[1] Bates v the Post Office Ltd (No 6: Horizon Issues) [2019] EWHC 3408 (QB), at http://www.bailii.org/ew/cases/EWHC/QB/2019/3408.html

The International Federation of Information Processing Societies (IFIP) is one source of definitions of terms used in computing. Another are international electrotechnical standards published by the International Electrotechnical Commission. The international standard definition of "reliability" in electrotechnology is:

> ability to perform as required, without failure, for a given time interval, under given conditions

(IEV, definition 192-01-24).[2] This is an absolute definition – "without failure". However, the term is more often used in software engineering to denote the degree to which a software-based system approximates this desirable situation. The discipline of "software reliability engineering" is concerned with estimating the chance of failure in operation, over a given time interval, under given conditions, to a given level of confidence (usually expressed either as a percentage or as a probability, equivalently).[3] Some consequences of results in software reliability engineering for the law are discussed in (LLTT 2020), particularly in the Appendix.

There is also an international standard definition of "robustness" in systems and software engineering, namely:

> degree to which a system or component can function correctly in the presence of invalid inputs or stressful environmental conditions

The definition has been current since at least 1990 (IEEE 610.12; in this document with an initial word "the") and was last published in 2017 (IEC 24765).

Notice the term "degree". This is a term with values; "always", "some of the time", "seldom", and/or "98%", "60%", "35%" are the kinds of values it can take. The question "is this system robust?" has thereby no meaning, or at most a derivative meaning. The question "to what degree is this system robust?" is compatible with the definition, and its answer would be one of the values suggested by the examples above.

One part of the definition which does not seem particularly applicable to software is the property of functioning correctly "under stressful environmental conditions". A computer might become too hot, say when the ambient temperature is consistently 30°C or higher, or the humidity might become too high, or water might enter inside the computer and enable all manner of short-circuits. Normally, devices such as computers have "given conditions", temperature ranges and humidity ranges and so forth, specified by the manufacturer, under which they are to operate. A computer could be said to be fairly robust if it continues operating in the expected manner even when the ambient conditions are outside those specified by the manufacturer. Software, however, consists of connected sequences of instructions[4] which the computer central processing unit (CPU) is to execute. The logic of those connected sequences is not influenced in the slightest by "environmental conditions", just as the operation of adding 2 and 3 together is not so influenced: the answer is 5, no matter what the ambient temperature. It follows that the robustness of software is given by the degree to which the software functions correctly when given invalid inputs.

This interpretation is consistent with that of the standard for software in civilian aircraft, including control software (ED-12C), which is:

> The extent to which software can continue to operate correctly despite the introduction of invalid inputs

What is an "invalid input"? Say the software takes data on values for patients' vital signs in a hospital, to form a database of those signs. One of those vital signs is body temperature. For living persons, this lies within a smallish range round about 37°C. In a database, the units can be implicit, so a temperature

[2] There are in fact some 40 or so definitions of "reliability" in various IEC standards, which may be reviewed by inputting this term to the on-line IEC Glossary (IEC Glossary). They are by no means all semantically equivalent, as shown by SemAn (Ladkin 2019) but the differences need not concern us here.

[3] The premier journal in software engineering is arguably the IEEE Transactions on Software Engineering. Many of the seminal papers in software reliability engineering in this "degree" sense have appeared in IEEE Trans. Soft. Eng. over the decades. The prestigious International Symposium on Software Reliability Engineering (ISSRE) is a conference which has been running for over 30 years.

[4] The correct term here would be a mathematical network of instructions. To avoid confusion with computer networks, which are collections of computers joined together by a communications medium such as Ethernet, or communicating with each other over electromagnetic signals such as WiFi, I speak of "connected sequences", even though this term is mathematically improper.

value can be taken to be around 37, if the units are °C. A value of 20, or of 50, would surely imply that the patient is already dead (indeed, it is hard to see how a value of 50°C could in any case be obtained from a patient in a normal ambient temperature). A value of -30 is out of the question. So there is some sense of a range for a living person; entering a value into the database which is outside this range would be entering "invalid input".

