A proposal to achieve professional software engineering in scientific research

HAROLD THIMBLEBY, See Change Fellow in Digital Health, Wales

Background: Computer code underpins modern science, and at the present time has a crucial role in leading our response to the COVID-19 pandemic. While models are routinely criticised for their assumptions, the algorithms and the quality of code implementing them often avoid scrutiny and, hence, scientific conclusions cannot be rigorously justified.

Problem: Assumptions in programs are hard to scrutinise as they are rarely explicit in published work. In addition, both algorithms and code have bugs, effectively unknown assumptions that have unwanted effects, undermining the rigor of claims. Code is fallible, so any model interpretation that relies on code is therefore fallible, and if the code is not published with adequate documentation, the code cannot be scrutinised. In turn, the scientific claims cannot be properly scrutinised.

Solutions: Code can be made *much* more reliable using software engineering good practice. Three specific solutions are proposed. First, professional software engineers can help and should be involved in critical research. Secondly, "Software Engineering Boards" (supplementing and analogous to Ethics or Institutional Review Boards) must be instigated and used. Thirdly, code, when used, must be considered an intrinsic part of any publication, and therefore must be formally reviewed by competent software engineers.

This paper's Supplemental Material includes a summary of professional software engineering best practice, particularly as applied to scientific research and publication.

CCS Concepts: • **Applied computing → Life and medical sciences**; **Physical sciences and engineering**; **Document management and text processing**; • **Software and its engineering → Software development methods**; • **Social and professional topics**;

Additional Key Words and Phrases: Scientific modelling; Dependability; Publishing

ACM Reference Format:
Harold Thimbleby. 2021. A proposal to achieve professional software engineering in scientific research. *Submitted ACM Trans. Softw. Eng. Methodol.* 1, 1 (April 2021), 15 pages.

"Criticism is the mother of methodology."
Robert P Abelson, *Statistics as Principled Argument*, 1995

1 INTRODUCTION

In good science it is uncontentious to report statistics fully and carefully. For example, a statistical claim might be written as follows:

> "… Bonferroni adjusted estimated mean difference between intervention-arms at 8-weeks 2.52 (95% CI 0.78, 4.27, p = 0.0009). Between group effect size d = 0.55 (95% CI 0.32, 0.77)." [32]

This is the standard, widely accepted form of writing — confidence intervals, p levels, and so on — that is used to present statistics so claims can be seen to be complete, easy to interpret and to scrutinise. Statistical claims need to be properly accountable and documented in a clear way: Speigelhalter [40] says statistical information needs to be accessible, intelligible, assessable, and usable; he also suggests probing questions to help assess statistical quality (see Supplemental Material section 4), as results should not be uncritically accepted just because they are claimed.

Author's address: Harold Thimbleby, See Change Fellow in Digital Health, Wales, harold@thimbleby.net.

1049-331X/2021/4-ART
URL https://doi.org/\@acmDOI

The skill and effort required to do statistics so it can be communicated clearly and correctly, as above, is not to be taken for granted. Scientists work closely with competent, often specialist, statisticians, who engage with the research from experiment design through to analysis and publication. It is assumed that statistics will be peer reviewed, and that review will often lead to improvement. Moreover, we accept that statistics is a professional science and is itself a subject of research and continual improvement. Among other implications of the depth and progress of the field of statistics, undergraduate statistics options for general scientists are insufficient training for rigorous work in science — their main value, perhaps, is to help collaborate with specialist statisticians.

Except in the most trivial of cases, all numbers and graphs, and the statistics themselves in scientific research, will have been generated by computer. Indeed, computers are now very widely used, not just to calculate statistics, but to run the models (the term is defined below) and sampling or sensors that generate the data that is analysed. Furthermore, the data — including the databases and bibliographic sources — and code to analyse it is stored on computers, raising basic questions of formats, backup, cyber-vulnerability, version control, integrity checking (e.g., managing human error) and auditing. All are non-trivial concerns.

Failure to properly document and explain computer code undermines the scientific value of the models and the results they generate, in the same way as failure to properly articulate statistics undermines the value of scientific claims. Indeed, as few papers use code that is as well-understood and as well-researched as standard statistical procedures (such as Bonferroni corrections), the scientific problems of poorly reported code are worse.

Results claimed, however nicely they may be expressed, are no more reliable than the code and data that was used to generate them. A common oversight is to present a mathematical model, such as a set of differential equations, but omit how that model is transformed into code that generates reliable results.

Just as in statistics, code and results from code, need to be discussed and presented in a way that ground belief in claims derived from using them. Specifically, code must be developed in a sufficiently professional and rigorous way that is able to support clear presentation and scrutiny (developing justifiably reliable code is the concern of software engineering, which is discussed further below in this paper and in this paper's Supplemental Material).

To make critical claims, to inform public policy or to support further research, models need to be run under varying assumptions [52]. Being able to understand (at least in principle) the exact code used in implementing a model is critical to having confidence in the results that rely on it. Unfortunately code is rarely considered a substantial part of the science to which it contributes.

In contrast, we would not believe a statistical result that was obtained from some *ad hoc* analysis with a new fangled method devised just for one project — instead, we demand statistics that is recognisable, even traditional, so we are assured we understand what has been done and how reliable results were obtained. Importantly, with statistics we understand, at least in principle, how it works — or, more precisely, how results *should* have been obtained and why or to what extent we can depend on them.

This paper will show that the concerns raised above are serious. The quality of much science is undermined because the code it relies on is rarely of adequate quality *for the uses to which it is put*, particularly when it influences health or public health policies. Furthermore, code is rarely published in any useful form or professionally scrutinised, and therefore does not contribute to furthering science, for instance through reproduction. This paper will explore the extent of these problems and reasons why they persist. The paper then suggests some ways forward. The Supplemental Material is an integral part of the suggested solutions. In particular, the Supplemental

Material section 4 summarises Speigelhater's statistics probes and draws explicit analogies for critical assessment of code.

2 THE ROLE OF CODE IN SCIENCE AND SCIENTIFIC PUBLICATION

For the purposes of this paper, models map theory and parameters to describe phenomena, typically to make predictions or to test and refine the models. With the possible exception of theoretical research, all but the simplest models require computers to evaluate; indeed even theoretical mathematics is now routinely performed by computer.

Whereas the mathematical form of a model may be concise and readily explained, even a basic computational representation of a model can easily run to thousands of lines of code (and its parameters — its data — may also be extensive). The chance that a thousand lines of hand-written code is error free is negligible, and therefore good practice demands that checks and constraints must be applied to improve its reliability. This issue is the concern of software engineering, which is discussed throughout this paper.

While scientific research may rely on relatively easily scrutinised mathematical models, or models that seem in principle easy to mathematise, the models that are run on computers to obtain the results published are sometimes not disclosed, and even when they are (at least in all cases reviewed here) they are inscrutable — and therefore the models are very likely to be unreliable *in principle*. If code is not well documented, this is not only a problem for reviewers and scientists reading the research to understand the intention of the code, but it also causes problems for the original researchers themselves: how can they understand its thinking well enough (e.g., a few weeks or months later) to maintain it correctly if has not been clearly documented? Without documentation, including a reasoned case to assure that the approach taken is sound [12], how do researchers, let alone reviewers, know exactly what they are doing?

The problem might be expressed like this. Even if a scientist provides access to some code, say "`x = k*exp(x)`" let alone access to many more lines — and programs are typically tens of thousands of lines long, and include many libraries of support code — there can be *no* concept of the code correctness *unless* there is also an explicitly stated relation between the code and the mathematical details of the intention.

Without an explicit link to the relevant mathematics (depending on the claims), it is impossible to reason whether the code is correct, and in turn it is impossible to scientifically scrutinise results obtained from using the code. Not providing code, providing partial code, or providing code without the associated reasoning is analogous to claiming "the results are significant" without any discussion of the relevant statistical methods and details that justify making such a claim. If such an unjustified pseudo-statistical claim was made in a scientific paper, a reviewer would be justified in wondering whether a competent experiment had even been performed — it would be generous to ask the author to provide the missing details so the paper could be better reviewed on resubmission. The same applies to code and its methods and reasoning.

Clearly, like statistics, programming (coding) can be done poorly and reported poorly, or it can be done well and reported well — and any mix between these extremes. The question is whether it matters, and if so, *when* it matters and, when it does, *what* can be done to *appropriately* help improve the quality of code (and discussions about the code) in scientific work.

This paper explores the role of code in scientific research. Initially focusing on examples from pandemic modelling (because of its relevance to serious matters of public health) this paper shows that scientific culture, including editorial processes, have not adapted to the increasingly dominant role of code and managing the risks of its fallibility. The paper then suggests how to improve, including building mutual engagement across the science and software engineering communities.

3 BUGS, CODE AND PROGRAMMING

Critiques of data and model assumptions are very common [34, 54] but program code is rarely mentioned. Yet data and program are formally equivalent (see further discussion in Supplemental Material, section 3). Program code has as great an affect on results as the data. Code is harder to scrutinise, which means errors in code can have more subtle effects on results.

It should be noted that almost all code contains "magic numbers" — that is, data masquerading as code. This common practice ensures that published data is very rarely all of the data because it omits magic numbers. This emphasises the need for repositories to require the inclusion of code so all data is actually available. This is another reason why journal data policies should be applied to code too, so all data embedded in code is covered by the policies (see the longer discussion in the Supplemental Material, section 3).

Bugs can be understood as discrepancies between what code is intended to do and what it actually does. Many bugs cause erroneous results, but bugs may be "fail safe" by causing a program to crash so no incorrect result can be delivered. Better, contracts and assertions are essential defensive programming technique that block compilation or block execution with incorrect results; they turn bugs into safe termination. None of the code examined for this paper includes either.

If code is not documented it cannot be clear what it is intended to do, so it is not even possible to detect and eliminate bugs. Indeed, even with good documentation, *intentional bugs* will remain, that is, code that correctly implements the wrong things. For instance, in numerical modelling, using an inappropriate method can introduce errors that are not "bugs" in the sense of incorrectly implementing what was wanted (e.g., ill-conditioning), but are bugs in the sense of producing incorrect results — that is, what was wanted was naïve. Similarly, misuse of random numbers (e.g., using standard libraries without testing them) is a common cause of bugs [22].

Following the Dunning-Kruger Effect [23, 28], unqualified programmers over-estimate their programming skills because they do not have the skills to recognise their lack of knowledge. Dunning and Kruger say,

> "People usually choose what they think is the most reasonable and optimal option [...] The failure to recognise that one has performed poorly will instead leave one to assume that one has performed well; as a result, the incompetent will tend to grossly overestimate their skills and abilities. [...] Not only do these people reach erroneous conclusions and make unfortunate choices, but their incompetence robs them of the metacognitive ability to realise it."

Unlike many skills (skating, brain surgery, ...) programming, typical of much engineering, is one where errors can go unnoticed for long periods of time — things seem to work nicely right up to the moment they fail. The worse programmers are, the more trivial bugs they tend to make, but trivial bugs are easy to find so, ironically, being a poor programmer *increases* one's self-assessment because debugging seems very productive. It is easy for poor programmers and their associates to believe they are better than they actually are.

It sounds harsh to call programmers incompetent, but challenged with the complexity of programs and the complexity of the domains programs are applied in, we are all incompetent and succumb to the limitations of our cognitive resources, creating blindspots in our thinking [44]. We *all* make mistakes we are unaware of. If we do not have the benefit of professional qualifications that have assessed us objectively, programmers therefore generally have a higher opinion of our own competence than is justified.

Everyone (including the author of the present paper) is subject to Human Factors: for instance, the standard cognitive bias of confirmation bias encourages us to look for bugs when code fails to do what is expected and then debug it to produce better results, but if code generates expected results not to bother to debug it further. This of course tends to make code increasingly conform to prior expectations, whether or not those expectations are scientifically justified. Typically, there was no prior specification of the code, so the code must be right, especially after all the debugging to make it "correct"! Thus coding routinely suffers from HARKing, a methodological trap widely recognised in statistics — hypothesising after the results are known [20].

Computers themselves are also a part of the problem. Naïvely modifying a program (as may occur during debugging) can make it more complex, and hence less scrutable. In theory programs can be written so that it is not possible to determine what they do or how they do it (whether deliberate obfuscation or accidentally), except by running them, if indeed it is possible to exactly reproduce the necessary context to do so [47]. The point is, introducing bugs should be avoided so far as possible in the first place, and programs should routinely have assertions and other methods to detect those bugs that are introduced (see this paper's Supplemental Material for more discussion of standard programming methodologies).

4 STATE OF THE ART IN PANDEMIC MODELLING

A review of epidemic modelling [16] says, "we use the words 'computational modelling' loosely," and then, curiously, the review discusses exclusively mathematical modelling, implying that for the authors, and for the peer reviewers, there is no conscious role for code or computation as such. It appears that the new insights, advances, rigour, and problems that computers bring to research are not considered relevant.

A systematic review [54] of published COVID models for individual diagnosis and prognosis in clinical care, including apps and online tools, noted the common failure to follow standard TRIPOD guidelines [27]. (TRIPOD guidelines ignore code completely, let alone its quality.) The review [54] ignored the mapping from models to their implementation, yet if code is unreliable, the model *cannot* be reliably used, and cannot be reliably interpreted. It should be noted that flowcharts, which the review did consider, are programs intended for direct human use. Flowcharts, too, must be designed as carefully as code, for exactly the same reason: it is hard to program reliably.

A high-profile 2020 COVID-19 model [3, 11] uses a modified 2005 computer program [9, 10] for H5N1 in Thailand, which did not model air travel or other factors required for later western COVID-19 modelling. The 2020 model forms part of a series of papers [9–11] none of which provide details of their code. A co-author disclosed [7] that the code is thousands of lines long and is undocumented C code. As Ferguson, the code author, noted in an interview,

> "For me the code is not a mess, but it's all in my head, completely undocumented. Nobody would be able to use it ..." [25]

This admission is tantamount to saying that the published scientific findings are not reproducible.

This is problematic, especially as the code would have required many non-trivial modifications to update it for COVID-19 with different assumptions; moreover, the code would have had to have been updated very rapidly in response to the urgent COVID-19 crisis. If this C code had been made available for review, the reviewers would not have known how to evaluate it without the relevant documentation. It is, in fact, hard to imagine how a large undocumented program could have been repeatedly modified over fifteen years without becoming incoherent. If code is undocumented, there would be an understandable temptation to modify it arbitrarily to get desired results; worse, without documentation and proper commenting, it is methodologically impossible to distinguish legitimate attempts at debugging from merely fudging the results. In contrast, if code is properly documented, the documentation defines the original intentions (including formally using mathematics to do so), and therefore any modifications will need to be justified and explained — or the theory revised.

The programming language C which was used is not a dependable language; to develop reliable code in C requires professional tools and skills. Moreover, C code is not portable, which limits making it available for other scientists to use safely (C notoriously gets different results on different compliers, libraries, and hardware). The Supplemental Material discusses these issues further.

