HPLS (2020) 42:54
https://doi.org/10.1007/s40656-020-00347-6

ORIGINAL PAPER

Software engineering standards for epidemiological models

Jack K. Horner[1] · **John F. Symons**[1]

Received: 16 June 2020 / Accepted: 18 October 2020 / Published online: 10 November 2020

Abstract There are many tangled normative and technical questions involved in evaluating the quality of software used in epidemiological simulations. In this paper we answer some of these questions and offer practical guidance to practitioners, funders, scientific journals, and consumers of epidemiological research. The heart of our paper is a case study of the Imperial College London (ICL) covid-19 simulator, set in the context of recent work in epistemology of simulation and philosophy of epidemiology.

Keywords COVID-19 · Public-health policy · Simulation · Software engineering

1 Introduction

There are many tangled normative and technical questions involved in evaluating the quality of software used in epidemiological simulations. In this paper we answer some of these questions and offer practical guidance to practitioners, funders, scientific journals, and consumers of epidemiological research. The heart of our paper is a case study in which we provide an analysis of the Imperial College London (ICL) covid-19 simulator. This simulator has been extensively used by the United Kingdom to help formulate public-health policy; it has been used to a somewhat lesser extent in public-health policy decision-making in the United States. Developed primarily to predict the effects of public-health interventions such as shutdowns,

✉ Jack K. Horner
jhorner@cybermesa.com

John F. Symons
johnfsymons@gmail.com

[1] Department of Philosophy, University of Kansas, Lawrence, KS, USA

quarantines, social-distancing, and the administration of vaccines, it can be viewed as a complicated data-driven variant of the "susceptible-infected-recovering" (SIR) family of epidemiological models, first described in Kermack and McKendrick (1927).

Our case study, combined with reflection on the state of the art in the philosophy of epidemiology and the ethics of engineering, serves as the basis for our recommendations for future epidemiological modeling projects. We contend that epidemiological simulators should be engineered and evaluated according to a set of public norms. We take as our model for the kinds of norms that we regard as appropriate, the framework of safety–critical standards developed by consensus of the software engineering community for applications such as automotive and aircraft control systems. To achieve that goal, the development and use of epidemiological simulators must have high levels of transparency, explainability, and reproducibility for stakeholders. Furthermore, we recommend that such standards be mandated by funding agencies for epidemiological contexts that have direct and significant public policy implications.

The structure of our argument is straightforward. In Sect. 2 we explain some recent work on the role computation in the philosophy of epidemiology. In Sect. 3 we highlight relevant research on the epistemology of computational modeling and simulation. In Sect. 4, we introduce a development framework that has evolved over the past four decades in the software engineering community. The purpose of this framework has been to provide a principled approach to balancing development cost and schedule against the possible harms of using software in high-risk venues. In Sect. 5, we evaluate, within the framework introduced in Sect. 4, the ICL covid-19 simulator. Our study of this simulator demonstrates that it does not satisfy the standards for safety–critical software in established industry and government practice.

We recognize that the ICL simulator has been subject to intense critical scrutiny because of its role in government decision making during the covid-19 pandemic, and our purpose here is not to pile additional criticism on the work of the ICL team. In this paper, we focus solely on the publicly available artifacts associated with the simulator. We do not assess the empirical assumptions or epidemiological methodology employed by the ICL team. Instead, we hope that by carefully considering this high-profile epidemiological simulator, we can encourage scientific and philosophical communities to reflect on the norms governing the engineering of scientific software in a wide range of important contexts.

Philosophers of science are beginning to understand the trade-offs that are at play in computational science and are becoming increasingly sensitive to the implications of software-intensive scientific inquiry for traditional issues in philosophy of science (Symons and Horner 2014). Philosophers have also recognized that we cannot understand appropriate norms for scientific practice solely by reasoning about them a priori—we must take empirical evidence and technical constraints into account. Most relevantly for this paper, for example, we cannot arrive at norms for evaluating the correctness of software without attention to the actual constraints facing software engineers. Developers of any large piece of software cannot escape practical and theoretical constraints on error correction. These constraints are not discoverable from the philosophical armchair (Horner and Symons 2019; Symons and Horner