Notice that invalid input is a different phenomenon from mistaken input: it is perfectly possible for medical personnel to measure a temperature of 38°C and mistakenly enter "36" into the database. Entering a value of "36" is in this case mistaken, but it is not invalid in that it lies within the range of body temperatures which a living person can exhibit. Indeed errors of this sort involving mistaken input happen (rather too) frequently, are often put down to lack of care on the part of personnel, but are often realised nowadays to be inadvertently enabled in many cases by the design of the equipment they are using (CuBlTh 2015). Both "38" and "36" are valid values of body temperature, in that they both lie within the range of body temperatures which a living person can exhibit.

Another case of invalid input occurs when the software awaits a numerical value for some quantity, but receives a sequence of natural-language text. Or when it is awaiting a report in some text format such as PDF, and receives a number. Here, it is said there is a "data type" error. Numbers are a data type (indeed, whole numbers are a different data type from numbers expressed as decimals, or numbers in scientific notation, in most software) and text is another, different data type.

For more than fifty years, there have existed high-level programming languages which allow a programmer to specify data types, and which enforce, when they are compiled into machine code, that the data being manipulated conforms to its specified type. One of the first was Algol 68. Another is the teaching language Pascal, already in use in the early 1970's for teaching programming at the University of California, Berkeley. The language Ada, used nowadays for a wealth of applications for which reliability is crucial, is another high-level language with what is called "strong data typing". For more information on high-level languages and machine code and their relation, see (LadTho 2020). In such languages, one can define a data type which we might call <body temperature> (different symbols for such types are used in different languages) by specifying that <body temperature> is a range of decimal numbers between 35 and 38, in mathematical notation the interval I_1 = [35, 38] (all numbers x such that $35 \leq x \leq 38$). Or we might decide, alternatively, to specify the interval I_2= [33,40]. Under the first option I_1, a value of 34 would be "invalid", because it lies outside the interval I_1, and the compiled high-level program would return what is called an "exception condition"; it would recognise the input as invalid and act accordingly. Under the second option I_2, the software would accept the value 34 as valid, because it lies within the interval.

Attempting to compute with input values that are invalid, and halting the execution of the program with an error message, is one example of what is known as a "run time error". There are industrially-mature techniques for developing software which guarantee the absence of run-time errors (providing mistakes are not made in the application of the techniques). Correct application[5] of such techniques thus leads to software which is guaranteed to be perfectly robust in the IEC sense. Let me call it IEC-robustness, although as noted the concept is also accepted by ISO, IEEE and EUROCAE. IEC-robustness can be assured[6] by the use of software development methods which avoid run-

[5] It is important here that correct application is practical in industry settings, which indeed it is for the techniques of which I am speaking – software engineers can use these techniques routinely and correctly, although of course mistakes will always be made here and there. Other techniques are used for detecting such mistakes.

[6] Meaning that, as long as the computer hardware continues to run "normally", without damage or failure, and the software continues to run "normally" on the hardware, without alteration, such operational errors are guaranteed not to occur.

time errors.

IEC-robustness does not seem to be what Mr. de Garr Robinson had in mind when he used the term. In the case being tried (Bates 2019), what were called the "Horizon Issues" concerned properties of the Horizon system other than those arising from dealing with anomalous input, as well as more general, if occasional, notable phenomena arising during use of Horizon. The issue of invalid input does not explicitly occur amongst the "Horizon Issues" as determined by the case management. One wonders why a completely different use of the term "robust" was introduced by counsel from that which electrotechnologists the world over would be using. I refer henceforth to "GR-robustness" to indicate the concept which I believe Mr. de Garr Robinson invokes.

The background to the case is that users of the Horizon system, "sub-postmasters", were accused, and in some cases convicted, of crimes in that they were alleged to have conducted fraudulent transactions. In the specific fraudulent transactions at issue in the criminal proceedings, the Horizon system was contended by the prosecution to have completed the transactions correctly, and the subpostmasters to have committed fraud. Whereas the defence argument was often that no fraud had been committed and the Horizon system had completed certain transactions incorrectly; and possibly invented transactions sui generis that had not in fact taken place. The judgement in (Bates 2019) indeed determined that it was possible that Horizon completed certain transactions incorrectly (a phenomenon acknowledged by both claimants and defendant) and indeed exhibited transactions that had not in fact taken place (called "phantom transactions").