Ferguson, author of the code, says of the criticisms of his code, "However, none of the criticisms of the code affects the mathematics or science of the simulation" [6]. Indeed. The problem the current paper is raising is that the epidemiology may be fine, but if it is not mapped into code that correctly implements the models, then the program's output should not be relied on without independent evidence. In science, this is the normal requirement of *reproducibility*.

The code in [3, 11] has been reproduced [6, 35], but this "reproduction" has merely confirmed the code can be run again and produce comparable results (compared, apparently in an Excel spreadsheet!). That is hardly surprising, regardless of the quality of the code. Instead, if reproducibility is to be a useful scientific criterion, an *independently* developed model needs to produce the same results — this is called N version programming, which is a standard software engineering practice in critical areas [15], like public health ought to be.

Because of the recognised importance of the paper, a project has been started to document its code [8]. Documenting the code now may describe what it does, *including* its bugs, but it is unlikely to explain what it was intended to have done. If nothing else, as the code is documented, bugs will be found, which will then be fixed (refactoring), and so the belatedly-documented code will not be the code that was used in the published models. It is well-known that documenting code helps improve it, so it is surprising to find an undocumented model being used in the first place. The revised code has now been published, and it has been heavily criticised (e.g., [31]), supporting the concerns expressed in the present paper.

Some epidemiology papers (e.g., [57]) publish models in pseudo-code, a simplified form of programming. Pseudo-code looks deceptively like real code that might be copied to try to reproduce it, but as pseudo-code introduces invisible simplifications. Pseudo-code, properly used, can give a helpful impression of the overall approach of an algorithm, certainly, but pseudo-code alone is not a surrogate for code: using it is arguably even worse than not publishing code at all. Pseudo-code is not precise enough to help anyone scrutinise a model; copying pseudo-code introduces avoidable bugs. An extensive criticism of pseudo-code, and discussion of tools for reliable publication of code can be found elsewhere [49]. The Supplemental Material provides further discussion of reproducibility.

5 BEYOND EPIDEMIOLOGY

Code needs to be carefully documented and explained because all code has tacit assumptions, bugs and cybersecurity vulnerabilities [39] that, if not articulated, can affect results in unknown ways that may undermine any claims. People reading the code will not know how to obtain good, let alone better results, because they do not know exactly what was intended in the first place. The problem is analogous to the problem of failing to elaborate statistical claims properly: failure to do so suggests that the claims may have unknown limitations or even downright flaws.

Even good quality code has, on average, a defect every 100 lines — and such a low rate is only achieved by experienced industrial software developers [24]. World-class software can attain maybe 1 bug per 1,000 lines of code. Code developed for experimental research purposes will have higher rates of bugs than professional industrial software, because the code is less well-defined and evolves as the researchers gain new insights into their models. In addition, and perhaps more widely recognised, code — especially but not exclusively mathematical code — can be subject to numerical errors [13]. It is therefore inevitable that typical modelling code has many bugs (reference [15] is a slightly-dated but very insightful discussion). Such bugs undermine confidence in model results.

Only if there is access to the *actual* code and data (in the specific version that was used for preparing the paper) does anyone know what researchers have done, but merely making code available (for instance, providing it in their Supplemental Material with papers, putting it in repositories, or using open source) is not sufficient. It is noteworthy that some papers disclosed that they had access to special hardware.

Some COVID-19 papers (e.g., [21]) make unfinished, incomplete code available. While some (e.g., [21, 50]) do make what they call "documented" code available, they provide no more than superficial comments: this is *not* documentation as properly understood. Such comments do not explain code, explain contracts, nor explain algorithms. Some readers of the present paper may not recognise these technical software terms; contracts, for instance, originated in work in the 1960s [17], and are now well-established practice in reliable programming (see the Supplemental Material for a checklist of many relevant, professional software engineering concepts and resources).

If full code is made available, it may be technically "reproducible," but the scientific point is to be able to understand and challenge, potentially refute, the findings; to do that, much more is required than merely being able to run the code [29, 36].

Even if a computer can run it, badly-written code (as found in *all* the research reviewed in the present paper, and indeed in computer science research itself [46]) is inscrutable, even if its original programmers think otherwise. Only if there is access to *adequate* documentation can anyone know what the researchers *intended* to do. Without all three (code, data, adequate documentation), there are dangers that a paper simplifies or exaggerates the results reported, and that omissions, bugs and errors in the code or data, generally unnoticed by the paper's authors and reviewers, will have affected the results they report [49].

Making outline code (including pseudo-code) available without proper documentation and without discussing its limitations is unethical: it encourages others to reproduce and build on poor work.

5.1 A pilot survey of the wider peer-reviewed research relying on code

The problems of unreliable code are not limited to COVID-19 modelling papers, which, understandably, were perhaps rushed into publication. For instance, a 2009 paper reporting a model of H5N1 pandemic mitigation strategies [33] provides no details of its code. Its Supplemental Material, which might have provided code, no longer exists.

Almost all scientific papers rely on generic computer code (for statistics or for plotting graphs, and so on) but what is of interest here is whether, and, if so, to what extent, code developed *specifically* as part of or to support a research contribution can be understood by colleagues and scientifically scrutinised.

A sample of 32 recent papers, not including the motivating papers discussed above, and covering a broad range of science were selected from the online editions (as of July 2020) of *Lancet Digital Health* ($N = 6$), *Nature Digital Medicine* ($N = 12$) and *Royal Society Open Science* ($N = 14$). The survey sampled the recent work of 264 published authors, perhaps directly involving around 392 leading scientists in all (authors, editors, reviewers — ignoring multiple roles). The survey was not intended to be a formal, representative sample of all scientific research in general, but it is hoped it will be sufficient to dispel the possibility that the issues described above in earlier sections of this paper are isolated practice, perhaps an idiosyncrasy of a few authors or of a particular field or perhaps due to a chance selection bias.

Number of papers sampled relying on code	32	100%	
Access to code			
Have some or all code available	12	38%	
Some or all code in principle available on request	8	25%	
No code available	12	38%	
Evidence of basic good software engineering practice			
Evidence program designed rigorously	0	0%	
Evidence source code properly tested	0	0%	
Evidence of makefile or equivalent	0	0%	
Other methods, e.g., independent coding methods	1	3%	
Documentation and comments			
Good code documentation and comments	2	6%	
Basic comments	3	9%	
No or only trivial comments (e.g., copyright)	29	91%	
Repository use			
Code repository (e.g., GitHub) — 1 was empty	10	31%	
Data repository (e.g., Dryad or GitHub)	9	28%	
Adherence to journal code policy (if any)			
Papers published in journals with code policies	26	81%	
Clear breaches of code policy	11	42%	(N = 26)

Table 1: Summary of the exploratory survey. Note that code quality was assessed (by reading it) for human use — due to the paper authors' complex and/or narrative interpretation of data in most papers, code, data and hardware/operating system dependencies, no assessment was made whether the code provided was sufficient to reproduce a paper's specific claims. Details of the survey methodology and the breakdown of measures into individual papers can be found in the Supplemental Material.

It is important that this survey is not taken as an evaluation, whether praise or criticism, of any specific paper: any measurement is subject to error, and individual paper assessments are inevitably noisy. Instead, by evaluating a set of diverse papers, the noise will tend to average out. Mean scores, as summarised in table 1, are more reliable measurements.

The methodology, data and analysis are provided in the Supplemental Material (which is also available from a repository — see details at the end of this paper).

Nature Digital Medicine and *Royal Society Open Science* have clear data and code policies (see Supplemental Material section 5), but actual practice falls short: 11 out of the 26 papers (42%) published in them and sampled in the survey manifestly breach their code policies. In contrast, *Lancet Digital Health*, despite substantial data policies, has no code policy at all to breach. The implication of these results is that the fields, and the editorial expertise of leading journals, misunderstand and dismiss code policies, or they are technically unable to assess them — and, if so, they also fail to say they are unable to assess them. This lack of expertise is consistent with the limited awareness of software best practice manifest in the published papers themselves.

Code repositories were used by 10 papers (31%), though one paper in the survey claimed to have code on GitHub but there was no code in the repository (at the time of doing the review or when double-checked six months later): in other words, the repository had never been used and the code had very probably not been reviewed.[*] This is a pity because repositories like GitHub provide help and targeted hints like "No description, website, or topics provided [...] no releases published."

Overall, there was no evidence that any code had been developed carefully, let alone by using recognised professional software engineering methods. In particular, no papers in the survey provide any claims or evidence of effective testing, for instance with evidence that tests were run on clean builds. While it may sound unrealistic to ask for evidence on software quality in a paper written for another field of science, the need is no less than the need for standard levels of rigor in statistics reporting, as discussed in the opening of this paper.

Data repositories (the Dryad Digital Repository, Figshare or similar) were used by 9 papers to provide structured access to their data. Unlike GitHub, which is a general purpose repository, Dryad has scientifically-informed guidelines on handling data, and all papers that used Dryad provided more than just their raw data — they provided a little, sometimes substantial, documentation for their data.

The findings from the survey are summarised in table 1, and discussed at greater length in the Supplemental Material.

6 DISCUSSION

Effective access to code is not routine. Using structured repositories that provide suggestions for and which encourage good practice (such as Dryad and GitHub), and requiring their use, would be a lever to improve the quality and value of code and documentation in published papers. The evidence suggests that, generally, some but not all manually developed code is uploaded to a repository just before submitting the paper in order to "go through the motions." In the surveyed papers there is no clear evidence (over the survey period) that any published code was maintained using the repositories.

The Supplemental Material provides a summary of the analysis and full citations for all papers in the sample.

6.1 Possible further work

Further work to extend the scope of the survey and challenge beyond the basic requirements of the present paper will be worthwhile. For example:

(1) The journal *The Lancet* published and then subsequently retracted a paper on using hydroxychloroquine as a treatment for COVID [26]. The paper was found to rely on fraudulent data [37, 38]. *The Lancet* subsequently tightened its data policies [41], for instance to require that more than one author must have directly accessed and verified the data reported in the manuscript. Curiously, the original (now retracted) paper declares

> "... all authors participated in critical revision of the manuscript for important intellectual content. MRM and ANP supervised the study. All authors approved the final manuscript and were responsible for the decision to submit for publication."

which seems to suggest that several original authors of the paper would have been happy to make the new declarations — and, of course, if there is fraud (as was established in this case) it seems likely that authors who make the new declarations of accessing and verifying data are unlikely to make reliable declarations.

The Lancet still has no code publication policy (see the Supplemental Material), and for more than one author to have "direct access" to the data they are very likely to access the data through the same code. If the code is faulty or fraudulent, an additional author's confirmation of the data is insufficient, and there is at least as much reason for code to be

[*]GitHub records show that it had not been deleted after paper submission.

fraudulent (not least because code is much harder to scrutinise than data). Code needs more than one author to check it, and ideally reviewers independent of the authors so they do not share the same assumptions and systems (for instance shared libraries, let alone potential collusion in fraud).

(2) In 2020 the *Journal of Vascular Surgery* published a controversial research paper [14], which had to be retracted on ethical grounds [4, 42]: it was a naïve study and the editorial process was unaware of digital norms. Notably, the paper fails to provide access to its anonymised data (with or without qualification), and fails to define the data anonymisation algorithm, and also fails to even mention the code that it developed and used to perform its study. The journal's data policy is itself very weak (the authors "should consider" including a footnote to offer limited access to the data) and, despite basic statistics policies, it has no policy at all for code (see Supplemental Material section 5). Ironically, the retracted article [14] is still online (as of August 2020) with no reference to any editorial statement to the effect that it has been retracted, despite this being trivial — and necessary — to achieve in the widely-accessed online medium.

(3) Medical research often aims to establish a formula to define a clinical parameter (such as body mass index, BMI) or to specify an optimal drug dose or other intervention for treatment. These formulas, for which there is conventional clinical evidence, are often used as the basis for computer code that provides advice or even directly controls interventions. Unfortunately a simple formula as may be published in a medical paper is *never* sufficient to specify code to implement it safely. For example, clinical papers do not need to evaluate or manage user error when operating apps, and therefore the statistical results of the research will be idealistic compared to the outcomes using an app under real conditions — which is what the clinical research is supposedly for. A widespread bug (and its fix) that is often overlooked is discussed in [48]; the paper includes an example of a popular clinical calculator (based on published clinical research) that calculated nonsense, and potentially dangerous, results. The paper [55] summarises evidence that such bugs, ignored by the clinical research literature, are commonplace in medical systems and devices.

7 CALL TO ACTION

Computer programs are the laboratories of modern scientists, and should be used with a comparable level of care that virologists use in their laboratories — lab books and all [36] — and for the same reasons: computer bugs accidentally cultured in one laboratory can infect research and policy worldwide. Given the urgency of rigorously understanding COVID-19, any epidemic for that matter, it is essential that epidemiologists engage professional software engineers to help develop reliable laboratory methodologies. For instance, code lab books can be generated and signed easily.

Software used for informing public health policy, medical research or other medical applications is *critical software*. Professional critical software development, as used in aviation and the nuclear power industry, is (put briefly) based on *correct by construction*: [53] effectively, design it right first time, supported by numerous rigorous techniques, such as formal methods, to manage error. (See extensive discussion in this paper's Supplemental Material.) Not coincidentally, these are *exactly* the right methods to ensure code is both dependable and scrutable. Conversely, not following these practices undermines the rigour of the science.

An analogous situation arises in ethics. There are some sorts of research that are ethically unacceptable, but few people have the objectivity and ethical expertise to make sound ethical judgements, particularly when it comes to assessing their own work. Misuse of data, exploiting vulnerable people, and not obtaining informed consent are typical ethical problems. National funders, and others, therefore require Ethics Boards to formally review ethical quality. Medical journals will not publish research that has not undergone appropriate ethical review.

Analogously, and supplementing Ethics Boards, Software Engineering Boards would authorise as well as provide advice to guide the implementation of high-quality software engineering. Just as journals require conflicts of interest statements, data availability statements, and ethics board clearance, we must move to epidemic modelling papers — and in due course, all scientific papers — being required to include Software Engineering Board clearance statements as appropriate. Software Engineers have a code of ethics that applies to *their* work in epidemic modelling [2].

The present paper did not explore data, because in almost all cases the code and data were explained so poorly and archived so haphazardly it would be impossible to know whether the author's intentions were being followed.[†] Some journals have policies that code is available (see the Supplemental Material), but they should require that code is not just available in principle but *actually works* on the relevant data. Ideally, the authors should test a clean deployed build of their code and save the results. Presumably a paper's authors must have run their code successfully on some data (if any, but see section 3 in the Supplemental Material) at least once, so preparing the code and data in a way that is reproducible should be a routine and uncontentious part of the rigorous development of code underpinning *any* scientific claims. This requirement is no more unreasonable than requesting good statistics, as discussed in the opening of the paper. And the solution is the same: relevant experts — whether statisticians or software engineers — need to be routinely available and engaged with the science. SEBs would be a straight forward way of helping achieve this.

There need to be many Software Engineering Boards (SEBs) to ensure convenient access and oversight, potentially at least one per university. Active, professional software engineers should be on these SEBs; this is not a job for people who are not qualified and experienced in the area or who are not actively connected with the true state of the art. There are many high-quality software companies (especially those in safety-critical areas like aviation and nuclear power) who would be willing and competent to help.