2019). Consequently, determining appropriate software standards for epidemiological simulators requires an ongoing interdisciplinary effort.[1]

While we rely on existing standards and norms in software engineering in our analysis, we are not insisting that the norms governing high-risk engineering contexts should be mapped directly and uncritically onto epidemiological modeling projects. However, we will argue that the norms governing epidemiological practice are not solely a matter for epidemiologists themselves. In addition to questions of social value and moral responsibility, epidemiological simulators are often informed or constrained by a range of epistemic, mathematical, economic, and technological considerations. It is clear that many aspects of epidemiological modeling already fall outside of the expertise of epidemiologists. The standards governing the development of important epidemiological simulators cannot be left solely to practitioners. The development of these standards requires careful normative reasoning that is often beyond their expertise.

In addition to falling outside of the epidemiologists' area of scientific specialization, normative deliberation governing scientific inquiry involve balancing the interests of the practitioners themselves with those of others. Standards for the development of epidemiological simulators must accordingly accommodate a complex set of stakeholders. We must balance the interests of the producers and consumers of epidemiological research along with the interests of the broader communities that are affected by the public policy decisions influenced by this research. In the fraught context of epidemiology this task inevitably involves balancing competing social values.

2 Some recent work on the role of computation in the philosophy of epidemiology

Computing technologies have played an important role in medical and biological practice in economically developed societies since at least the 1960s (See e.g. Keller 2002). Evaluating the role and usefulness of data-driven computational models and simulations is complicated in biological contexts for reasons others have explored in detail (See e.g. Leonelli 2011, 2012, 2016; Stevens 2017 for example). Epidemiology is an even more challenging context for evaluating the role of computational models and simulations than, for example, in molecular biology for reasons we will discuss below. More generally, as many philosophers have noted, computational models in biology have distinctive technical and epistemological features that make them uniquely difficult to assess.[2] While philosophers have addressed the role of data science and computational modeling in biological contexts for two decades there has been relatively little scholarly attention given to the norms governing

[1] Much of the philosophical literature about software does not consider the philosophical import of general engineering constraints on software. See Sect. 2 for further detail.

[2] See for example López-Rubio and Ratti (2019) for a discussion of the trade-offs between mechanistic explanation and prediction in applications of machine learning to molecular biology.

the engineering practices underlying these models. Likewise, while philosophers of epidemiology have correctly emphasized the normative and political aspects of research in epidemiology, they have largely neglected the norms guiding engineering practices in the development of epidemiological models and simulations. Since models and simulations are fundamental to epidemiological predictions and recommendations, they should also be subject to critical scrutiny.

In the context of epidemiology, computational modeling and simulation techniques have become indispensable research tools (See Smolinski et al. 2003). As Boschetti and colleagues have noted, computational modeling is sometimes our only way of advancing scientific inquiry in contexts where ethical considerations or practical constraints prevent the use of traditional experimental techniques (Boschetti et al. 2012). In epidemiology, complicated simulations and the manipulation of large data sets, along with the ethical and practical obstacles to experimentation have meant that computational methods have become centrally important research tools.

During the COVID-19 pandemic in particular it was widely reported that the results of computer simulations provided by the ICL team weighed decisively in public policy deliberations in both the United Kingdom and United States governments (Landler and Castle 2020). Government officials are reported to have relied on epidemiological modeling and simulation to predict mortality due to the virus and to anticipate its effects on the healthcare system. These simulations are also used to assess the relative merit of alternative interventions and public health responses to the pandemic (Freedman 2020).[3] In an emergency decision making context, it is reasonable to turn to acknowledged experts on the relevant topics and throughout the pandemic, political decision makers in the United Kingdom government have been eager to present their decisions as grounded in the best available scientific evidence (UK Government 2020). The extent to which decision makers have or have not 'followed the science' has become a fraught and highly politicized matter in many democratic societies (Stevens 2020; Sharma 2020).