GR-robustness has more to do with reliability – here, spelt out for a transaction-processing system, of completing individual transactions correctly rather than incorrectly – than IEC-robustness. However, there is a second aspect of dependability which is surely relevant – that of not generating transactions which did not take place. This does not appear to be explicitly addressed in GR-robustness. GR-robustness includes, supplemental to the reliability property, a criterion concerning how the software behaves in case of failure.

Transactions are demands on a system. A demand arrives as a form of input (the transaction is initiated) and there is a closing point (the transaction is completed) at which the demand will have been accommodated successfully or unsuccessfully. To tell if a particular on-demand SW operation O is GR-robust, it is necessary

> A: To determine if/how O is reliable.[7] Namely, to what expectation of failure[8] and to what confidence level this expectation may be held.[9]
>
> B: To specify mitigants M1,....,Mk for the failure of O (and, presumably, to explain how the declared-mitigations actually mitigate);
>
> C: To determine the failure characteristics of M1,....,Mk (failure expectations, stochastic (in)dependence of/on each other, etc);
>
> D: To estimate P(O fails and M1,....,Mk all fail on a demand) to confidence level C. (And, of course, to determine what confidence level C is required.)

Consider first Condition A. It speaks of "expectation of failure" in, say, a time period. System failure behaviour might be "bursty", in which failures cluster in a relatively short time period, with longer periods of time between clusters, or more uniform, in which failures occur regularly with more or less even time periods between; and anything in between. When we

[7] Since all parties in (Bates 2019) agreed that the absolute definition of reliability was not attained by such a complex system as Horizon (which the prosecution in a related case expressed by saying that Horizon was not "infallible"), the reliability characteristics are those of "software reliability engineering": saying to what degree and to what confidence level the system is reliable.

[8] The terminology "expectation of failure", or "failure expectation", is coined here and not standard. The precise technical terms used (e.g., "mean time to failure", or "probability of failure on demand"), are dependent on which statistical modelling mathematics is used to capture the failure phenomenology of the software. We need a term which is independent of modelling technique, so I introduced one. Note also that the values taken by the concepts tied to specific modelling mathematics can be very different: e.g., if a system is perfect and exhibits no failures, then "probability of failure on demand" will be 0, but "mean time to failure" will be ∞!

[9] Recall that failure expectation and confidence level are not unique. We may have 70% confidence in less than one failure per day, as well as, at the same time, 90% confidence in less than one failure per hour.

speak of "failure expectation", we mean some kind of statistical average of the number of failures we expect to occur in a given time period, or the expected length of time until the next failure occurs. Where FE is the failure expectation, and C the confidence level, I have found that it is not generally understood that there are many pairs of valid <FE, C> corresponding to a given history of system behaviour. It seems sensible, for a given FE, to express the highest C which that FE can attain, given evidence N. Let us call it CMAX(FE). Then, for a given system, any given evidence N of its failure behaviour will result in many <FE, CMAX(FE)> pairs.

Given evidence N of system behaviour including failure, it follows there is a curve with independent variable FE (on the "x axis") and dependent variable CMAX(FE) (on the "y axis") which expresses the reliability of the system. This curve is surely the most precise output fulfilling Condition A. Less precise, but still practically useful, outputs would consist of a finite – small – sample of confidence levels CMAX(FE) and their corresponding FE values.

With regard to Condition B, it is not only important to say what the mitigations M1,....,Mk are, but to explain how they mitigate the failure, and indeed to provide an argument (even a formal verification) that those explanations are indeed correct; that M1,....,Mk indeed mitigate in the manner claimed. Recall the IFIP WG 10.4 emphasis on the justification of trust in their definition of dependability.

Condition C deals with the case in which the mitigations themselves fail. Suppose O fails because it requires access to a database DB, which becomes unavailable, and the mitigations M1,....,Mk all use DB. Then the mitigations are also unavailable. O and the mitigations M1,....,Mk are not stochastically independent in this case – they are subject to a common-cause failure, namely the unavailability of DB. Because they are not stochastically independent of each other, assessing the statistical failure parameters of each of M1,....,Mk by themselves will not generally help to determine the chances of unmitigated failure of O, because there are few if any ways of combining those parameters if they are not stochastically independent. There are of course ways in computer science of dealing with operations which might not be able to complete if they require access to a resource which is not available – "rollback" procedures and such constructs as "recovery blocks" have been known for decades. In a well-designed system, procedures M1,....,Mk involved in rollback and recovery will operate logically-independently of any operation O for which they are the recovery, but this independence must be demonstrated, and part of good system design is to enable such demonstrations to be relatively easily given.