Open Source generally improves the quality of software. SEBs will take account of the fact that open source software enables contributors to be located anywhere, typically without a formal contractual relationship with the leading researchers. Where appropriate, then, SEBs might require *local* version control, unit testing, static analysis and other quality control methods for which the lead scientist and software engineer remain responsible, and may even need to sign off (and certainly be signed off by the SEB). Software engineering publishers are already developing rigorous badging initiatives to indicate the level of formal review of the quality of software for use in peer reviewed publications [1]. See this paper's Supplemental Materialfor further suggestions.

A potential argument against SEBs is that they may become onerous, onerous to run and onerous to comply with their requirements. A more mature view is that SEBs need their processes to be adaptable and proportionate. If software being developed is of low risk, then less stringent engineering is required than if the software could cause frequent and critical outcomes, say in their impact on public health policy for a nation. Hence SEBs processes are likely to follow a risk analysis, perhaps starting with a simple checklist. There are standard ways to do this, such as following IEC 61508:2010 [19, 30] or similar. Following a risk analysis (based on safety assurance cases, controlled documents and so on, if appropriate to the domain), the Board would focus scrutiny where it is beneficial without obstructing routine science.

A professional organisation, such as the UK Royal Academy of Engineering ideally working in collaboration with other national international bodies such as IFIP, should be asked to develop

[†]For the present paper, all the code, data, analysis and documents are available for download in a single zip file.

and support a framework for SEBs. SEBs could be quickly established to provide direct access to mature software engineering expertise for both researchers and for journals seeking competent peer reviewers. In addition, particularly during a pandemic, SEBs would provide direct access to their expertise for Governments and public policy organisations. Given the urgency, this paper recommends that *ad hoc* SEBs should be established for this purpose.

Further issues are discussed in the Supplemental Material.

8 CONCLUSIONS

While this paper was originally motivated by Ferguson's public statements (e.g., [7, 25]), the broader evidence reviewed in this paper suggests that coding practice makes for poor science generally, not just in epidemiology — and many other studies (e.g., the classic [15]) support this view. Certainly, with such high stakes, we need to improve the quality of software engineering that supports public health policies, both in research and in public policy issues informed by research, such as track and trace [43] and in balancing COVID mutation pressures against vaccine shortages and delays between vaccinations [51].

The challenge to epidemic modelling, and to scientific research more generally, is to manage software development to reduce the unnoticed impact of bugs and poor programming practices. Computer code should be explicit and properly documented. Papers should be explicit on their software methodologies, limitations and weaknesses, just as Whitty expressed more generally about the standards of science [52]. Professional software methodologies should not be ignored.

Unfortunately, while programming is easy, and is often taken for granted, programming *well* is very difficult [44]. We know from software research than ordinary programming is very buggy and unreliable. Programming well is essential for informing public health policy. Simply put, without quality code and data, research is not open to scrutiny, let alone proper review, and its quality should be suspect. Many would argue that availability of code and data ensure research is reproducible, but that is a very weak criterion: both need to be documented and clear enough to be open to informed scrutiny [5, 29, 49].

We must prioritise getting appropriate professional software engineering skills and resources particularly to bear on the COVID-19 pandemic without delay. Software Engineering Boards (introduced in this paper) are a straight forward, constructive and practical way to support and improve computer-based modelling. This paper's Supplemental Material summarises the relevant professional software engineering practice that Software Engineering Boards would use, including discussing how and why software engineering helps improve code dependability and quality.

Future work should extend improving software engineering standards into other areas, particularly in healthcare [44, 56].

Notes [not all required by ACM TSEM, but provided for refereeing]

Acknowledgements. The author is grateful for comments from: Ross Anderson, Nicholas Beale, Ann Blandford, Paul Cairns, Rod Chapman, José Corréa de Sà, Paul Curzon, Jeremy Gibbons, Richard Harvey, Will Hawkins, Ben Hocking, Daniel Jackson, Peter Ladkin, Bev Littlewood, Paolo Masci, Stephen Mason, Robert Nachbar, Martin Newby, Patrick Oladimeji, Claudia Pagliari, Simon Robinson, Jonathan Rowanhill, John Rushby, Susan Stepney, Prue Thimbleby, Will Thimbleby, Martyn Thomas, and Ben Wilson provided very helpful comments.

Ethics. This article presents research with ethical considerations but does not fall not within the usual scope of ethics policies.

Data access. There is a full discussion of the method and its benefits in the Supplemental Material, section 3. All material is available for download at URL github.com/haroldthimbleby/Software-Enginering-Boards (which has been tested in a clean build, etc). The data is encoded in JSON.

Trivial JavaScript code checks and converts the master JSON data into LaTeX number registers and summary tables, etc, thus making it trivial to typeset results reliably in the paper (e.g., table 1 in this paper). In addition, a standard CSV file is generated in case this is more convenient to use, for instance to browse in Excel or other spreadsheet application. All data, and details of the method can be found in the Supplemental Material.

Authors contributions. Harold Thimbleby is the sole author. A very early short draft of this paper, containing no data was submitted to the UK Parliamentary Science and Technology Select Committee's inquiry into UK Science, Research and Technology Capability and Influence in Global Disease Outbreaks, under reference LAS905222, 7 April, 2020. The evidence, which was not peer reviewed and is only available after an explicit search, very briefly summarises the case for Software Engineering Boards but without the detailed analysis and case studies of the literature, etc, that are in the present paper. It is now available to the public [18, 45].

Competing interests. The author declares no competing interests.

Funding. This work was jointly supported by See Change (M&RA-P), Scotland (an anonymous funder), by the Engineering and Physical Sciences Research Council [grant EP/M022722/1], and by the Royal Academy of Engineering through the Engineering X Pandemic Preparedness programme [grant EXPP2021\1\186]. The funders had no involvement in the research or in the writing of this paper.

REFERENCES

[1] ACM. 2020. Artifact Review and Badging — Current. Artifact Review and Badging Version 1.1 (2020). URL www.acm.org/publications/policies/artifact-review-and-badging-current

[2] ACM. 2020. Code of Ethics and Professional Conduct. (2020). URL www.acm.org/code-of-ethics Accessed 23 April 2020.

[3] D. Adam. 2020. Modelling the pandemic: The simulations driving the world's response to COVID-19. *Nature* 580 (2020), 316–318. URL https://doi.org/10.1038/d41586-020-01003-6

[4] J. Baumann. 29 July, 2020. #MedBikini Backlash Exposes Research Ethics Boards' Digital Gaps. *Bloomberg Law* (29 July, 2020). URL news.bloomberglaw.com/pharma-and-life-sciences/medbikini-backlash-exposes-research-ethics-boards-digital-gaps

[5] F. C. Y. Benureau and N. P. Rougier. 2017. Re-run, Repeat, Reproduce, Reuse, Replicate: Transforming Code into Scientific. *Frontiers in Neuroinformatics* 11, 69 (2017). URL https://doi.org/10.3389/fninf.2017.00069

[6] D. S. Chawla. 8 June 2020. Critiqued coronavirus simulation gets thumbs up from code-checking efforts. *Nature* (8 June 2020). URL www.nature.com/articles/d41586-020-01685-y

[7] N. Ferguson. 22 March 2020. Tweet. URL twitter.com/neil_ferguson/status/1241835454707699713

[8] N. Ferguson. 22 March 2020. Tweet. URL twitter.com/neil_ferguson/status/1241835456947519492

[9] N. M. Ferguson, D. A. T. Cummings, C. Fraser, J. C. Cajka, P. C. Cooley, and D. S. Burke. 2005. Strategies for mitigating an influenza pandemic. *Nature* 437 (2005), 209–214. URL https://doi.org/10.1038/nature04017

[10] N. M. Ferguson, D. A. T. Cummings, C. Fraser, James C. Cajka, Philip C. Cooley, and Donald S. Burke. 2006. Strategies for mitigating an influenza pandemic. *Nature* 442 (2006), 448–452. URL https://doi.org/10.1038/nature04795

[11] Neil M. Ferguson, Daniel Laydon, Gemma Nedjati-Gilani, Natsuko Imai, Kylie Ainslie, Marc Baguelin, Sangeeta Bhatia, Adhiratha Boonyasiri, Zulma Cucunubá, Gina Cuomo-Dannenburg, Amy Dighe, Ilaria Dorigatti, Han Fu, Katy Gaythorpe, Will Green, Arran Hamlet, Wes Hinsley, Lucy C. Okell, Sabine van Elsland, Hayley Thompson, Robert Verity, Erik Volz, Haowei Wang, Yuanrong Wang, Patrick G. T. Walker, Caroline Walters, Peter Winskill, Charles Whittaker, Christl A. Donnelly, Steven Riley, and Azra C. Ghani. 16 March 2020. Impact of non-pharmaceutical interventions (NPIs) to reduce COVID-19 mortality and healthcare demand. URL www.imperial.ac.uk/media/imperial-college/medicine/sph/ide/gida-fellowships/Imperial-College-COVID19-NPI-modelling-16-03-2020.pdf

[12] Ibrahim Habli, Rob Alexander, Richard Hawkins, Mark Sujan, John McDermid, Chiara Picardi, and Tom Lawton. 2020. Enhancing COVID-19 decision making by creating an assurance case for epidemiological models. *BMJ Health & Care Informatics* 27, e100165 (2020), 1–5. URL https://doi.org/10.1136/bmjhci-2020-100165

[13] R. W. Hamming. 1987. *Numerical Methods for Scientists and Engineers.* Dover Publications Inc.

[14] S. Hardouin, T. W. Cheng, E. L. Mitchell, S. J. Raulli, D. W. Jones, J. J. Siracuse, and A. Farber. 2020. Prevalence of unprofessional social media content among young vascular surgeons. *Journal of Vascular Surgery* 72, 2 (2020),

667–671. URL https://doi.org/10.1016/j.jvs.2019.10.069

[15] L. Hatton and A. Roberts. 1994. How accurate is scientific software? *IEEE Transactions on Software Engineering* 20, 10 (1994), 785–797. URL https://doi.org/10.1109/32.328993

[16] H. Heesterbeek, R. M. Anderson, V. Andreasen, Viggo Andreasen, Shweta Bansal, Daniela De Angelis, Chris Dye, Ken T. D. Eames, W. John Edmunds, Simon D. W. Frost, Sebastian Funk, T. Deirdre Hollingsworth, Thomas House, Valerie Isham, Petra Klepac, Justin Lessler, James O. Lloyd-Smith, C. Jessica E. Metcalf, Denis Mollison, Lorenzo Pellis, Juliet R. C. Pulliam, Mick G. Roberts, Cecile Viboud, and Isaac Newton Institute IDD Collaboration. 2015. Modeling infectious disease dynamics in the complex landscape of global health. *Science* 347, 6227 (2015), aaa4339. URL https://doi.org/10.1126/science.aaa4339

[17] C. A. R. Hoare. 1969. An axiomatic basis for computer programming. *Commun. ACM* 12, 10 (1969), 576–580. URL https://doi.org/10.1145/363235.363259

[18] House of Commons Science and Technology Committee. 16 December 2020. *The UK response to covid-19: use of scientific advice*. Vol. HC 136. House of Commons. URL committees.parliament.uk/publications/4165/documents/41300/default

[19] International Electrotechnical Commission (IEC). 2010. *IEC 61508:2010 CMV Commented version, Functional safety of electrical/electronic/programmable electronic safety-related systems*. URL webstore.iec.ch/publication/22273

[20] Norbert L. Kerr. 1998. HARKing: Hypothesizing after the results are known. *Personality and Social Psychology Review* 2, 3 (1998), 196–217. URL https://doi.org/10.1207/s15327957pspr0203_4

[21] S. M. Kissler, C. Tedijanto, E. Goldstein, Stephen M. Kissler, Christine Tedijanto, Edward Goldstein, Yonatan H. Grad, and Marc Lipsitch. 2020. Projecting the transmission dynamics of SARS-CoV-2 through the postpandemic period. *Science* (2020). URL https://doi.org/10.1126/science.abb5793

[22] D. E. Knuth. 1998. *The Art of Computer Programming (Seminumerical algorithms)* (3rd ed.). Vol. 2. Addison-Wesley.

[23] J. Kruger and D. Dunning. 1999. Unskilled and Unaware of It: How Difficulties in Recognizing One's Own Incompetence Lead to Inflated Self-Assessments. *Journal of Personality and Social Psychology* 77, 6 (1999), 1121–1134. URL https://doi.org/10.1037/0022-3514.77.6.1121

[24] P. B. Ladkin, B. Littlewood, H. Thimbleby, and M. Thomas. 2020. The Law Commission presumption concerning the dependability of computer evidence. *Digital Evidence and Electronic Signature Law Review* 17 (2020). URL https://doi.org/10.14296/deeslr.v17i0.5143(NBSee\url{journals.sas.ac.uk/deeslr/article/view/5143}untiltheDOIisresolved.)

[25] J. Leake. 29 March 2020. Neil Ferguson interview: No 10's infection guru recruits game developers to build coronavirus pandemic model. *The Sunday Times* (29 March 2020). URL www.thetimes.co.uk/article/neil-ferguson-interview-no-10s-infection-guru-recruits-game-developers-to-build-coronavirus-pandemic-model-zl5rdtjq5

[26] Mandeep R. Mehra, Sapan S. Desai, Frank Ruschitzka, and Amit N. Patel. 2020. RETRACTED: Hydroxychloroquine or chloroquine with or without a macrolide for treatment of COVID-19: A multinational registry analysis. (2020). URL https://doi.org/10.1016/S0140-6736(20)31180-6

[27] K. G. Moons, D. G. Altman, J. B. Reitsma, J. P. Ioannidis, P. Macaskill, E. W. Steyerberg, A. J. Vickers, D. F. Ransohoff, and G. S. Collins. 2015. Transparent Reporting of a multivariable prediction model for Individual Prognosis or Diagnosis (TRIPOD): Explanation and elaboration. *Annals of Internal Medicine* 162, 1 (2015), W1–73. URL https://doi.org/10.7326/M14-0698

[28] Edward Nuhfer, Steven Fleisher, Christopher Cogan, Karl Wirth, and Eric Gaze. 2017. How Random Noise and a Graphical Convention Subverted Behavioral Scientists' Explanations of Self-Assessment Data: Numeracy Underlies Better Alternatives. *Numeracy* 10, 1 (2017), 4. URL https://doi.org/10.5038/1936-4660.10.1.4

[29] K. R. Popper. 2002. *Conjectures and Refutations: The Growth of Scientific Knowledge* (2nd ed.). Routledge.

[30] F. Redmill. 2000. Understanding the Use, Misuse and Abuse of Safety Integrity Levels. In *Lessons in System Safety, Eighth Safety-critical Systems Symposium*. URL homepages.cs.ncl.ac.uk/felix.redmill/publications/1%20SILs.pdf Revised.