In these discussions it is often mistakenly assumed that policy is fully determined by our best epidemiology. As we explain below, this assumption involves a misunderstanding of both the nature of epidemiology and its proper role in decision-making in democratic societies. Difficult trade-offs between different kinds of societal values and moral obligations will not, generally, be resolved by scientific expertise. Epidemiologists cannot tell us, for example, in the case of the COVID-19 epidemic whether public health interventions ought to value the well-being or education of children more highly than reducing the health risks to the elderly. These are moral and political decisions that are not illuminated directly by increased scientific understanding or better models and simulations. There is, moreover, an obvious dependence of deliberations in epidemiological contexts on the reliability of software tools that help to inform those deliberations: we cannot make software-informed ethical

[3] Perhaps the most important policy role of these simulations has been their perceived predictive power. For a discussion of the predictive role of computational models see Boschetti and Symons (2011) and Symons and Boschetti (2013). See Ioannidis et al. (2020) for an assessment of the predictive power of prominent covid-19 modeling efforts to date.

decisions unless the software is doing what we think it should be doing. All else being the same, to achieve that end, we want to minimize, as far as is practical, the frequency of errors in epidemiological software.

The role and status of computer simulations frequently figures, albeit unsystematically, in debates about what it means for governments and institutions to 'follow the science'. It is clear from reporting and from the actions of the United Kingdom government that the simulation results provided by the ICL team were decisively important in policy deliberations in March and April of 2020 (Landler and Castle 2020).[4] It is also clear that the ICL group occupied a high position of scientific authority and trust from the perspective of political decision makers.

'Follow the science' presumably means 'follow the best science'. However, determining which epidemiological recommendation is best is not a straightforward matter. Given the complexity of the factors relevant to decision making during a pandemic, the public in democratic societies and their political representatives have placed great trust in the community of epidemiologists. This is understandable, but often, public declarations of trust have implicitly projected an idealized and unrealistic level of neutrality and objectivity onto epidemiological research. This runs counter to our best critical understanding as drawn from the history and philosophy of epidemiology. As we will explain below, disagreement among epidemiologists can stem from differences with respect to values (Stevens 2020).[5]

The way we have assigned trust to the epidemiological community is not unreasonable, but it involves oversimplification that can lead us to misunderstand our responsibilities as consumers of their research. We are operating with something like the following commonsense understanding of the relationship between scientific expertise and policy making:

> Commonsense view of scientific evidence as a guide to policy making
>
> Decisions that involve risk of serious harm require us to deliberate as carefully as is feasible. Policy makers often have to rely on expert advice since our best available evidence and guidance for decision-making in many matters comes from scientific experts. In such contexts, it is usually rational to follow the advice of the relevant scientific community in order to increase the likelihood that our decisions promote our values and interests. Commonsense recognizes that natural science cannot tell us what we ought to value or what our policy goals ought to be. Nevertheless, under ideal circumstances science can provide an understanding of the facts in a way that helps us to act consistently with the moral or political principles we share.

[4] In mid-March, the ICL model was predicting that absent any public health interventions, the UK would suffer half a million deaths from COVID-19 (2 million deaths in the U.S.).

[5] See Stevens (2020) for a discussion of the confusion around 'follow the science' rhetoric in UK policy making. He writes: "A provisional and contested set of statements about how the world is cannot be used directly as a rule for what governments should do. Ministers have to decide for themselves. They must take responsibility for these decisions and their own inevitable mistakes, rather than relying on science as if it were an apolitical and indisputable tablet of stone." https://doi.org/10.1038/s41562-020-0894-x.

Much in the commonsense view is in our view correct. However, it draws a sharp distinction between social values and norms and scientific inquiry in a way that is especially difficult to achieve in the case of epidemiology for reasons we will explain below. The assumption that epidemiology is value neutral makes our insistence on the importance of high standards for scientific software seem like an unwarranted intrusion on scientific practice. However, there are degrees of neutrality in the sciences when it comes to values. For example, when one turns to an epidemiologist for advice, one cannot be as confident of the relatively value neutral nature of their scientific judgment as one would be in discussions with a chemist.[6] The history and philosophy of epidemiology have highlighted the complex moral and political landscape of the study of epidemics. In this context the standards governing how software for simulations ought to be applied are similarly complex.