Condition D will be fulfillable most easily if the stochastic independence of failure of O and the failure of mitigation mechanisms M1,....,Mk has been convincingly argued, for then the failure likelihood is just the multiplicative product of the individual failure likelihoods. But this scenario is prima facie quite unlikely. If O is a complex operation, and there is need for rollback if O does not complete, then it is quite likely that both O and its rollback mitigations will need at least some common access to some resource. The example of a database DB was used above, but this resource might be any one of a number of things, and will constitute a common-cause failure possibility for both O and its mitigants if it fails.

It is beyond the state of the art in software reliability engineering to be able to give general rules for assessment of Condition D independent of the very specific system architecture in which operation O takes place. This entails that, in general, a court hearing arguments about GR-robustness should expect disclosure of details about the system architecture and failure-expectation values derived from rates of observed failure enough to enable specialists to come to conclusions concerning Condition D. These details of system architecture will include much more detail than we have seen in some recent cases in which the robustness of software system architecture has been disclosed in legal proceedings.

A third concept of "robust"

The Horizon system was also claimed to be "robust" in a criminal court case, *Regina v Seema Misra* (Seema Misra, 2009) in which the "robust[ness]" of the Horizon system was advocated by prosecuting counsel, Mr. Tatford. The (claimed) robustness of the Horizon system was given as the main reason for concluding that discrepancies in system accounting were due to criminal activity by Ms. Misra. Ms. Misra was convicted, and sentenced to a term of imprisonment.

Mr. Tatford characterised the nature of "computer error" in the Horizon system:

> "if there was a computer error, we all use computers these days, if you have a computer error you are aware of it. …. in this post office, there were all sorts of printouts to tell you if things are going wrong and you are doing a stock take all the time. So you would have thought that if there was a computer error the defendant would have been aware of it and would have mentioned it in her interview"[10]

Counsel seems to be proposing a version of what Ladkin, Littlewood, Thimbleby and Thomas (LLTT 2020) call "the Tapper Condition", that "most computer error is either immediately detectable or results from error in the data entered into the machine" (Tapper 1991). Counsel is saying that "computer error" is "immediately detectable", and is not concerned in this case with error in input data. Let me call this the T-Tapper Condition ("T" for Tatford). Ladkin and others (LLTT 2020) have pointed out that the Tapper Condition is by no means universally satisfied by software-based computer systems. The judgement of Fraser J identified the occurrence in the Horizon system of "phantom transactions", supposed-transactions that were spontaneously generated by the Horizon system hardware and software rather than initiated by any system user. Such "computer error" was far from being "immediately detectable". It follows that the Horizon system did not satisfy the Tapper Condition.

The T-Tapper Condition is in fact stronger than the Tapper Condition. The Tapper Condition does not require that any computer error resulting from erroneous input data be immediately observable, but the T-Tapper Condition requires that any "problem" be immediately observable, including presumably "problems" arising from erroneous input data.[11] If the Horizon system does not satisfy the Tapper Condition, then a fortiori it does not satisfy the stronger T-Tapper Condition. It follows that prosecuting counsel's claim of "robust[ness]", in his sense, in Seema Misra 2009 was and is incorrect.

Mr. Tatford did not claim that Horizon is perfect: "no computer system in fact is going ever to be perfect. They all have problems from time to time."[12] He proffers no definition of what he means by "perfect", but does make the connection with not having "problems". This is close to the International Electrotechnical Vocabulary concept of "reliable": "ability to perform as required, without failure, for a given time interval, under given conditions" (IEV, Definition 192-01-24). Here we would identify an occurrence of "a computer problem"[13] (Tatford) with the system "not performing as required" (derived from Definition 192-01-24). Note that there are to be given constraints: the time interval, and other "conditions".