[31] D. Richards and K. Boudnik. 16 May 2020. Neil Ferguson's Imperial model could be the most devastating software mistake of all time. *The Telegraph* (16 May 2020). URL www.telegraph.co.uk/technology/2020/05/16/neil-fergusons-imperial-model-could-devastating-software-mistake

[32] Derek Richards, Angel Enrique, Nora Eilert, Matthew Franklin, Jorge Palacios, Daniel Duffy, Caroline Earley, Judith Chapman, Grace Jell, Sarah Sollesse, and Ladislav Timulak. 2020. A pragmatic randomized waitlist-controlled effectiveness and cost-effectiveness trial of digital interventions for depression and anxiety. *Nature Digital Medicine* 3, 85 (2020). URL https://doi.org/10.1038/s41746-020-0293-8

[33] B. Sander, A. Nizam, L. P. Garrison Jr, M. J. Postma, M. E. Halloran, and Longini Jr. I. M. 2009. Economic evaluation of influenza pandemic mitigation strategies in the us using a stochastic microsimulation transmission model. *Value Health* 12, 2 (2009), 226–233. URL https://doi.org/10.1111/j.1524-4733.2008.00437.x

[34] A. Sayburn. 2020. Covid-19: Experts question analysis suggesting half UK population has been infected. *BMJ* 368 (2020), m1216. URL https://doi.org/10.1136/bmj.m1216

[35] Andrew Scheuber and Sabine L. van Elsland. 1 June 2020. Codecheck confirms reproducibility of COVID-19 model results. (1 June 2020).
URL www.imperial.ac.uk/news/197875/codecheck-confirms-reproducibility-covid-19-model-results

[36] S. Schnell. 2015. Ten Simple Rules for a Computational Biologist's Laboratory Notebook. *PLoS Computational Biology* 11, 9 (2015), e1004385. URL https://doi.org/10.1371/journal.pcbi.1004385

[37] Kelly Servick. 2020. COVID-19 data scandal prompts tweaks to elite journal's review process. *Science* (2020).
URL https://doi.org/10.1126/science.abe8656

[38] Kelly Servick and Martin Enserink. 2020. A mysterious company's coronavirus papers in top medical journals may be unraveling. *Science* (2020). URL https://doi.org/10.1126/science.abd1337

[39] B. Shneiderman. 2016. Opinion: The dangers of faulty, biased, or malicious algorithms requires independent oversight. *Proceedings National Academy of Sciences* 113, 48 (2016), 13538–13540.
URL https://doi.org/10.1073/pnas.1618211113

[40] D. Speigelhalter. 2019. *The Art of Statistics: Learning from Data.* Pelican Books.

[41] The Editors. 2020. Learning from a retraction. *The Lancet* (2020). URL https://doi.org/10.1016/S0140-6736(20)31958-9

[42] The Editors (of *Journal of Vascular Surgery*). 2020. Editors' Statement Regarding "Prevalence of unprofessional social media content among young vascular surgeons". *FaceBook* (2020).
URL www.facebook.com/TheJVascSurg/photos/a.611986142331744/1381024778761206/?type=3&theater, and copied to journal home page URL www.jvascsurg.org Accessed 30 July 2020.

[43] Harold Thimbleby. 2020. The problem isn't Excel, it's unprofessional software engineering. *BMJ* 371, m4181 (2020).
URL https://doi.org/10.1136/bmj.m4181

[44] H. Thimbleby. 2021. *Fix IT: How to solve the problems of digital healthcare.* Oxford University Press.

[45] Harold Thimbleby. 29 April 2020. Written Evidence Submitted by Harold Thimbleby. See[18].
URL committees.parliament.uk/work/91/default/publications/written-evidence/?SearchTerm=thimbleby

[46] Harold Thimbleby. 9 May, 2004. Give your computer's IQ a boost — *Journal of Machine Learning Research. Times Higher Education Supplement* (9 May, 2004).
URL www.timeshighereducation.co.uk/story.asp?sectioncode=26&storycode=176549

[47] H. Thimbleby, S. O. Anderson, and P. Cairns. 1999. A Framework for Modelling Trojans and Computer Virus Infection. *Computer Journal* 41 (1999), 444–458. URL https://doi.org/10.1093/comjnl/41.7.444

[48] H. Thimbleby and P. Cairns. 2017. Interactive numerals. *Royal Society Open Science* 4, 4 (2017), 160903.
URL https://doi.org/10.1098/rsos.160903

[49] H. Thimbleby and D. Williams. 2018. A tool for publishing reproducible algorithms & A reproducible, elegant algorithm for sequential experiments. *Science of Computer Programming* 156 (2018), 45–67.
URL https://doi.org/10.1016/j.scico.2017.12.010 Also on GitHub: URL GitHub.com/haroldthimbleby/relit.

[50] Verity, R., Okell, L. C., Dorigatti, I. Peter Winskill, Charles Whittaker, Natsuko Imai, Gina Cuomo-Dannenburg, Hayley Thompson, Patrick G. T. Walker, Han Fu, Amy Dighe, Jamie T. Griffin, Marc Baguelin, Sangeeta Bhatia, Adhiratha Boonyasiri, Anne Cori, Zulma Cucunubá, Rich FitzJohn, Katy Gaythorpe, Will Green, Arran Hamlet, Wes Hinsley, Daniel Laydon, Gemma Nedjati-Gilani, Steven Riley, Sabine van Elsland, Erik Volz, Haowei Wang, Yuanrong Wang, Xiaoyue Xi, Christl A. Donnelly, Azra C. Ghani, and Neil M. Ferguson. 2020. Estimates of the severity of coronavirus disease 2019: A model-based analysis. *Lancet* (2020).
URL https://doi.org/10.1016/S1473-3099(20)30243-7

[51] Meredith Wadman. 2021. Could too much time between doses drive the coronavirus to outwit vaccines? *Science* (2021). URL https://doi.org/10.1126/science.abg5655

[52] C. J. M. Whitty. 2015. What makes an academic paper useful for health policy? *BMC Medicine* 13 (2015), 301.
URL https://doi.org/10.1186/s12916-015-0544-8

[53] J. C. P. Woodcock, P. G. Larsen, J. C. Bicarregui, and J. S. Fitzgerald. 2009. Formal methods: Practice and experience. *Comput. Surveys* 41, 4 (2009). URL https://doi.org/10.1145/1592434.1592436

[54] Laure Wynants, Ben Van Calster, Marc M. J. Bonten, Gary S. Collins, Thomas P. A. Debray, Maarten De Vos, Maria C. Haller, Georg Heinze, Karel G. M. Moons, Richard D. Riley, Ewoud Schuit, Luc J. M. Smits, Kym I. E. Snell, Ewout W. Steyerberg, Christine Wallisch, and Maarten van Smeden. 2020. Prediction models for diagnosis and prognosis of covid-19 infection: Systematic review and critical appraisal. *BMJ* 369, m1328 (2020).
URL https://doi.org/10.1136/bmj.m1328

[55] Y. Zhang, P. Masci, P. Jones, and H. Thimbleby. 2019. User Interface Software Errors in Medical Devices. *Biomedical Instrumentation & Technology* 53, 3 (2019), 182–194. URL https://doi.org/10.2345/0899-8205-53.3.182

[56] Y. Zhang, P. Masci, P. Jones, and H. Thimbleby. 2019. User Interface Software Errors in Medical Devices. *Biomedical Instrumentation & Technology* 53, 3 (2019), 182–194. URL https://doi.org/10.2345/0899-8205-53.3.182

[57] A. Zlojutro, D. Rey, and L. Gardner. 2019. A decision-support framework to optimize border control for global outbreak mitigation. *Nature Scientific Reports* 9, 2216 (2019). URL https://doi.org/10.1038/s41598-019-38665-w

Supplemental Material: A proposal to achieve professional software engineering in scientific research

HAROLD THIMBLEBY, See Change Fellow in Digital Health, Wales

ACM Reference Format:
Harold Thimbleby. 2021. Supplemental Material: A proposal to achieve professional software engineering in scientific research. *Submitted ACM Trans. Softw. Eng. Methodol.* 1, 1 (April 2021), 27 pages.

Author's address: Harold Thimbleby, See Change Fellow in Digital Health, Wales, harold@thimbleby.net.

© 2021 Harold Thimbleby
1049-331X/2021/4-ART
URL https://doi.org/\@acmDOI

1 FURTHER ISSUES FOR SOFTWARE ENGINEERING BOARDS (SEBS)

1.1 Brief definition

Software Engineering Boards, henceforth SEBs, will be used to help and assure that critical code, including epidemic modelling, is of high standard, to provide assurance for scientific papers, Government public health and other policies, etc, that the code used is of appropriate quality for its intended uses.

Further details of the SEB proposal is in the main paper. Here we raise further issues for SEBs (additional to those covered in the main paper's introduction to SEBs), potential limitations and possible responses that can be addressed over time:

(1) Until there are national qualifications, nobody — certainly nobody without professional training in software — really knows just how bad (or good) they are at software engineering.
(2) When code is taken seriously, concerns may be raised on programmers' contributions to research, intellectual property rights, and co-authoring [85]. Software engineering is a hard, creative discipline, and getting epidemiological (and other scientific) models to work is generally a significant challenge, on a par with the setting up and exploring the mathematical models themselves. Often software engineers will need to explore boundary cases of models, and this typically involves hard technical mathematics [13]. Often the software engineers will be solving entirely new problems and contributing to the research. How this is handled needs exploring. How software engineers are appropriately credited and cited for their contributions also needs exploring.
(3) SEBs require policies on professional issues such as membership, transparency, and accountability.
(4) There should be a clear separation between the SEB members' activities as part of the Board, and their other activities, including professional advice, code development, or training (which is likely to be in demand from the same people who require formal approvals from the SEBs).
(5) Professional Engineering Bodies have a central role to play in professionalism, ranging from education and accreditation to providing professional structures and policies for SEBs. For example, should and if so how should the programming skills taught to computational scientists (epidemiologists, computational biologists, economists, computational chemists, ...) be accredited?
(6) In the main paper, SEBs are viewed as a constructive contribution to good science, specifically helping improve the quality of epidemiological modelling. More generally, SEBs will have wider roles, for instance in overseeing software subject to medical device regulation [44].
(7) SEBs may fruitfully collaborate with other engineering disciplines to share and develop best practice. For example, engineers in other domains (e.g., civil engineers) routinely sign off projects, yet, on the other hand, they often overlook the quality of software engineering their projects implicitly rely on — for the same reasons as the scientific work discussed in this paper overlooks the dependence on quality software.
(8) Clearly, at least while this paper's concepts are tested and mature, SEBs will need to collaborate closely with research organisations, journals, and funding agencies in order to develop incremental developments to policies and processes that will be most effective, and which can be introduced most productively over time to the scientific community at large. Funding agencies may wish to support such strategic work, as they have previously funded one-off projects such as [80].

There are other ideas to help make SEBs work, but it is clear they are part of the solution. We must not let perfection be the enemy of the good. SEBs don't need to be perfect on day one, but they do need to get going in some shape or form to start making their vital contribution.

1.2 Relationships of SEBs to Ethics Boards

(1) Although SEBs may start with a checklist approach, like Ethics Boards generally do, it cannot be assumed that people approaching SEBs know enough about software engineering to perform adequate software assessments when there is any risk (as there is in public policy, medical apps, and so on). SEBs may also provide mentoring and training.
(2) Unlike Ethics Boards, which provide hands-off oversight, SEBs should provide professional advice, perhaps providing training or actually helping hands-on develop appropriately reliable software. During a pandemic SEBs would be very willing to do this, but in the long run it is not sustainable as voluntary labour, so all research, particularly medical research, should include support for professional software engineering.
(3) Ethics Boards typically require researchers to fill in forms and provide details, which is a feasible approach as researchers know if they are doing experiments on children, for instance, so the forms are relatively easy to fill in (if often quite tedious). On the other hand, few healthcare and medical researchers understand software and programming, so they are *not* able to fill in useful software forms on their own. SEBs need to know how well engineered the software really is, not how good its developers *think* it is. As typical programs are enormous, SEBs are either going to need resources to evaluate programs, or they will need to supervise independent bodies that can do it for them.
(4) SEBs should have a two-way collaboration with Ethics Boards.
   - SEBs have to deal with ethical concerns, and how they may be implemented in code. One of the papers [105] in the survey (discussed later in this Supplemental Material) is a case in point, as is the growing cross-fertilisation between AI and ethics (e.g., [72]).
   - Ethics Boards also have to deal with software, and it is clear that they often fail to do this effectively. The case of the retraction of a peer reviewed articles for *The Lancet* [37, 38, 41] and the *Journal of Vascular Surgery* [4, 14, 42], discussed in the main paper, are cases in point.
(5) Like some Ethics Boards, SEBs might become, or be perceived as becoming, onerous and heavy handed — as if the Board is not interested in ethics but only in following a bureaucratic pathway. It seems essential, then, that SEBs have (and perhaps are chaired by) experienced, practising, professional software engineers to avoid this problem.

1.3 SEBs are necessary but not sufficient

The main paper provides evidence and argues that SEBs (or equivalent) are necessary to help improve the quality of science, specifically science relying, explicitly or implicitly, on tools or methods based in software.

SEBs address the problems identified at the laboratory end of doing science; they do not address the processes of review, editorial control, and action based on claimed results. As shown in the review of 32 papers, only some journals have code policies, and the policies are not enforced. In other words, improving the professionalisation of software engineering has to proceed from doing science, which the paper covers, to the downstream issues of review and publication. SEBs may work with journals, funding agencies and even international standards agencies to improve broader awareness of professional software engineering, but this is a topic the present paper has not addressed. It needs doing.

2 SOFTWARE ENGINEERING BEST PRACTICE

2.1 Introduction and standard references

This Supplemental Material provides more explanations and justification for following standard software engineering practices that support reliable modelling, reliable research, and, most generally, reliable science.

The reader is referred to standard textbooks for more information (e.g., [79, 69]), as well as to specialised texts that are more specifically addressed to software engineering in science (e.g., [80]).

The book *Why Programs Fail* [87] is a very good practical guide to developing better code, and will be found very accessible. Humphrey [67] outlines a thorough discipline for anyone wanting to become a good programmer. Improvement is such an important activity, Humphrey has also published a book to persuade managers of the benefits [66]. Further suggestions for background reading can be found throughout this section.

2.2 Essential components of best practice

Software Engineering includes the following topics, which are discussed at more length below:

(0) Requirements
(1) Formal methods
(2) Defensive programming
(3) Using dependable programming languages
(4) Open source and version control
(5) Rigorous testing
(6) Good documentation and record keeping
(7) Usability
(8) Reusing quality solutions
(9) Simplicity
(10) Compliance with standards
(11) Effective multidisciplinary teamwork
(12) Continuous Professional Development (CPD)
(13) Security and other factors

(0) *Without defining requirements, not enough skilled effort will be put into designing and implementing reliable software — or excess effort will be wasted.*

It is not always necessary to program well if the code to be produced is for fun, experimenting, or for demonstrations. On the other hand, if code is intended for life-critical applications, then it is worth putting more engineering effort into it. The first step of software engineering, then, is to assess the requirements, specifically the reliability requirements of the code that is going to be produced.