Disputes within epidemiology involve normative considerations in ways that disputes between chemists or physicists, for example, almost never do. Consider debates concerning the social determinants of health, where disputants may offer causal stories about the origins of some public health concern that assume, or are motivated at least in part by their preferred socio-political values.[7] In epidemiology, social, political, and other considerations are difficult to disentangle from the manner in which scientific questions are framed. The way epidemiologists think about causation, agency, possible interventions, relevant populations, risk, disease, and responsibility, are all informed by the values governing their practice.[8]

Public reflection on norms is relevant for the practice and not just the application of epidemiology. In order to explain why, consider a disease like type-2 diabetes. There are interventions that would be effective in stopping the spread of this disease that we would regard as unconscionable violations of individual autonomy, or that most of us would presently regard as contrary to the ultimate goals of public health.[9] For example, we might reject heavy taxation on calorie dense foods, mandatory exercise programs, etc. as possible responses to the disease because of the importance of other kinds of social goods. Generally speaking, the set of acceptable interventions that are given scientific consideration will be shaped by a range of social values.

[6] There are philosophers of science who will object here, pointing to the role of value considerations in all sciences, even in chemistry. Our point here is not to say that chemistry is perfectly neutral with respect to value considerations. We are simply noting that there are degrees to which social values, political considerations, etc. play a role in science. It is a mistake to say that because there is some political or social element to all sciences, there is no difference of degree.

[7] See Broadbent (2012) for a discussion of causal reasoning in epidemiology.

[8] As mentioned above, for example, during the early stages of the COVID-19 crisis, harm to the education of the young and risks to the life of very elderly people were weighed against one another without a great deal of explicit public deliberation. The assumptions about social priorities that motivated school closures and other interventions that harmed children and young people may well be defensible. The kinds of interventions that were attempted in the early stages of the pandemic are all defensible given some set of social values. In most cases, epidemiologists did not engage in explicit and public deliberation concerning their presuppositions about social values when they offered their initial recommendations with respect to interventions.

[9] See Tabish et al. (2007) for a defense of categorizing diabetes as an epidemic.

In addition to disagreeing with respect to what would count as an acceptable intervention in public health, social groups may also disagree over what kinds of health issues should be classified as diseases or as epidemics. There is considerable disagreement over, for example, the claim that common mental health problems like anxiety and depression should be regarded as epidemics.[10] Claims that obesity or attention deficit hyperactivity disorder are at epidemic levels in the United States, for example, are difficult to state categorically without reference to a large set of controversial normative assumptions. Ultimately, social values are negotiable. People with differing values can attempt to persuade one another with respect to the relative importance of conflicting values. Given the role of social values in determining the space of acceptable public health interventions, the characterization of health, and the taxonomy of disease, epidemiological inquiry will always be situated within a particular social context and cannot be entirely neutral with respect to normative questions.

Attempts to characterize the subject matter of epidemiology will also generally require reference to concepts that have normative features. Mathilde Frérot and colleagues surveyed the literature from 1978 to 2017 in order to determine the ways that epidemiologists understand their enterprise and how that understanding has changed through time. They examine 102 definitions of 'epidemiology' and found that five terms were present in more than 50% of definitions: "population", "study", "disease", "health" and "distribution" (Frérot et al. 2018). Philosophers of epidemiology have noted that definitions of epidemiology will vary depending on the social and political contexts involved. In their introduction to the recent *Synthese* volume on philosophy of epidemiology, Jonathan Kaplan and Sean Valles emphasize this contested nature of epidemiology (Kaplan and Valles 2019).[11] They contend that "since the welfare of populations and communities are always at stake in epidemiology, the issues at hand are directly or indirectly political issues" (Kaplan and Valles 2019).

Our task in this paper is to encourage attention to the norms governing software engineering in epidemiology. The significance of these models for policy decisions that affect many of us in significant ways is clear. Given that epidemiology is not neutral with respect to social values, non-practitioners have a right and an interest to concern themselves with the standards governing software engineering in this discipline. In addition, funders, journals, policy makers, and the broader public

[10] See Baxter et al. (2014) for an argument against considering common mental health problems like anxiety and depression as epidemics.

[11] They contrast what they see as the divergence between the views of the World Health Organization and the United States Centers for Disease Control. While they do not provide evidence for divergence between these two organizations, they do note two conflicting characterizations of epidemiology, both of which are drawn from Dicker et al. (2006): "Epidemiology is the study of the distribution and determinants of health-related states or events in specified populations, and the application of this study to the control of health problems." (2006, I-1) and later in the same document: "in epidemiology, the 'patient' is the community" (2006, I–4). See Frérot et al. (2018) for a careful empirical assessment of the variety of ways that epidemiology has been characterized in recent decades.

are entitled to require standards are sufficiently high to ensure that simulations are trustworthy.