Mr. Tatford goes on to introduce the idea of robustness:

> "So it has got to be a pretty robust system and you will hear some evidence from an expert in the field as to the quality of the system. Nobody is saying it is perfect …. but the Crown say it is a robust system and that if there really was a computer problem the defendant would have been aware of it. That is the whole point because when you use a computer system you realise there is something wrong if not from the screen itself but from the printouts you are getting when you are doing the stock take."[14]

There seems to be an identification of "robust" with system "quality", as well as with satisfying the T-Tapper Condition. Further,

> "There was in 2006 a problem at a post office in Falkirk in Scotland called Calendar [sic] Square .. [witness for the prosecution] says that does not apply here. [Expert witness for the defence] says it does.

[10] Day 1 Monday 11 October 2010, 49C – D.

[11] "Erroneous input data" in the sense of the Tapper condition is a different concept from "fraudulent, i.e., intentionally incorrect, input data", which is what was under investigation in Regina v Seema Misra. Erroneous data in the sense of the Tapper condition are supposed to cause observable anomaly. Whereas if a transaction records £2 takings when the subpostmaster actually pocketed £200, in a case of fraud, the fraud is perpetrated most effectively when this transaction gives no indication at the system level of being erroneous, but "looks as if it is right".

[12] Day 1 Monday 11 October 2010, 48E – H.

[13] I do not know of any technical definition of "computer problem". The terms "fault", "failure", "error", and "defect" all occur, and are defined, in various technical literature, and the word "bug" is also commonly used, for instance in the judgement of Fraser J in Bates v Post Office Ltd.

[14] Day 1 Monday 11 October 2010, 49F – 50B.

> The Crown say it does not because that problem, as I say, computer problems should be obvious to the user."[15]

Again, the T-Tapper Condition is invoked.

The term "robust" was invoked by counsel for the defence in the cross-examination of Mr. Jenkins, expert witness for the prosecution:

> "Q. Can I ask you this, that do you have any experience of creating computer systems for banks?
>
> A. No.
>
> …….
>
> Q. Because you have got no experience of [systems for banks]?
>
> A. Not of doing systems for banks, no.
>
> Q. So you do not – cannot tell the court whether this Horizon system would be a failing system if you compared it to a retail bank?
>
> A. ……. I've not compared it with a retail bank.
>
> …….
>
> A. …….I'm saying it's been tested against the criteria that's been put on us by Post Office.
>
> Q. You see, what has happened, everyone has put trust in the Horizon system, that it is infallible, it is robust.
>
> ……..
>
> Q. And I am just asking for assistance and reassurance but you cannot give that, that it is compatible to a bank, can you?
>
> A. Because it is not seen as being a banking system."[16]

Here, defence counsel is suggesting that the prosecution has been arguing that the system is infallible (the prosecution has already noted they are not claiming the system is infallible – see above) and robust. It is not clear if counsel is suggesting that robustness is the same concept as infallibility. Since the prosecution has been arguing that the Horizon system is robust, but explicitly said that they are not claiming it is infallible, it would seem the prosecution intends the two concepts to be different. Note that the IEV definition of reliable implies "perform[ing] as required, without failure". This is surely close to what is meant by infallible. But infallibility seems to come without constraints on time or conditions.

The judge sums up the prosecution's contentions as follows:

> "……. the prosecution's case which as I have drafted currently, subject to your observations, is that there is ample evidence to establish that Horizon is a tried and tested system in use at thousands of post offices for several years and fundamentally robust and reliable"[17]

The judge is here bringing the concepts "robust" and "reliable" together. If the IEC definition of "reliable" is used, then it is close to "infallible". So both counsel for the defence and the judge are suggesting by association that "robust" is cognate with "reliable/infallible", but the prosecution has already said in its opening address that "no computer system is perfect", and we have argued that "perfect" in this sense is cognate with "reliable" in the IEV sense (without the constraints). So the properties which the prosecution are claiming for the Horizon system are not identical with the properties the judge or defence counsel are saying the prosecution is claiming.

A fourth notion of robustness

There is another notion of robustness which is used with some computer-based systems. It is worth while to note it here, because of its superficial similarity in some ways to the notion discussed in Seema Misra 2009, in order to point out that turns out not to be relevant to the cases discussed above. It is a notion used in statistics. The scientific-model-builder and statistician George Box is well known for his bon mot that all (scientific) models are wrong, but some models are useful. In 1979 he considered the notion of robustness in models, which even then were realised mostly in digital computer code in order for their values and parameters to be calculated.[18] Box defined robustness as follows.