In practice, requirements and expectations change. Early experimental code, developed informally, may well be built on later to support models intended to inform public policy, for instance. Unfortunately, prototypes may impress project leaders who then want to rush into production software because, it seems, "it obviously works." Fortunately, best practice software engineering can be adopted at any stage, particularly by using *reverse engineering*. In reverse engineering, one carefully works out (generally partly automatically) what has already been implemented. This specification, carefully reviewed, is then used as the basis for a more rigorous software engineering process that implements a more reliable version of the system.

(1) *Without formal methods, there is no rigorous and checked specification of a program, so nobody — including its developers — will know exactly what it is supposed to do.*

In the physical world, to do something as simple as design and build a barbeque, you would need to use elementary mathematics to calculate how many bricks to buy. To build something more substantial, such as block of flats, you would need to use structural engineering (with certified structural engineers) to ensure the building was safe. Although programming lends itself to mathematical analysis, it is surprising that few programmers use explicit mathematics at all in the design and implementation of software.

The type and use of mathematics used in software engineering is formal methods. Not using formal methods ensures the resulting code is unsafe and unreliable. Of particular relevance to scientific modelling: there must be an explicit use of formal methods to ensure mathematical models (such as differential equations) are correctly implemented in code (and to understand the any limitations of doing so).

Formal methods require sophisticated knowledge of logic [53], as well as practical knowledge of using appropriate formal methods tools (Alloy, HOL, PVS, SPARK, and *many* others). Using the right tools is essential for reliable programming, because the tools do quickly and reliably what, done by hand, would be slow and error-prone. Standard tools cover verification, static analysis of code, version control, documentation, and so on — this paper explains why some of these activities are essential for reliable programming below.

Crucially, tools are designed to catch common human errors that we are all prone to. Many tools are designed to avoid common human errors arising *in the first place*; notably, the MISRA C toolset simply stops the developer using the most error-prone features of normal C, and hence improves the quality of programming with little effort.

Many programming languages and programming environments have integrated features that support formal methods. For example, Hoare's triples [17] (and formal thinking based on similar ideas) are readily supported by assertions, as either provided explicitly in a programming language or through a simple API. In particular, assertions readily support contracts, an important rigorous way of programming: assertions allow the program, the programming language, or tools (as the case may be) to automatically (and hence rigorously) check essential details of the program. Hoare's original 1969 paper [17] is very strongly recommended because it is a classic paper that has stood the test of time; in the 1960s it was leading research, but now it can be read as an excellent introduction, given how the field of software engineering has advanced and become more specialised and sophisticated over the decades since. Hoare is also a very good writer.

Formal methods have the huge advantage that they "think differently" and therefore help uncover design problems and bugs that can be found in no other way. Because formal methods are logical, mathematical theories (safety properties, and so forth) can be expressed and checked (often automatically); this provides a very high degree of insight into a program's details, and hence supports fault tolerance (e.g., redundancy). Ultimately, formal methods provides good reasons to believe the quality of the final code — that it does what it is supposed to do. Unfortunately, because formal methods are mathematical, few programmers have experience of using them. Fortunately tools are widely available to help use formal methods very effectively.

(2) *Without defensive programming, any errors — in data, code, hardware, or in use — will go unnoticed and be uncorrected.*

Defensive programming is based on a range of methods, including error checking, independent calculation (using multiple implementations written by independent programmers), assertions, regression testing, etc. Notoriously, what are often unconsciously dismissed as trivial concerns frequently lead to the hardest to diagnose errors, such as buggy handling of "well-known, trivial" things like numbers [48]. The great advantage of defensive programming is that it detects, and may be able to recover from, bugs that have been missed earlier in the development process (such as typos in the code). Defensive programming requires professional training to be used effectively, for example it is not widely known that some choices of programming language make defensive programming unnecessarily hard [83].

A special case of defensive programming appropriate for pandemic modelling is mixing methods. Do not rely on one programming method, but mix methods (e.g., different numerical methods) to use and compare multiple approaches to the modelling.

(3) *Using inappropriate programming languages undermines reliability.*

Many popular languages are popular because they are easy to use, which is not the same as being reliable to use. The fewer constraints a language imposes, the easier it *seems* to be to program in, but the lack of constraints means the language cannot provide the checks stricter languages do. C, for instance, which is one of the languages widely used for modelling [7, 63], is not a good choice for a reliable programming language — it has many intrinsic weaknesses that are well-known to professionals, but which frequently trap inexperienced programmers. (This is not the place for a review [83] but Excel is even worse in this regard.) In particular, C is not a portable language (unless extreme care is taken), which means models will work differently on different types of computer. SPARK Ada is one example of a much more appropriate programming language to use. SPARK Ada also has the advantage that most Ada programmers are better qualified than most C programmers.

(4) *Version control and open source organises and helps software development.*

It is appreciated that the models may change and be adapted as new data and insights become available. Changing models makes it even harder to ensure that they are correct, and thus emphasises the relevance of the core message this paper: we have to find ways to make computer models more reliable, inspectable, and verifiable. Version control keeps a record of what code was used when, and enables reconstruction of earlier versions of code that has been used. Version control is supported by many tools (such as Git, Subversion, etc).

If version control is not used, one has no idea what the current program actually is. Version control is essential for *reproducibility*: [5] it enables efforts to duplicate work to start with the exact version that was used in any published paper, provided that the published paper discloses the version and a URL for the relevant repository. Note that version control should also be used for data and web site data used by code, otherwise the results reported are not replicable.

If results cannot be reproduced, has anything reliable been contributed? When a modelling paper presents results from a model, it is important to reproduce those results without using the same code. Better still, research should be reproduced without sharing libraries or APIs (for example, results from a model using R might be reproduced using Mathematica — this is a case of N (where, in this case, $N = 2$) version Programming [15]). Reproducing the same results relying on the same codebase tells you little. The more independent reproductions of results the greater the evidence for belief in the implications.

Clearly, with the transformations a program from avian flu in Thailand [9] to COVID-19 in the United States and in Great Britain [11] taking place over many years, version control would have been very helpful to keep proper track of the changes. Note that professional version control repositories also provide secure off-site back up, ensuring the long-term access to the code and documentation — this would avoid loss of Supplemental Material problems, as occurred in [33].

Most version control systems would, in addition, enable open source methods so the code could be shared — and reviewed — by a wider community. Open source is not a panacea, however; it raises many trade-offs. Particularly for world-wide concerns like pandemic modelling, it increases diversity in the software developers, and fosters a diverse scientific collaboration. Open source can raise people's standards — some countries [58, 59] are using Excel models to manage COVID-19, and open source projects properly implemented would help these people enormously.

Open source raises important licensing and management questions to ensure the quality of contributions. A salutary open source case is NPM, where lawyers from a company called Kik triggered Azer Koçulu, that is, a *single* programmer, to remove all his code from a repository. This caused problems to many thousands of JavaScript programmers worldwide who could no longer compile anything — ironically, including Kik itself [74].

Critically in the case of epidemic modelling, open source democratises the model development and interpretation, and enables properly-informed public debate. Note that many (if not most) successful open source projects have had a closed team of highly dedicated and well-paid developers.

(5) *Without professional testing, there is no acceptable evidence that a program works under real conditions.*

In poorly-run software development it is very easy to miss bugs, because the flawed thinking that inserted bugs in the code is going to be the same flawed thinking with the same misconceptions that tries to detect them. Rigorous testing includes methods like fault injection. Here, the idea is that if testing finds no bugs, that may be because the testing is not rigorous enough rather than that the program actually has no bugs. Fault injection inserts random bugs, and then testing gives statistical insights into the number of bugs in a program (depending on how many deliberate bugs it successfully finds).

It is very tempting to test code while it is being built, save some or all of the code on a repository, but forget to check that the code has not changed out of recognition of the earlier tests — tests should be saved so that modified code can easily be tested again. For example, if a test reveals a bug, the bug should be fixed *and* the test needs to be re-run to check the fix worked (and did not introduce other bugs previously eliminated).

It is important that code is saved and then downloaded to a clean site, confirmed it is consistent, and a new build made (preferably by an independent tester), which is then re-tested. If this procedure (or equivalent) is not followed, there is no assurance that the code made available with the paper is complete and works reliably.

There are many other important testing methods [15, 79, 69].

(6) *Without documentation and record keeping, nobody — least of all the programmer — knows what code is supposed to do or how to get it to do it.*

Documentation covers internal documentation (how code works), developer (how to include it in other programs), configuration (how to configure and compile the code in different environments), external documentation (how the code is used), and help (documentation available while using the program).

For critical projects, such as for pandemic modelling, all documentation (including software) should be formally controlled, typically digitally signed and backed up in secure repositories. One would also expect a structured assurance case to be made, both to help the authors understand and complete their own reasoning and to help reviewers scrutinise it [12].

For purely scientific purposes, perhaps the most important form of internal is internal documentation: how to understand how and why the code works. This is different from developer documentation, which is how to *use* the code in other programs. For example, code for solving a differential equation needs explaining — what method does it use, what assumptions does it have? In contrast, the developer documentation for differentiation would say things like it solves ordinary differential equations with parameters e for the function f with the independent variable x in the interval $[u, v]$, or whatever, but *how* it solves equations is of little interest to the developer who just needs to use it. How code works — internal documentation — is essential for the epidemiologist, or more generally any scientist. An example of a simple SIR epidemiological model's internal documentation can be found at URL http://www.harold.thimbleby.net/sir

There are many tools to help manage documentation (Javadoc, Doxygen, ...). Literate programming is one very effective way of documenting code, and has been used for very large programming projects [70]. Literate programming has also been used directly to help publish clearer and more rigorous papers based on code [49] — a paper that also includes a wider review of the issues.

Documentation should be supplemented by details of algorithms and proofs of correctness (or references to appropriate literature). All the documentation needs to be available to enable others to correctly download, install and correctly use a program — and to enable them, should they wish, to repurpose it reliably for their own work. In addition, documentation requires specifications and, in turn, *their* documentation.

A important role of documentation is to cover configuration: how to get code to work — without configuration, code is generally useless. The most basic is a README file, which explains how to get going; more useful approaches to configuration include make files, which are programs that do the configuration automatically.

Without proper record keeping, code becomes almost impossible to maintain if programmers leave the project. Note that computer tools can make record keeping, laboratory books etc, trivial — if they are used.

(7) *If code is not usable, even if it is "correct" it will not be used and interpreted correctly.*

Usability is an important consideration: [76, 82] is the program usable by its intended users so they can obtain correct results? Often the programmers developing code know it so well they misjudge how easy it will be for anyone else to use it — this is a very serious problem for the lone programmer (possibly working in another country) supporting a research team. Usability is especially important when programs are to be used by other researchers and by non-programmers, including epidemiologists.

In publishing science, an important class of user includes the scientists and others who will use or replicate the work described. When code used in research is non-trivial, it is essential that the process of successfully downloading code and configuring it to run is made as usable as possible. Typically so-called makefiles are provided, which are shell scripts or apps that run on the target machine, establish its hardware and other features, then automatically configure and compile the code to work on that machine. Makefiles typically also provide demo and test runs and other helpful features. Other approaches to improve usability are zip files, so every relevant file can be conveniently downloaded in one step, and using standard repositories, such as GitHub which allow new forks to be made, and so on.

(8) *Without using existing solutions (libraries, APIs, etc) reinventing code merely reinvents bugs.*

Reusing quality code (mathematical functions, database operations, user interface features, connectivity, etc) avoids having to develop it oneself, saves time and avoids the risks of introducing new bugs. The more code that is reused, the more likely many people will have contributed to improving it — for example, reusing a standard database package will provide Atomicity, Consistency, Isolation, and Durability (so-called ACID properties) without any further work (nor even needing to understanding what useful guarantees these basic properties ensure).

Note that reusing code assumes the originators of the code followed good software engineering practice — particularly including good documentation; equally, if the code being developed building on it follows good software engineering practice, it too can be shared and further improved as it gets more exposure. Its quality improves through having scrutiny by the wider community, and in successful cases, leading to consensus on the best methods. Indeed, reuse, scrutiny, and consensus are the foundations of good science.

Anticipating reuse during program development is called *flexibility*, where various programming techniques can greatly enhance the ease and reliability of reuse [64].

A special case of reuse is to use software tools to help with software development. The tools (if appropriately chosen) have been carefully developed and widely tested. Tools enable software developers to avoid or solve complex programming problems (including maintenance) repeatedly and with ease.

(9) *Poor programmers often fix bugs rather than the causes of bugs: complexity and obfuscation.*

When a program doesn't quite do what is wanted, it is tempting to add more features or variables, or to treat the problem as an "exception" and program around it — which inserts more code and, almost certainly, more bugs. This way lies over-fitting, a problem familiar from statistics (and machine learning). Programs can be made over-complex and they can then do anything; an over-complex program may seem correct by accident. Instead, the hallmarks of good science are that of parsimony and simplicity; if a simple program can do what is needed it is more likely to be correct. A simpler program is easier to prove correct, easier to program, and easier to debug. A special case of needing simplicity is when fixing bugs: instead of fixing bugs one at a time, one should be fixing the *reasons* why the bugs have happened. Generally, when bugs are fixed, programmers should determine *why* the bugs occurred, and thence repair the program more strategically.

(10) *International standards have been developed to support critical software development.*

To ensure adherence to best practice and, importantly, to avoid being unaware of relevant methodologies, professional software development projects adopt and adhere to relevant standards, such as ISO/IEC/IEEE 90003:2018 [68]. However, for safety-critical models or models of national policy significance, much stronger standards such as aviation software standards, such as RTCA DO-178C/EUROCAE ED-12C [73], commonly called DO-178C, will be more appropriate. Publications should then cite the standards to which their computer models comply.

Note that medical device regulation, which has its own standards, is lagging behind professional software engineering practice, and currently provides no useful guidance for critical software development [44].

(11) *Effective multidisciplinary teamwork is essential because no individual has the capacity to develop non-trivial reliable software.*

As this long list illustrates, Software Engineering is a complex and wide-ranging subject. Software engineering cannot be done effectively by individuals working alone (for instance, code review is impossible for individuals to perform effectively), even without considering the complexities of the domain the code is intended for (in the present case, including pandemic modelling, mathematical modelling, public health policy, etc). Multidisciplinary teamwork is essential.

Modern software is complex, and no one person can have the skills to understand all relevant aspects of all but the most trivial of programs. Furthermore, programming is a cognitively demanding task, and causes loss of situational awareness (that is, cognitive "overload" making one unable to track requirements beyond those thought to be directly related to the specific task in hand). The main solution to both problems is teamwork, to bring fresh insights, different mindsets and skills to the task.

Peer review of code is an essential teamwork practice in reliable program development: [61, 69] it is easy to make programming mistakes that one is unaware of, and an independent peer review process is required to help identify such unnoticed errors.

Almost all software will be used by other people, and user interface design is the field concerned with developing usable and effective software. A fundamental component of user interface design is working with users and user testing: without engaging users, developers are very likely to introduce quirks that make systems less usable (often less safe) than they should be. In short, users have to be brought into the software team too.

(12) *Computing technologies are advancing rapidly, and best practice in software engineering is continually evolving.*

As computing technology continues to develop rapidly — especially as new programming tools and systems are introduced — best practice in software engineering is also rapidly evolving. Continuous Professional Development (CPD) is essential.