In the next section we discuss some of the most important epistemic aspects of trustworthiness for computer simulations. As we shall argue, part of determining the standards for what count as good software engineering practice will be determined by the level of risk involved in the deployment of the simulation.

3 The epistemology of epidemiological computer simulations

There has been significant public interest in the epistemic trustworthiness of epidemiological modeling efforts.[12] Most criticisms have raised doubts concerning the assumptions and the quality of the data that go into the models rather than with respect to the quality of the software underlying simulations. Our focus in the following is on properties of the software as software, rather than the scientific status of the assumptions, the mathematical models, or the quality of the data driving these simulations. Critics have occasionally pointed to weaknesses in the publicly available code for simulators (Lewis 2020).[13] We will address some of these criticisms below, Public concern over the status of computational modeling has also involved more abstract epistemological themes. For example, Kreps and Kriner (2020) note that populist media figures have cast doubt on computational modeling as a scientific enterprise. In addition, the influential Fox News personality Tucker Carlson has asserted that computational models of Covid-19 are "completely disconnected from reality" (Sider and Ward 2020). In recent years philosophers have addressed some of the central epistemic problems associated with computer simulations in science.[14] Epidemiological simulations are subject to all such problems, and in this section, we review some of this philosophical literature.

One standard approach to understanding why scientific communities come to trust simulations relies on an analogy with the ways that epistemic entitlements work in other less controversial forms of inquiry (see for example Barberousse and Vorms 2014). In ordinary life, for example, we are generally entitled to trust the testimony of other people, the reliability of our senses, and the capacity of our basic cognitive faculties, such as our memory to transmit information

[12] Kreps and Kriner (2020) provide an assessment of public trust in covid-19 models, highlighting the role of uncertainty and the revision of models in deterioration of public trust. Through a series of experimental surveys, they attempt to show how the shifting scientific consensus can be reconciled with public trust in epidemiology. In their discussion, they present many examples of the ways in which media and political actors cast doubt on modeling. Notably, some of the most corrosive criticisms blend attacks on the empirical assumptions driving the models with broadly philosophical criticisms of computational models as "completely disconnected from reality". For more on this line of criticism in populist media and politics see Sider and Ward (2020).

[13] There has been considerable popular attention to the issue of the quality of code in epidemiological simulations during the COVID-19 pandemic. The quality of these analyses is mixed, for a flavor of some of the commentary see for example Lewis (2020).

[14] See Juan Durán (2018) and Winsberg (2019) for an overview of the epistemic issues related to computer simulation.

without altering it in epistemically significant ways. This use of the idea of epistemic entitlement, largely drawn from Tyler Burge's (1993; 1998) arguments, has been highly influential among philosophers in debates over the epistemology of computer simulation (Barberrouse and Vorms 2014; Beisbart 2017). Symons and Alvarado disagree, arguing instead that the analogical account of epistemic warrants is not appropriate in the context of computer simulations. They have insisted on epistemic standards of the kind we apply to traditional scientific instruments (Symons and Alvarado 2019; Alvarado 2020). On this view, computer simulations are not experts and should not be treated as such. Instead, they are built by teams of experts or by experts working alone who may not be expert software engineers. Thus, given the interdisciplinary integration necessary in a team, the use of the analogy with trusting experts is inappropriate. The analogy is even less fitting in the specific case of epidemiological simulation than it is in science more generally, given that epidemiology relies on interdisciplinary teams with distinct sets of disciplinary standards. Furthermore, the resulting simulations are heavily mediated by what Eric Winsberg called motley practices (Winsberg 2010).