> "Robustness may be defined as the property of a procedure which renders the answers it

[15] Day 1 Monday 11 October 2010, 54B – F.
[16] Day 4 Thursday 14 October 2010, 85E – 86F.
[17] Tuesday 19 October 2010, 10B-D.
[18] Nowadays it often seems as if the code itself is the model, rather like the Phillips Hydraulic Computer, the MONIAC (Wiki MONIAC n.d.).

> gives insensitive to departures, of a kind which occur in practice, from ideal assumptions. Since assumptions imply some kind of scientific model, I believe that it is necessary to look at the process of scientific modelling itself to understand the nature of and the need for robust procedures. Against such a view it might be urged that some useful robust procedures have been derived empirically without an explicitly stated model. However, an empirical procedure implies some unstated model and there is often great virtue in bringing into the open the kind of assumptions that lead to useful methods." (Box 1979)

We can see here some characteristics similar to the concept of robustness promoted by the prosecution in Seema Misra 2009. The answers the model (the code) gives are not necessarily perfect, but they are "insensitive to departures, of a kind which occur in practice, from ideal assumptions". The assumption is not literally exactly right, but the output produced under the assumption is somehow insensitive to this deviation from reality. The code (model) is not perfect (by assumption) but you can somehow trust what it outputs. This seems very like what the prosecution argued in Seema Misra 2009: the Horizon system is not perfect, but it does what you want it to do, therefore anomalies must be due to intentional subversion, i.e. fraud.

It is thus worth pointing out that this notion does not apply to transaction-processing systems. A transaction is an exact process, unlike an estimation of a quantity in a scientific model. When the transaction concerns an item exchanged for £31.34, then that exact amount £31.34 must be recorded and processed. It is not acceptable for such a system to handle such amounts as "£30 (give or take a bit)". Whereas the latter is what Box was talking about. Box's notion of robustness says that you get a usefully similar answer with input £30 (giving or taking a bit) as you do with £31.34. This might be all right for friends divvying up a restaurant bill for a group meal informally amongst themselves, but not so for a commercial transaction, where it would simply mean you would be recording a transaction incorrectly, which is in no way useful. Box's notion of robustness has no place in discussion concerning transaction processing systems.

Discussion

The notion of "robust" has a defined meaning in international standards for software engineering, and concerns how the system copes with invalid input data, namely it refers to the degree to which a system or component can function correctly in the presence of invalid inputs or stressful environmental conditions. Software engineering techniques such as strong data typing have existed for over half a century which can help to ensure robustness.

A different sense of "robust" was introduced in Bates 2019 by counsel for the defence, who gave a complex definition which we have called "GR-robust", which at least one of the phenomena confirmed in Fraser J's judgement, phantom transactions, did not satisfy. It follows from this judgement that the Horizon system was not GR-robust. We have indicated what kind of evidence would be required to establish that a software-based system is GR robust; it is quite extensive and requires detail of the system architecture (and evidence that that architecture was in fact implemented as intended).

The notion of the Horizon system being "robust" was significant to the arguments of prosecution and defence in Seema Misra 2009. The notion of "robust" was, however, not defined explicitly by either counsel or the judge. We have argued that it was interpreted by the prosecution as semantically equivalent to the T-Tapper Condition, and by the judge and defence counsel as semantically equivalent to the IEV definition of "reliable" (without the time and condition constraints). The judgement of Fraser J in Bates 2019 established that Horizon was not reliable (over the entire period of its use, under the conditions existing throughout that period), where "reliable" is meant in the IEC sense. It follows from the judgement of Fraser J that Horizon also did not satisfy the Tapper Condition (observed in (LLTT 2020)), and thus that it did not satisfy the T-Tapper Condition, which is logically stronger than the (plain) Tapper Condition. And, thus, that the prosecution's claim in Seema Misra 2009 also fails: Horizon was not robust in the sense of fulfilling the T-Tapper Condition.