Ironically, the more organised CPD the more likely the content itself will lag behind. There is an argument for two-way links between universities (and other research organisations), research science developers, including enabling developers to undertake part-time research degrees. Research degrees teach not just current best-practice but also how to stay abreast of the relevant technologies and literature as it develops.

The UK's Software Sustainability Institute is one initiative that is making important contributions [62, 78], and its web site will no doubt remain timely and up to date in a way that this paper cannot.

Note that CPD is not just a matter of learning current best practice, but a continual process as best practice itself continually evolves. In software engineering, a current (as of 2021) initiative concerns reproducible code artifacts and badging papers to clearly show the approaches they take [1], and this will in due course have a direct impact on software engineering standards in other fields.

(13) *Other factors ....*

Of course, there are many other factors to be considered for the professional development of critical code, such as using appropriate methods to ensure cybersecurity [60, 77], particularly while also being able to up- and download secure updates.

For pandemic modelling specifically, understanding the limitations of numerical methods (in particular, how numerical methods are affected by the choice of programming language and style of programming) is critical.[1] Hamming [13] is considered a classic, but there is a huge choice available.

For reasons of space, the present paper does not discuss the issues raised by AI, nor the many very important, non-trivial social and professional concerns, which have complex implications for software engineering practice, such as managing programming teams, data ethics, privacy, legal

[1]For example [111] was noticed to use literal numbers at too high a precision for the chosen language, where conformant implementations use IEEE 754 double precision 64-bit floating point. Such an error typically has an undefined impact on results, and unfortunately is easy to overlook as the program almost certainly ignores the error when running.

liability [75], or software as a matter in law, as in disputes over model results or disputes over ownership of code [71].

3 CODE, DATA AND PUBLICATION

There is no fundamental difference between code and data, and no distinction that is relevant for scientific publication purposes. There is no distinction one can imagine that cannot easily, even accidentally, be circumvented. In other words, a journal's data policies and code policies should be the identical — and the conventionally stricter data policies should also apply to code. It is baffling that some journals have data policies that are weaker than their data policies; it is certainly indefensible to have no code policies at all.

Significant cybervulnerabilities result from there being no difference between code and data. For example: an email arrives, which brings data to a user. The user opens an attachment, perhaps a word processor text document, which is more data. The word processor runs macros in the text document — but now it is code. The macros move data onto the user's disc. The data there then runs as code, and corrupts the user's data across the disc — which includes both data and code stored in files. And so on. Each step of a computer virus infection crosses over non-existent "boundaries" between data and code [84].

This section's discussion may sound like arcane and irrelevant pedantry, but these issues are at the very foundations of Computer Science.[2] If we ignore or misunderstand these basic things — or overlook them in policies and procedures — bugs are the inevitable (and confusing) consequence.

The main paper points out that data is often embedded in code as "magic numbers." Let's now explain how.

A fragment of program code might say

```
x = 324.9+sin(theta*pi/2);
```

This is clearly all source code, but the number 324.9 above is likely to be some sort of relevant data, though it might be a physical constant whose value does not depend at all on *this* experiment. The next hard-coded value mentioned in the calculation is difficult to categorise: is the value of π empirical data or is it part of a standard formula? Historically, even π was definitely an empirical value, but today it is a mathematical constant — except in experiments to determine its value empirically. The point is, the distinctions between data, program and even mathematical constants are purely a matter of perspective.

Unfortunately, there is data that is extremely easy to overlook (and therefore is very hard to manage). You may assume that the function `sin`, as used in the calculation example above, is the standard trigonometric function for calculating sines (and because of π, you assume it is taking radians as the parameter type) but almost all programming languages allow `sin` to be any function whatsoever. Confusingly, it is generally a different function when the code is run on a different computer.

It is impossible to tell. More complex code will often have many facts "hard wired" into the code — so in fact the code contains data. Code can even read in formulas from data and compile them to perform further calculations, and so on. Equally, data can control the flow of code. For example, data summarising patients may include their gender, but the program processes males and females differently. Then data starts becoming like code.

Many computer programs blur the distinctions deliberately, to create virtual machines. Data is then run on the virtual machine as program. Many programs provide standard features to do this,

[2] Many of the foundational issues were explored thoroughly by Christopher Strachey and others in the 1960s; Strachey's classic lectures are reprinted in an accessible 2000 publication [81]. Being originally a very old paper this classic introduction is much easier to read than many more recent discussions of the foundations of Computer Science.

such as LISP's and JavaScript's `eval` functions. Henderson's book [65] builds an elegant Pascal program to run *any* LISP program as data, and then shows that the LISP program can run itself running other programs, so it is now its own code and *its* data — despite being purely data to the Pascal program. There are numerous advantages to doing this, including: the Pascal program is not just reading data, but structured data that must conform to the rules of LISP; the LISP running itself runs faster than the original Pascal running LISP, even though the Pascal virtual machine is still doing it in the recursive case; LISP is a much more powerful language than Pascal, so a virtual machine can be used to escape the barriers of a limited implementation such as Pascal. In short, any distinctions between code and data are impossible to maintain.

In the present paper, we knowingly built on this blur between data and code. However, what we did was not unusual except in our explicit and rigorous approach to managing and summarising data reliably in the paper.

The paper and its Supplemental Material are typeset in LaTeX, a popular typesetting language. LaTeX not only has text (as you are reading right now) but it also has code. For example, "LaTeX" was typeset by running the code for a macro called `\LaTeX`, which then calculated how to position the letters as they are wanted. When π was written above, the code that generated what you read actually said `$\pi$` — so is this data that just says π or is it code that tells the computer to change character sets from Latin to Greek, and then uses `\pi` as a program variable name to select a particular glyph from the data about typesetting Greek characters? The distinctions are all a bit moot. In other words, the publication itself is data to a LaTeX program, and within that data it includes further programs. Indeed, LaTeX is run on a virtual machine, in exactly the same way that Henderson's LISP is, and doing so provides the same advantages.

The data for this paper's survey was itself originally written as literal text in LaTeX: it meant that LaTeX could process it to produce a typeset table (as in the Supplemental Material above). As the extent of the data grew, it rapidly became apparent that LaTeX is a poor choice to manage structured data. A simple JavaScript program was written to convert the LaTeX data into JSON (which is much more readable than LaTeX) and also generate CSV files that can be processed in standard office software such as Excel, which some readers may prefer. In fact, examining and comparing the same data in the contrasting formats, this typeset file, in JSON, and in Excel (reading the generated CSV) provided multiple different perspectives of the data that increased redundancy and confidence that the data was correct and correctly handled.

It is important to note that using such techniques is quite routine in science publication, though often pre-existing tools are used to streamline the process (and to ensure that it is more widely understood). The paper [100], for example, in addition to using a typesetting system for publication, also placed its code in a repository using R Markdown [86], a programming environment based on R designed for generating and documenting lab books — almost the polar opposite of LaTeX, which is designed for publication but can be used for programming.

Finally note that what look like magic numbers used throughout the present paper (such as the 32 in "32 papers were evaluated") are all in fact named, calculated and placed *in situ* directly from statistical computations performed on the JSON paper's data.

4 THE SPEIGELHALTER TRUSTWORHINESS QUESTIONS

David Speigelhalter is concerned how statistics is often misused and misunderstood. In his *The Art of Statistics* [40] Speigelhalter brings together his advice for making reliable statistical claims: they need to be accessible, intelligible, assessable, and usable — the claims need to be properly accountable. Speigelhalter proposes ten questions to ask when confronted with any claim based on statistical evidence. Some of his questions are quite general, and might be applied to any sort of scientific claims, but all have analogous questions that could be addressed to software code or papers relying on code — analogues are suggested in **bold** below.

What might seem like dauntingly technical software issues are no more demanding than the basic statistical issues that are regularly acceded to; failing to ask them is as risky as dismissing statistical scrutiny.

4.1 How trustworthy are the numbers?

(1) *How rigorously has the study been done?* For example, check for 'internal validity,' appropriate design and wording of questions, pre-registration of the protocol, take a representative sample, using randomization, and making a fair comparison with a control group.

- ► **How rigorously has the software engineering been done? Section 2 in the Supplemental Material provides a list of important issues that must be addressed for any reliable software.**
- ► **"Internal validity" assumes that there is evidence the programmers had uncertainty in the code's reliability and checked it. Were different methods used and compared, or was all confidence put into a single implementation? What internal consistency checks does the implementation have? Were invariants and assertions defined and checked?**

(2) *What is the statistical uncertainty/confidence in the findings?* Check margins of error, confidence intervals, statistical significance, multiple comparisons, systemic bias.

- ► **How are the claims presented that give us confidence in the code that they are based on? Are there discussions of invariants, independent checks for errors, and so on? Again, Supplemental Material section 2 provides further discussion of such issues.**

(3) *Is the summary appropriate?* Check appropriate use of averages, variability, relative and absolute risks.

- ► **If the claims are exploratory, weaker standards of coding can be used; if the claims are a basis for critical decisions, then there should be evidence of using appropriate software engineering (such as defensive programming) to provide appropriate confidence in the results claimed.**

4.2 How trustworthy is the source?

(4) *How reliable is the source of the story?* Consider the possibility of a biased source with conflicts of interest, and check publication is independently peer-reviewed. Ask yourself, 'Why does this source want me to hear this story?'

- ► **The source of many science stories is the output of running some code. How reliable is this code? What evidence is there that the code was well-engineered so its reliability can be trusted?**

(5) *Is the story being spun?* Be aware of the use of framing, emotional appeal through quoting anecdotes about extreme cases, misleading graphs, exaggerated headlines, big-sounding numbers.

- ► **Be wary of AI and ML which may have been trained by chance or specifically (if not deliberately) to get the results described.**

(6) *What am I not being told?* This is perhaps the most important question of all. Think about cherry-picked results, missing information that would conflict with the story, and lack of independent comment.

- ► **Cherry picking with code is often unconscious and is very common: when running code produces the "cherries" for a paper it is tempting to stop testing**

the code and just assume it is running correctly. So, what evidence is there that the code was rigorously developed and cherry picking avoided?

4.3 How trustworthy is the interpretation?

(7) *How does the claim fit with what else is known?* Consider the context, appropriate comparators, including historical data, and what other studies have shown, ideally in a meta-analysis.
- ► **Is there any discussion of the code and how does it compare with other peer-reviewed publications using code used for similar purposes?**

(8) *What's the claimed explanation for what has been seen?* Vital issues are correlation v. causation, regression to the mean, inappropriate claim that a non-significant result means 'no effect,' confounding attribution, prosecutor's fallacy.
- ► **These are all good statistical questions. The software engineering analogy is: are the claims backed up by a sufficiently detailed discussion of the algorithms and software engineering that justify the appropriateness of the chosen software implementation? The Supplemental Material list in section 2 provides examples of expected explanations for the trustworthiness of running some code.**

(9) *How relevant to the story is the audience?* Think about generalisability, whether the people being studied are special case, has there been an extrapolation from mice to people.
- ► **Generalisability is equivalent to is the code available, easy to understand and use for more general purposes — including further work and checking the reproducibility of the claims being made?**

(10) *Is the claimed effect important?* Check whether the magnitude of the effect is practically significant, and be especially wary of claims of 'increased risk.'

ADDITIONAL REFERENCES FOR SUPPLEMENTAL MATERIAL

References numbered 1–57 appear in the reference list in the main paper.

[58] M. Abir, C. Nelson, E. W. Chan, H. Al-Ibrahim, C. Cutter, K. Patel, and A. Bogart. 2020. *RAND Critical Care Surge Response Tool: An Excel-Based Model for Helping Hospitals Respond to the COVID-19 Crisis.* RAND Corporation. URL www.rand.org/pubs/tools/TLA164-1.html

[59] N. M. Alvarez, E. Gonzalez-Gonzalez, and G. Trujillo-de Santiago. 2020. Modeling COVID-19 epidemics in an Excel spreadsheet: Democratizing the access to first-hand accurate predictions of epidemic outbreaks. *MedRxiv* (2020). URL https://doi.org/10.1101/2020.03.23.20041590 Preprint.

[60] R. Anderson. 2020. *Security Engineering* (3rd ed.). Wiley.

[61] T Baum, H Leßmann, and K Schneider. 2017. The Choice of Code Review Process: A Survey on the State of the Practice. In *Lecture Notes in Computer Science (Product-Focused Software Process Improvement: 18th International Conference, Vol. 10611).* 111–127. URL https://doi.org/10.1007/978-3-319-69926-4_9

[62] Alys Brett, Michael Croucher, Robert Haines, Simon Hettrick, James Hetherington, Mark Stillwell, and Claire Wyatt. 2017. *State of the Nation Report for Research Software Engineers.* Research Software Engineer Network. URL https://doi.org/10.5281/zenodo.495360

[63] D. L. Chao, M. E. Halloran, V. J. Obenchain, and I. M. Longini Jr. 2010. FluTE, a Publicly Available Stochastic Influenza Epidemic Simulation Model. *PLOS Computational Biology* 6, 1 (2010), e1000656. URL https://doi.org/10.1371/journal.pcbi.1000656 Source code available at URL GitHub.com/dlchao/FluTE.

[64] Chris Hanson and Gerald Jay Sussman. 2021. *Software design for flexibility: How to avoid programming yourself into a corner.* MIT Press.

[65] Peter Henderson. 1980. *Functional Programming: Application and Implementation.* Prentice-Hall International.

[66] W. S. Humphrey. 2001. *Winning with Software: An Executive Strategy.* Addison-Wesley Professional.

[67] W. S. Humphrey. 2005. *PSP: A Self-Improvement Process for Software Engineers.* Addison Wesley.

[68] ISO/IEC JTC 1/SC 7 Software and systems engineering Committees. 2015. *Software engineering — Guidelines for the application of ISO 9001:2015 to computer software.* International Organization for Standardization (ISO). URL www.iso.org/standard/74348.html

[69] J. Knight. 2012. *Fundamentals of Dependable Computing for Software Engineers.* CRC Press.

[70] D. E. Knuth. 1992. *Literate programming.* Vol. 27. Center for the Study of Language and Information Publication Lecture Notes.

[71] S. Mason and D. Seng. 2017. *Electronic Evidence* (4th ed.). Humanities Digital Library. URL https://doi.org/10.14296/517.9781911507079 (NB See URL humanities-digital-library.org/index.php/hdl/catalog/book/electronicevidence until DOI is resolved.).

[72] C. Misselhorn. 2018. Artificial Morality. Concepts, Issues and Challenges. *Social Science and Public Policy* 55 (2018), 161–169. URL https://doi.org/10.1007/s12115-018-0229-y

[73] RTCA Committee SC-205. 2011. *DO-178C — Software Considerations in Airborne Systems and Equipment Certification.* RTCA. URL my.rtca.org/NC__Product?id=a1B36000001IcmqEAC

[74] Isaac Z. Schlueter. 23 March 2016. *Blog: kik, left-pad, and npm.* NPM Blog. URL blog.npmjs.org/post/141577284765/kik-left-pad-and-npm Accessed 10 April 2020.

[75] B. Schneier. 2018. *Click Here To Kill Everybody — Security and Survival in a Hyper-connected World.* W. W. Norton & Company, Inc.