The fact that we are not able to trust computer simulations by analogy with the manner in which we trust individual scientific experts leaves us with the problem of how policy makers and the public should decide which simulations and which models to rely upon. There are many dimensions to this challenge, and it is beyond the scope of this paper to address this broader problem (see, for example, Symons and Alvarado 2019). Trusting simulations involves many complicated criteria. However, for the remainder of this paper we will argue that at least one obvious and necessary condition for justifiable use of simulations for public policy is that they be funded, managed, specified, designed, implemented, and maintained in accordance with the best available software engineering practices, in order to help minimize, as much as practical, the frequency of the kinds of errors that occur in all software development, regardless of application. We contend in this paper that these practices are as important to epidemiological policy-making as good experimental methods are in non-software-intensive scientific regimes. And not least, sound decision-making in epidemiological contexts that depends on epidemiological simulators must be able to assume that errors in that software have been minimized as much as practical. Among other things, that minimization requires adhering to software engineering practices that have (empirically) been shown to help minimize software error.

Our recommendations will, in the near term, increase the cost of these simulation efforts and will require increased collaboration between scientists and software engineers. However, we contend that the risks involved in decisions based on epidemiological modeling efforts warrant the additional resources and effort that we recommend here. During the Covid-19 pandemic it has become clear that public trust in epidemiology is undermined by the perception that its simulators are not developed according to the kind of rigorous standards that we expect in traditional scientific practice. In the following section we explore standards that can help ensure that simulators are not only trusted, but also trustworthy.

4 Standards for software engineering

As with all aspects of epidemiology, engineering standards governing the development of simulations are a matter where normative considerations overlap with technical and mathematical constraints. Because of this, critical scrutiny of these simulations is not the exclusive purview of any subset of scientific experts as we argued above. Practical guidelines for developers of scientific software are described and defended below. We did not invent these standards. Instead, our recommendations draw upon the history of software engineering. For the past five decades, the software engineering community has sought to codify practices and procedures that have been empirically determined to help minimize development cost, schedule, and risk (Boehm 1973; Myers 1976; Boehm, Abts, Brown, Chulani, Clark, Horowitz et al. 2000). This effort has produced an evolving series of software engineering standards, one of the most recent of which is ISO 2017.

We note that scientific-inquiry software is generally not developed according to standards as demanding as those required for safety–critical software by ISO 2017. Thus, our recommendations will be controversial and may be regarded as excessively restrictive for those who view epidemiology solely in terms of scientific inquiry.

Perhaps the most controversial feature of our proposal is the application of an approach drawn from engineering ethics to a discipline that primarily regards itself as scientific inquiry.[15] While the responsible practice of engineering is generally sensitive to the harms involved in various projects (Roddis 1993; Lynch and Kline 2000), one might argue that a science like epidemiology is different. The ethics of scientific inquiry, that argument would contend, are very different from the ethics of engineering. Most philosophers of science are likely to agree. The kinds of simulations that epidemiologists have produced have been regarded by philosophers, for example, as either formal or abstract objects or as special forms of experiments capable of yielding empirical information about the systems they simulate.[16] This view of simulators (more generally, software), however, by fiat ignores a wide range of general "engineering" issues that directly bear on the reliability and trustworthiness of simulators. For example, questions about how we can help to ensure that a developer of a component in a simulator clearly understands how the software h/she develops integrates with software written by others, and what documentation programming, and verification practices help to maximize reliability, do not arise if we consider simulator software to be an abstract object or a special form of experiment. Following Alvarado (2020) we believe that in addition to serving as formal

[15] Epidemiologists sometimes present their work as a basic science for clinical practice in medicine. See eg. Sackett et al. (1985) and Bonita et al. (2006).

[16] Weisberg (2012) and Pincock (2011) regard computer models as formal extension of mathematical representation. Morrison (2009, 2015) regards computer simulations as being a form of scientific experimentation (Ruphy 2015). Morrison and others have argued that computer simulations involve extra-mathematical considerations (Winsberg 2010). These include measurement practices (Morrison 2009), representations and imaging (Barberousse et al. 2014), and hypothesis testing and generation (Hartmann and Frigg 2005). For a comprehensive overview of the state of philosophical discussions of computer simulations see Durán (2018).

models or experiments, epidemiological simulations should also be understood as engineered scientific instruments. In general, scientific instruments are expected to meet "fail-safe" engineering development standards that address engineering issues of the kind just mentioned.