Peter Bernard Ladkin is a systems-safety specialist with a background in software dependability and logic. His causal accident analysis method Why-Because Analysis (WBA) is used by some 11,000 engineers worldwide. He taught at Bielefeld University and is CEO of tech-transfer companies Causalis Limited and Causalis Ingenieurgesellschaft mbH.

References

(AvLaRaLa 2004) Algirdas Avizienis, Jean-Claude Laprie, Brian Randell and Carl Landwehr, Basic Concepts and Taxonomy of Dependable and Secure Computing, IEEE Trans. Dep. Sec. Comp. 1(1), Jan-Mar 2004.

(Bates 2019) *Bates v the Post Office Ltd (No 6: Horizon Issues)* [2019] EWHC 3408 (QB), available at http://www.bailii.org/ew/cases/EWHC/QB/2019/3408.html

(Seema Misra 2009) *Regina v Seema Misra*, T20090070, In the Crown Court at Guilford, Trial dates: 11, 12, 13, 14, 15, 18, 19, 20, 21 October and 11 November 2010, His Honour Judge N. A. Stewart and a jury, 12 *Digital Evidence and Electronic Signature Law Review* (2015) Introduction, 44 – 55; Documents Supplement, available at https://journals.sas.ac.uk/deeslr/article/view/2217

(Box 1979) George E. P. Box, Robustness in the Strategy of Scientific Model Building, MRC Technical Summary Report #1954, Mathematics Research Center, University of Wisconsin-Madison, May 1979. Available from https://apps.dtic.mil/dtic/tr/fulltext/u2/a070213.pdf Also published in Robustness in Statistics, ed. Robert L. Launer and Graham N. Wilkinson, Academic Press, 1979. Available through https://www.sciencedirect.com/science/article/pii/B9780124381506500182

(CuBlTh 2015) Paul Curzon, Ann Blandford, Harold Thimbleby and Anna Louise Cox, Safer Interactive Medical Device Design: Insights from the CHI+MED Project, Security and Safety 3(9). Available at https://www.researchgate.net/publication/301452018_Safer_Interactive_Medical_Device_Design_Insights_from_the_CHIMED_Project

(ED-12C) EUROCAE, ED-12C - Software considerations in airborne systems and equipment certification. EUROCAE, 1999. This document is also published by a North-American consortium, RTCA, under the designation DO-178C. It is known in the industry both as DO-178C and as ED- 12C.

(IEC 24765) International Organisation for Standardization/International Electrotechnical Commission/Institute of Electrical and Electronics Engineers, ISO/IEC/IEEE International Standard 24765-2017 Systems and software engineering – vocabulary, ISO/IEC/IEEE 2017. Available through the ISO On-line Browsing Platform at https://www.iso.org/obp/ui/#iso:std:iso-iec-ieee:24765:ed-2:v1:en

(IEEE 610.12) Institute of Electrical and Electronics Engineers, IEEE Std. 610.12-1990 IEEE Standard Glossary of Software Engineering Terminology, IEEE, 1990.

(IEC Glossary) International Electrotechnical Commission, Glossary. On-line WWW resource, available at http://std.iec.ch/glossary

(IEV) International Electrotechnical Commission, IEC 60050, International Electrotechnical Vocabulary. International Electrotechnical Vocabulary, no date. Available at http://www.electropedia.org

(LadTho 2020) Peter Bernard Ladkin and Martyn Thomas, Software Quality, Its Nature, and the Cultures of Building It, preprint 2018/2020. Available from https://www.researchgate.net/publication/339697246_Software_Quality_Its_Nature_and_the_Cultures_of_Building_It

(Ladkin 2019) Peter Bernard Ladkin, Semantic Analysis: SemAn, preprint 2019-08-14. Available from the author.

(LLTT 2020) Peter Bernard Ladkin, Bev Littlewood, Harold Thimbleby and Martyn Thomas CBE, The Law

Commission presumption concerning the dependability of computer evidence. Digital Evidence and Electronic Signature Law Review 17, 2020.

(Tapper 1991) Colin Tapper, Discovery in Modern Times: A Voyage around the Common Law World, 67 Chicago-Kent Law Review 217, 248, 1991.

(Wiki MONIAC n.d.) Wikipedia, MONIAC, no date. Available at https://en.wikipedia.org/wiki/MONIAC