[76] B. Shneiderman, C. Plaisant, M. Cohen, and *et al.* 2016. *Designing the User Interface: Strategies for Effective Human-Computer Interaction* (6th ed.). Pearson. URL www.cs.umd.edu/hcil/DTUI6

[77] A. Shostack and M. E. Zurko. 2020. Secure Development Tools and Techniques Need More Research That Will Increase Their Impact and Effectiveness in Practice. *Commun. ACM* 63, 5 (2020), 39–41. URL https://doi.org/10.1145/3386908

[78] Software Sustainability Institute. 2020. Web site. URL software.ac.uk Accessed 3 August 2020.

[79] I. Sommerville. 2015. *Software Engineering* (10th ed.). Pearson.

[80] Susan Stepney, Fiona A. C. Polack, Kieran Alden, Paul S. Andrews, James L. Bown, Alastair Droop, Richard B. Greaves, Mark Read, Adam T. Sampson, Jon Timmis, and Alan F. T. Winfield. 2018. *Engineering Simulations as Scientific Instruments: A Pattern Language.*

[81] C. Strachey. 2000. Fundamental Concepts in Programming Languages. *Higher-Order and Symbolic Computation* 13 (2000), 11–49. URL https://doi.org/10.1023/A:1010000313106

[82] H. Thimbleby. 2007. *Press On: Principles of Interaction Programming.* MIT Press.

[83] H. Thimbleby. 2012. Heedless Programming: Ignoring Detectable Error is a Widespread Hazard. *Software — Practice & Experience* 42, 11 (2012), 1393–1407. URL https://doi.org/10.1002/spe.1141

[84] H. Thimbleby, S. O. Anderson, and Paul Cairns. 1999. A Framework for Modelling Trojans and Computer Virus Infection. *Computer Journal* 41, 7 (1999), 444–458. URL https://doi.org/10.1093/comjnl/41.7.444

[85] Vancouver Group. 1997. Uniform requirements for manuscripts submitted to biomedical journals. *JAMA* 277, 11 (1997), 927–934.
URL https://doi.org/10.1001/jama.1997.03540350077040

[86] Yihui Xie, J. J. Allaire, and Garrett Grolemund. 2020. R Markdown: The Definitive Guide. (2020).

[87] A. Zeller. 2006. *Why Programs Fail: A Guide to Systematic Debugging.* Morgan Kaufmann.

5 SUMMARY OF PILOT SURVEY

5.1 Assessment criteria and methods

A survey sampled of recent papers that were published online in July 2020, accepted for publication after peer review in 3 high-profile, highly competitive leading peer-reviewed journals, namely *Lancet Digital Health* ($N = 6$), *Nature Digital Medicine* ($N = 12$) and *Royal Society Open Science* ($N = 14$). Papers were selected from the journals' July 2020 new online listings where the paper's title implied that code had been used in the research. Commentary, correspondence and editorials were excluded. The sample represents what the editorial and the broader peer review community considers to be good practice.

The selection process will have certainly missed some papers that use code, but the criterion selects papers where the wording of the title indicates that the authors consider code to be a component of the scientific contribution. Indeed, all sampled papers used code in their research. Although there is unavoidable subjectivity in the paper evaluations and uncontrolled bias from using a single evaluator (the author of this paper), it is hoped that using a sample of 32 papers from 3 diverse journals is sufficient to randomise errors so that they largely cancel out, and the overall trends as discussed in this paper are reliable. It should be noted that, except where a paper provides a URL to a code repository, much code was disorganised so possibly not all code was reviewed because it was too hard to find (some emails to authors have not been responded to).

Since almost every scientific paper relies on generic computer code (calculating statistics, plotting graphs, storing and manipulating data, accessing internet resources, etc), the baseline of papers using code was not assessed. Papers whose title indicated their contribution included or relied on bespoke code were selected, and all those clearly relied heavily on their own specifically developed code. Papers that may have relied on bespoke code but whose titles made no such implication were not assessed.

There is considerable debate over what good commenting practice is, but this is because comments have many roles — from helping students to get marks in assessments, asserting intellectual rights, reminding the developer of things to do, managing version control, to explaining the code to third parties. Different programming languages also develop "cultures" that encourage different approaches for comments (examples include R Markdown, Mathematica Notebooks, JavaDoc, Haskell's Haddock, and so on). For scientific code, however, the explanatory role is critical, and this is what was assessed in this survey. Note that comments used for explanation does not preclude any other uses.

The completeness or executability of code was not assessed, although if code was obviously incomplete this was noted above. Whether code runs as claimed is a matter of research integrity, which is beyond the scope of this survey. What is relevant to the study is whether the code is described in sufficient detail that the methods used can be scrutinised. Obviously being able to run the code will help, but clarity in documentation and comments is critical. It is more like "can we see the critical pages from your lab book so we understand what you did?" rather than "can we have free run of your laboratory?"

As an informal survey, intended to establish whether the issues in epidemic modelling were more widespread, and given the very poor level of documentation found in scientific code, it was not felt necessary to have independent or blind assessment.

The data was recorded in JSON (JavaScript object notation), which is a simple standard data format. A JavaScript program sanity checks the JSON data and then converts it to various tables and LaTeX number registers that are used throughout the paper. The strict sanity checks found a few oversights (e.g., if there are comments of any sort there must be some accessible code), which led to a productive double-checking of all the facts of the original papers. For example, some papers that apparently had no code available during the first assessment, presumably causing problems for review, had uploaded code by the time of the double-checking.[3] In turn, a field `doubleChecked` was added to supplement the original data field `accessed` to track the human process of double-checking the data — the double-checking also inevitably confirmed details like the DOI was previously typed correctly. (Previously the DOI had been just copied *into* the database and there had been no necessity to double-check it against clerical errors.)

The JavaScript program generates files from the JSON, with all the definitions; these files were then included in both the main paper and in this Supplemental Material, so when the paper or Supplemental Material is typeset all tables and specific data items are typeset automatically, consistently and reliably by LaTeX. For example, the register `\dataN` is set to the value 32, which is the total number of papers assessed in the JSON data, and the macro `\journalBreakdown` is defined directly from the data to be the following text:

> *Lancet Digital Health* ($N = 6$), *Nature Digital Medicine* ($N = 12$) and *Royal Society Open Science* ($N = 14$)

— which of course is the JavaScript's automatic breakdown of the total $N = 32$ by journal name. The exact same text was also used in the main paper.

An interesting consequence of this automatic approach is that as the author found themselves writing text such as:

> Code repositories were used by 10 papers ...

it motivated extending the JavaScript data processing so that *all* specific quantities mentioned in the paper are traceable directly back to the JSON data. The phrase above is now in fact written in LaTeX in the paper as follows:

```
Code repositories were used by
\plural{\countUsesVersionControlRepository}{paper} …
```

where `\plural` automatically writes a word ("paper" in this case) in singular or plural form as required. When typeset the text above is generated with the relevant number or numbers inserted.

Of course, the variable `countUsesVersionControlRepository` = 10 in this case.

The full JavaScript JSON data and processing code is provided on the repository as described in the main paper.

5.2 Sampled journal code policies

Extract from *Royal Society Open Science* author guidelines

It is a condition of publication that authors make the primary data, materials (such as statistical tools, protocols, software) and code publicly available. These must be provided at the point of submission for our Editors and reviewers for peer-review, and then made publicly available at acceptance. [...] As a minimum, sufficient information and data are required to allow others to

[3]Note that double-checking was performed by the same person as the first assessment, though with the benefit of a six month gap to bring a degree of independence.

replicate all study findings reported in the article. Data and code should be deposited in a form that will allow maximum reuse. As part of our open data policy, we ask that data and code are hosted in a public, recognised repository, with an open license (CC0 or CC-BY) clearly visible on the landing page of your dataset.

URL royalsociety.org/journals/authors/author-guidelines/#data
accessed 29 July 2020.

Extract from *Nature Digital Medicine* author guidelines
A condition of publication in a Nature Research journal is that authors are required to make materials, data, code, and associated protocols promptly available to readers without undue qualifications. [...] A condition of publication in a Nature Research journal is that authors are required to make unique materials promptly available to others without undue qualifications.

URL www.nature.com/nature-research/editorial-policies/reporting-standards#availability-of-data
accessed 29 July 2020.

Extract from *Lancet Digital Health* author guidelines
— Has detailed data policies, but no code policy.

URL marlin-prod.literatumonline.com/pb-assets/Lancet/authors/tldh-info-for-authors.pdf
accessed 29 July 2020.

Extract from *Journal of Vascular Surgery* author guidelines
— Has detailed data policies, but no code policy. While no *Journal of Vascular Surgery* papers were surveyed, the following statement on data policies is relevant for section 6.1 in the main paper:

The authors are required to produce the data on which the manuscript is based for examination by the Editors or their assignees, should they request it. [...] The authors should consider including a footnote in the manuscript indicating their willingness to make the original data available to other investigators through electronic media to permit alternative analysis and/or inclusion in a meta-analysis.

URL www.editorialmanager.com/jvs/account/JVS_Instructions%20for%20Authors2020.pdf
accessed 29 July 2020.

5.3 Assessments and criteria

Legend:

P_c	Journal has a code policy (see section 5.b)
$P_{c\text{-breach}}$	Paper breaches journal code policy (see section 5.b)
R_c	Paper uses a code repository (e.g., GitHub)
$R_{c\text{-empty}}$	Code repository contains no code
R_d	Paper uses a data repository (e.g., Dryad, Figshare, GitHub)
S_{NONE}	No code available at all (note: code is not expected for standard models, systems or statistical methods)
S_p	Paper says source code is available in principle
S_+	Paper or URL provides source code
$S_{rigorous}$	Evidence that source code was developed rigorously
S_{tested}	Evidence that source code has been run with a clean build and tested
S_{make}	Has makefile or other configuration (see section 2.b)
$S_{otherSE}$	Other evidence of good practice; see details in summary table
C_0	Code has no non-trivial comments
C_1	Code only has trivial comments (e.g., copyright)
C_2	Helpful comments explaining code intent, rather than rephrasing the code
C_+	Code has substantial, useful comments and documentation

Ref	Data	Code
88	On request	"Code is available upon request from the corresponding author" (requested) P_c S_p
89	"The datasets used in the current study are available from the corresponding author upon reasonable request and under consideration of the ethical regulations" R_d	Matlab. Documented overview, but only trivial comments P_c R_c S_+ C_1
90	"In accordance with Twitter policies of data sharing, data used in the generation of the algorithm for this study will not be made publicly available"	"Due to the sensitive and potentially stigmatizing nature of this tool, code used for algorithm generation or implementation on individual Twitter profiles will not be made publicly available" P_c $P_{\text{c-breach}}$ S_{NONE}
91	"The datasets generated during and/or analyzed during the current study are available from the corresponding author on reasonable request."“”	"This code would be made available upon reasonable request." (requested) P_c S_p
92	Nothing available	Nothing available (despite building two voice-based virtual counselors) P_c $P_{\text{c-breach}}$ S_{NONE}
93	"The datasets generated and analyzed during the study are not currently publicly available due to HIPAA compliance agreement but are available from the corresponding author on reasonable request"	Poor commenting, no documentation P_c R_c S_+ C_1
94	"The dataset generated and analyzed for this study will not be made publicly available due to patient privacy and lack of informed consent to allow sharing of patient data outside of the research team"	No code available P_c $P_{\text{c-breach}}$ S_{NONE}
95	"The datasets generated during and/or analyzed during the current study are not publicly available due to institutional restrictions on data sharing and privacy concerns. However, the data are available from the corresponding author on reasonable request"	Empty GitHub repository: "Code coming soon..." it says P_c $P_{\text{c-breach}}$ R_c $R_{\text{c-empty}}$ S_{NONE}

Ref	Data	Code
96	“The i2b2 data that support the findings of this study are available from i2b2 but restrictions apply to the availability of these data, which require signed safe usage and research-only. Data from UCSF are not available at this time as they have not been legally certified as being De-Identified, however, this process is underway and the data may be available by the time of publication by contacting the authors. Requesters identity as researchers will need to be confirmed, safe usage guarantees will need to be signed, and other restrictions may apply”	Basic documentation, very little comment P_c R_c S_+ C_1
97	“Not available due to restrictions in the ethical permit, but may be available on request”	Trivial comments, no documentation P_c R_c S_+ C_1
98	“The data that support the findings of this study are available in a deidentified form from Cleveland Clinic, but restrictions apply to the availability of these data, which were used under Cleveland Clinic data policies for the current study, and so are not publicly available”	“We used only free and open-source software” some of which is unspecified P_c $P_{\text{c-breach}}$ S_{NONE}
99	“The i-ROP cohort study data for ROP is not publicly available due to patient privacy restrictions, though potential collaborators are directed to contact the study investigators …”	Not all code on GitHub, minor comments P_c R_c S_+ C_1
100	Data available on Dryad R_d	Code and example runs available in R Markdown P_c S_+ C_+
101	Data directly written into program code	Basic Matlab with routine comments P_c $P_{\text{c-breach}}$ S_+ C_1
102	Data available on Dryad plus publicly available data from the 1000 genomes project. Currently (apparently) for private view R_d	Code available for private view, though some code available with minor comments. Paper describes using two contrasting methods to help confirm correctness P_c S_p S_{otherSE} C_2
103	Data available on Dryad R_d	Reasonaby commented code on Dryad, but code is not complete and presumably never checked P_c S_p C_2
104	On request	R lightly commented P_c S_p C_1
105	No data required	Unrunnable incomplete code fragment P_c $P_{\text{c-breach}}$ S_p
106	Data embedded in PDF	No code available P_c $P_{\text{c-breach}}$ S_{NONE}
107	Data available on Dryad R_d	Some comments, some code in Matlab P_c S_p C_2

Ref	Data	Code
108	Partial data on Dryad R_d	Documented R, including manual P_c R_c S_+ C_+
109	No data required	"We constructed a bioeconomic model for an RSSF [restricted fishing effort small-scale fishery] using game theory" for which results are discussed, yet no code is available P_c $P_{c\text{-breach}}$ S_{NONE}
110	Data cited, not all available	Trivial documentation P_c R_c S_+ C_1
111	On Figshare R_d	On Figshare, large amount of disorganised and undocumented code. Helpful features to make usable for third parties P_c S_+ C_1
112	Data on Dryad R_d	No code available P_c $P_{c\text{-breach}}$ S_{NONE}
113	Data on various web sites	No code available P_c $P_{c\text{-breach}}$ S_{NONE}
114	Data on request	"The coding used to train the artificial intelligence model are dependent on annotation, infrastructure, and hardware, so cannot be released." (!) Algorithm (not source code) available on request. S_{NONE}
115	Data on request	Python scripts can be requested S_p
116	Unspecified location on large website requiring registration R_d	Has overall documentation but poorly commented Matlab code on GitHub R_c S_+ C_1
117	Available to researchers who meet criteria for access to confidential data	Despite the paper being a "deep learning algorithm" the code is not available S_{NONE}
118	Data access conditional on approved study proposal	Almost completely uncommented Python, but does have a basic setup script R_c S_+ C_0
119	Unspecified locations on several large websites	Python used and apparently GitHub, but — an oversight? — no code is available S_{NONE}

5.4 Summary of assessments

Number of papers sampled relying on code	32	100%	
Access to code			
Have some or all code available	12	38%	
Some or all code in principle available on request	8	25%	
No code available	12	38%	
Evidence of basic good software engineering practice			
Evidence program designed rigorously	0	0%	
Evidence source code properly tested	0	0%	
Evidence of makefile or equivalent	0	0%	
Other methods, e.g., independent coding methods	1	3%	
Documentation and comments			
Good code documentation and comments	2	6%	
Basic comments	3	9%	
No or only trivial comments (e.g., copyright)	29	91%	
Repository use			
Code repository (e.g., GitHub) — 1 was empty	10	31%	
Data repository (e.g., Dryad or GitHub)	9	28%	
Adherence to journal code policy (if any)			
Papers published in journals with code policies	26	81%	
Clear breaches of code policy	11	42%	(N = 26)

This is exactly the same table as table 1 from the main paper, reproduced here for convenience. See section 5.5.1 in this Supplemental Material for details of the process that generated it.