As Roddis (1993) notes, in engineering ethics, the standards governing instruments and practices are determined, at least in part, by the harms that can result from failures. We contend that high risk-management standards are required for software engineering in epidemiological simulators given the high costs of failure involved in the deployment of these instruments in public policy decision making.

We argue that where great harms can result, scientists, funding agencies, and governments ought to adopt standards of software engineering that are at least as high as the standards that societies routinely demand in, for example, critical infrastructure, aviation, or military contexts. Because of the nature and extent of the harm to societies that errors in these simulations can cause, epidemiological modelers are subject to a special obligation to adhere to high standards in the development of their software.

One objection to insisting on such standards is the risk that convergence to a single set of standards might inhibit or slow the development of scientific inquiry. We believe that this risk is not significant in the long term and that open and transparent scientific software built to high risk-management standards is likely to help rather than hinder the scientific enterprise.

Standards like ISO 2017, by virtue of the empowerment clauses they contain, are highly tailorable to specific risk regimes. For example, these standards would certainly permit some simulators, for example, those used solely to assist inquiry, to be developed in a way that does not have to meet "fail-safe" standards. ISO 2017 requires that other simulators, such as those used to verify the safety of nuclear reactors be built to "ultra-paranoid" safety standards. The key point here is that engineering standards are consciously shaped by the judgment of risk involved in the development of the system in question. Epidemiological simulators involved in public policy decision making obviously involve extremely high risks of harm.

5 Software engineering standards in pandemic policy-making

Since the late 1960s, the software engineering community has sought to codify consensus software development practices and procedures that have been (empirically) determined to help minimize development cost, risk (both developmental and operational), and to help ensure that the products of such projects reflect user needs and values (Boehm 1973; Myers 1976; Boehm, Abts, Brown, Chulani, Clark, Horowitz et al. 2000), where values include the normative interests of all stakeholders. These codification efforts have produced a series of software engineering standards.[17] One of the most recent and widely used software engineering standards is ISO 2017.

[17] Such a standard is not a contract; in the absence of a contract, compliance with a standard is therefore voluntary. A contract, however, can make compliance with a standard mandatory.

As noted in Sect. 4, the standards allow extensive tailoring or as a function of cost, schedule, risk to property and life, and other harms of comparable consequence. ISO 2017 permits software whose failure would result in inconsequential loss of property, life, or revenue, or harms of comparable consequence, for example software developed solely for personal use, can be developed with little formality In contrast, ISO 2017 requires that software whose failure could result in large loss of property or life (e.g., aircraft or automobile control), or other harms of comparable consequence, be developed with extensive formality.[18]

Although there is some variation among these standards, they characterize software projects in terms of lifecycle phases, each with formal review and documentation requirements, both which directly contribute, as needed, to the transparency, explainability, and reproducibility of the software (for the relevant community of stakeholders) developed under those standards. These phases are:

1. Specification
2. Logical design
3. Physical design
4. Implementation
5. Test
6. Maintenance

The economic and risk-management rationale for a phase-structured approach to software development and management is based on two major premises (Boehm 1981, 38):

I. In order to create a "successful" software product, we must, in effect, execute all of the phases at some stage anyway.
II. Any different ordering of the phases will produce a less successful software product.

Rationale (I) follows directly from questions that inevitably arise in the development of any software system: "What is the software supposed to do?" (Specification phase), "How do we ensure that everyone who helps to develop part the software understands how his/her part of the software correctly integrates with the rest of the software?", especially if not all personnel know all aspects of the system (logical,

[18] For further information about software standards for high-consequence applications, see Boehm, Abts, Brown, Chulani, Clark, Horowitz et al. (2000), Hatton (1995), ISO 2017, Koopman (2014), MISRA (2004, 2008), NASA (2004), Rierson (2013), RTCA (2012), FDA (2002).

and physical, design phases),[19] and "How do we determine that the software is doing what is supposed to do" (Test phase).

Rationale (II) derives directly from empirical studies of the costs of fixing an error in a software system as a function of the phase in which the error is detected and corrected. These studies show that in a large (> ~ 50,000 source lines of code (SLOC; Boehm, Abts, Brown, Chulani, Clark, Horowitz et al. 2000, 395)) or highly technical software project, a typical error is 100 times more expensive to correct in the maintenance phase than in the specification phase; in small projects (< ~ 10,000 SLOC), a typical error is 20 times more expensive to correct in the maintenance phase than in the specification phase (Boehm 1976, 1981, 40).