E REFERENCES FOR SAMPLED PAPERS

[88] Callahan A, Steinberg E, Fries JA, Gombar S, Patel B, Corbin CK and Shah NH, "Estimating the efficacy of symptom-based screening for COVID-19," *Nature Digital Medicine*, **3**(95), 2020. DOI `10.1038/s41746-020-0300-0`
Accessed 14 July 2020. Double-checked 17 January 2021.

[89] Kanzler CM, Rinderknecht MD, Schwarz A, Lamers I, Gagnon C, Held JPO, Feys P, Luft AR, Gassert R and Lambercy O, "A data-driven framework for selecting and validating digital health metrics: use-case in neurological sensorimotor impairments," *Nature Digital Medicine*, **3**(80), 2020. DOI `10.1038/s41746-020-0286-7` Code URL github.com/ChristophKanzler/MetricSelectionFramework
Accessed 14 July 2020. Double-checked 17 January 2021.

[90] Roy A, Nikolitch K, McGinn R, Jinah S, Klement W and Kaminsky ZA, "A machine learning approach predicts future risk to suicidal ideation from social media data," *Nature Digital Medicine*, **3**(78), 2020. DOI `10.1038/s41746-020-0287-6`
Accessed 14 July 2020. Double-checked 17 January 2021.

[91] Levine DM, Co Z, Newmark LP, Groisser AR, Holmgren AJ, Haas JA and Bates DW, "Design and testing of a mobile health application rating tool," *Nature Digital Medicine*, **3**(74), 2020. DOI `10.1038/s41746-020-0268-9`
Accessed 14 July 2020. Double-checked 17 January 2021.

[92] Kannampallil T, Smyth JM, Jones S, Payne PRO and Ma J, "Cognitive plausibility in voice-based AI health counselors," *Nature Digital Medicine*, **3**(72), 2020. DOI 10.1038/s41746-020-0278-7
Accessed 14 July 2020. Double-checked 17 January 2021.

[93] Huang S, Kothari T, Banerjee I, Chute C, Ball RL, Borus N, Huang A, Patel BN, Rajpurkar P, Irvin J, Dunnmon J, Bledsoe J, Shpanskaya K, Dhaliwal A, Zamanian R, Ng AY and Lungren MP, "PENet a scalable deep-learning model for automated diagnosis of pulmonary embolism using volumetric CT imaging," *Nature Digital Medicine*, **3**(61), 2020. DOI 10.1038/s41746-020-0266-y Code URL github.com/marshuang80/PENet
Accessed 14 July 2020. Double-checked 17 January 2021.

[94] Dhruva SS, Ross JS, Akar JG, Caldwell B, Childers K, Chow W, Ciaccio L, Coplan P, Dong J, Dykhoff HJ, Johnston S, Kellogg T, Long C, Noseworthy PA, Roberts K, Saha A, Yoo A and Shah ND, "Aggregating multiple real-world data sources using a patient-centered health-data-sharing platform," *Nature Digital Medicine*, **3**(60), 2020. DOI 10.1038/s41746-020-0265-z
Accessed 14 July 2020. Double-checked 17 January 2021.

[95] Hofer IS, Lee C, Gabel E, Baldi P and Cannesson M, "Development and validation of a deep neural network model to predict postoperative mortality, acute kidney injury, and reintubation using a single feature set," *Nature Digital Medicine*, **3**(58), 2020. DOI 10.1038/s41746-020-0248-0 Code URL github.com/cklee219/PostoperativeOutcomes_RiskNet
Accessed 14 July 2020. Double-checked 17 January 2021.

[96] Norgeot B, Muenzen K, Peterson TA, Fan X, Glicksberg BS, Schenk G, Rutenberg E, Oskotsky B, Sirota M, Yazdany J, Schmajuk G, Ludwig D, Goldstein T and Butte AJ, "Protected Health Information filter (Philter): accurately and securely de-identifying free-text clinical notes," *Nature Digital Medicine*, **3**(57), 2020. DOI 10.1038/s41746-020-0258-y Code URL github.com/BCHSI/philter-ucsf
Accessed 14 July 2020. Double-checked 17 January 2021.

[97] Choi D, Park JJ, Ali T and Lee S, "Artificial intelligence for the diagnosis of heart failure," *Nature Digital Medicine*, **3**(54), 2020. DOI 10.1038/s41746-020-0261-3 Code URL github.com/ubiquitous-computing-lab/AI-CDSS-Cardiovascular-Silo
Accessed 14 July 2020. Double-checked 17 January 2021.

[98] Hilton CB, Milinovich A, Felix C, Vakharia N, Crone T, Donovan C, Proctor A and Nazha A, "Personalized predictions of patient outcomes during and after hospitalization using artificial intelligence," *Nature Digital Medicine*, **3**(51), 2020. DOI 10.1038/s41746-020-0249-z
Accessed 14 July 2020. Double-checked 17 January 2021.

[99] Li MD, Chang K, Bearce B, Chang BY, Huang AJ, Campbell JP, Brown JM, Singh P, Hoebel KV, Erdoğmuş D, Ioannidis S, Palmer W, Chiang MF and Kalpathy-Cramer J, "Siamese neural networks for continuous disease severity evaluation and change detection in medical imaging," *Nature Digital Medicine*, **3**(48), 2020. DOI 10.1038/s41746-020-0255-1 Code URL github.com/QTIM-Lab/SiameseChange
Accessed 14 July 2020. Double-checked 19 January 2021.

[100] Hoffman JI, Nagel R, Litzke V, Wells DA and Amos W, "Genetic analysis of *Boletus edulis* suggests that intra-specific competition may reduce local genetic diversity as a woodland ages," *Royal Society Open Science*, **7**(200419), 2020. DOI `10.1098/rsos.200419` Code URL datadryad.org/stash/dataset/doi:10.5061/dryad.1g1jwstrw
Accessed 22 July 2020. Double-checked 26 January 2021.

[101] Grönquist P, Panchadcharam P, Wood D, Menges A, Rüggeberg M and Wittel FK, "Computational analysis of hygromorphic self-shaping wood gridshell structures," *Royal Society Open Science*, **7**(192210), 2020. DOI `10.1098/rsos.192210` Code URL royalsocietypublishing.org/doi/suppl/10.1098/rsos.192210
Accessed 22 July 2020. Double-checked 26 January 2021.

[102] Amos W, "Signals interpreted as archaic introgression appear to be driven primarily by faster evolution in Africa," *Royal Society Open Science*, **7**(191900), 2020. DOI `10.1098/rsos.191900` Code URL datadryad.org/stash/share/ichHKrWj7hqlznOaR6NQVzITgp40dlqWvWAgAxyafiQ
Accessed 22 July 2020. Double-checked 26 January 2021.

[103] Gordon M, Viganola D, Bishop M, Chen Y, Dreber A, Goldfedder B, Holzmeister F, Johannesson M, Liu Y, Twardy C, Wang J and Pfeiffer T, "Are replication rates the same across academic fields? Community forecasts from the DARPA SCORE programme," *Royal Society Open Science*, **7**(200566), 2020. DOI `10.1098/rsos.200566` Code URL royalsocietypublishing.org/doi/suppl/10.1098/rsos.200566
Accessed 22 July 2020. Double-checked 26 January 2021.

[104] Evans D and Field AP, "Predictors of mathematical attainment trajectories across the primary-to-secondary education transition: parental factors and the home environment," *Royal Society Open Science*, **7**(200422), 2020. DOI `10.1098/rsos.200422` Code URL osf.io/a5xsz/?view_only=87ae173f775b40d79d6cd0fdcf6d4a9c
Accessed 22 July 2020. Double-checked 26 January 2021.

[105] Beale N, Battey H, Davison AC and MacKay RS, "An unethical optimization principle," *Royal Society Open Science*, **7**(200462), 2020. DOI `10.1098/rsos.200462`
Accessed 22 July 2020. Double-checked 26 January 2021.

[106] Cherevko AA, Gologush TS, Petrenko IA, Ostapenko VV and Panarin VA, "Modelling of the arteriovenous malformation embolization optimal scenario," *Royal Society Open Science*, **7**(191992), 2020. DOI `10.1098/rsos.191992`
Accessed 22 July 2020. Double-checked 26 January 2021.

[107] Soczawa-Stronczyk AA and Bocian M, "Gait coordination in overground walking with a virtual reality avatar," *Royal Society Open Science*, **7**(200622), 2020. DOI `10.1098/rsos.200622` Code URL datadryad.org/stash/dataset/doi:10.5061/dryad.vx0k6djnr
Accessed 22 July 2020. Double-checked 26 January 2021.

[108] Duruz S, Vajana E, Burren A, Flury C and Joost S, "Big dairy data to unravel effects of environmental, physiological and morphological factors on milk production of mountain-pastured Braunvieh cows," *Royal Society Open Science*, **7**(200638), 2020. DOI `10.1098/rsos.200638` Code URL github.com/SolangeD/lactModel
Accessed 22 July 2020. Double-checked 26 January 2021.

[109] de Azevedo EZD, Dantas DV and Daura-Jorge FG, "Risk tolerance and control perception in a game-theoretic bioeconomic model for small-scale fisheries," *Royal Society Open Science*, **7**(200621), 2020. DOI 10.1098/rsos.200621
Accessed 22 July 2020. Double-checked 26 January 2021.

[110] Abdolhosseini-Qomi AM, Jafari SH, Taghizadeh A, Yazdani N, Asadpour M and Rahgozar M, "Link prediction in real-world multiplex networks via layer reconstruction method," *Royal Society Open Science*, **7**(191928), 2020. DOI 10.1098/rsos.191928 Code URL github.com/UT-NSG/LRM
Accessed 22 July 2020. Double-checked 26 January 2021.

[111] Webster J and Amos M, "A Turing test for crowds," *Royal Society Open Science*, **7**(200307), 2020. DOI 10.1098/rsos.200307 Code URL figshare.com/collections/Supplementary_information_for_Webster_J_and_Amos_M_A_Turing_Test_for_Crowds_/4859118/1
Accessed 22 July 2020. Double-checked 26 January 2021.

[112] Zhu Y-l, Wang C-J, Gao F, Xiao Z-x, Zhao P-l and Wang J-y, "Calculation on surface energy and electronic properties of CoS_2," *Royal Society Open Science*, **7**(191653), 2020. DOI 10.1098/rsos.191653
Accessed 22 July 2020. Double-checked 26 January 2021.

[113] Yu B, Scott CJ, Xue X, Yue X and Dou X, "Derivation of global ionospheric Sporadic E critical frequency (f_oEs) data from the amplitude variations in GPS/GNSS radio occultations," *Royal Society Open Science*, **7**(200320), 2020. DOI 10.1098/rsos.200320
Accessed 22 July 2020. Double-checked 26 January 2021.

[114] Joon-myoung K, Younghoon C, Ki-Hyun J, Soohyun C, Kyung-Hee K, Seung D B, Soomin J, Jinsik P and Byung-Hee O, "A deep learning algorithm to detect anaemia with ECGs: a retrospective, multicentre study," *Lancet Digital Health*, **2**(7):e358–67, 2020. DOI 10.1016/S2589-7500(20)30108-4
Accessed 24 July 2020. Double-checked 26 January 2021.

[115] Zhu H, Cheng C, Yin H, Li X, Zuo P, Ding J, Lin F, Wang J, Zhou B, Li Y, Hu S, Xiong Y, Wang B, Wan G, Yang X and Yuan Y, "Automatic multilabel electrocardiogram diagnosis of heart rhythm or conduction abnormalities with deep learning: a cohort study," *Lancet Digital Health*, **2**(7):e348–57, 2020. DOI 10.1016/S2589-7500(20)30107-2
Accessed 24 July 2020. Double-checked 26 January 2021.

[116] Fung R, Villar J, Dashti A, Ismail LC, Staines-Urias E, Ohuma EO, Salomon LJ, Victora CG, Barros FC, Lambert A, Carvalho M, Jaffer Y A, Noble JA, Gravett MG, Purwar M, Pang R, Bertino E, Munim S, Min AM, McGready R, Norris SA, Bhutta ZA, Kennedy SH, Papageorghiou AT and Ourmazd A, "Achieving accurate estimates of fetal gestational age and personalised predictions of fetal growth based on data from an international prospective cohort study: a population-based machine learning study," *Lancet Digital Health*, **2**(7):e368–75, 2020. DOI 10.1016/S2589-7500(20)30131-X Code URL github.com/ki-analysis/manifold-ga
Accessed 24 July 2020. Double-checked 26 January 2021.

[117] Sabanayagam C, Xu D, Ting DSW, Nusinovici S, Banu R, Hamzah H, Lim C, Tham Y-C, Cheung CY, Tai ES, Wang XY, Jonas JB, Cheng C-Y, Lee ML, Hsu W and Wong TY, "A deep learning algorithm to detect chronic kidney disease from retinal photographs in community-based populations," *Lancet Digital Health*, **2**(7):e295–302, 2020. DOI 10.1016/S2589-7500(20)30063-7
Accessed 24 July 2020. Double-checked 26 January 2021.

[118] Monteiro M, Newcombe VF, Mathieu F, Adatia K, Kamnitsas K, Ferrante E, Das T, Whitehouse D, Rueckert D, Menon DK and Glocker B, "Multiclass semantic segmentation and quantification of traumatic brain injury lesions on head CT using deep learning: an algorithm development and multicentre validation study," *Lancet Digital Health*, **2**(7):e314–22, 2020. DOI `10.1016/S2589-7500(20)30085-6` Code URL github.com/biomedia-mira/blast-ct
Accessed 24 July 2020. Double-checked 27 January 2021.

[119] Liu K-L, Wu T, Chen P-T, Tsai Y M, Roth H, Wu M-S, Liao W-C and Wang W, "Deep learning to distinguish pancreatic cancer tissue from non-cancerous pancreatic tissue: a retrospective study with cross-racial external validation," *Lancet Digital Health*, **2**(7):e303–13, 2020. DOI `10.1016/S2589-7500(20)30078-9`
Accessed 24 July 2020. Double-checked 27 January 2021.