Each of phases 1–6 imposes requirements on, or equivalently, allocates requirements to the processes and products of one or more successor phases. Taken end-to-end, the resulting requirements-allocation induces a hypergraph (Berge 1973) spanning the elements (documentation, processes, and code) in the system.

Documentation is crucial to ensuring the transparency, explainability, and reproducibility of software. Even though this point seems self-evident, it is sometimes incorrectly argued that the code in a software system determines what that software is intended to do. So, the argument goes, we do not need documentation: code is "self-documenting". Why is this view incorrect? Very simply, the syntax and semantics of programming languages are far from sufficient to determine the intended application semantics (what the code is intended to do) of a given body of software. Any program, regardless of what the code seems to be about, could be used solely to show that the machine on which it runs will in some sense cycle the program, without regard to anything else that program might be intended to do. Only the combination of the specification, the logical design documentation, the physical design documentation, various test suites, and the code proper, can hope to capture the semantics of what the code is supposed to do.

There is no guarantee that using a software development process of the kind described in this section will yield an error-free product.[20] Empirical studies of software error and its causes show, however, that if such a framework is not used, with very high probability, software will contain at least 10 times as many errors software developed within such a framework (Boehm 1973, 1976, 1981; Myers 1976, 40).

While informal software development is often tolerated in academic contexts, standards must be higher in the case of epidemiological modeling that is used in public-health policy-making. Why? The epidemiological simulators used in policy-making are typically used in a way that errors in those simulators could lead to substantial loss of property or life, or to other harms of comparable consequence. To

[19] On average, five years after initial deployment of a software system, only 20% the original developers of the software remain on the project (Boehm, Abts, Brown, Chulani, Clark, Horowitz et al. 2000, 48). 10 years after initial deployment, on average, none of the original developers remain on the project. On small projects, furthermore, the loss of even a single key team member can force the project to restart or be abandoned. Detailed documentation is the only way to help mitigate these risks.

[20] See Horner and Symons (2019b) for a discussion of whether it is even possible, in all cases of interest, to determine that we have produced error-free software.

make informed public-health policy, decisions-makers must be able to assume that every practical effort has been made to minimize error in software.

The development of general software engineering standards has combined a recognition of both general principles of engineering ethics and attention to the empirical features of software engineering practice. In the next section we apply these standards to the ICL covid-19 simulator.

5.1 A case study

During the period from late-March through late-May 2020, we assessed how well the publicly accessible artifacts associated with the ICL covid-19 simulator conform to ISO 2017 when that standard is tailored to maximize reliability. Our assessment was based on informed software engineering judgment, reading those artifacts, building and executing some of the code, and applying various analysis tools (identified below) to the artifacts in that archive.

Our assessment was constrained by some important limitations. Most importantly, to our knowledge there is no publicly accessible documentation that officially identifies the baseline for the ICL covid-19 simulator project, though a cursory inspection of publicly available materials might suggest otherwise. For example, as of 12 June 2020, an ICL covid-19 project website (ICL 2020c) appears to identify the mapping between certain code archives and various team papers and reports. Our analysis revealed, however, that the code archives identified on this website contained modification date/time stamps that are later than the issuance dates of these papers and reports. We further discovered that some of the graphics that appeared in the papers and reports referenced on the website were not directly produced by any of the code in the associated code archives. (It is possible, of course, that some of these graphics were produced by applying software that is not identified in the reports/papers or on the website to the outputs of code that does appear in the archives.) In addition, according to Eglen 2020, the code in the publicly accessible ICL covid-19 simulator archive (ICL 2020b) is not identical to the version of the code that produced the tables in ICL (2020a) ("Report 9"), which was fundamental to COVID-19 policy decision-making in the UK and the US in early 2020. Eglen reports that an assessment of ICL (2020b) produces results that agree with the content of some tables in ICL 2020b for the test cases run in Eglen 2020. It is therefore not possible to infer from this website, or from the papers/reports linked at this website, the identity of the specific code used produce the results reported in those documents.

We note that there is, at present, no legal or institutional requirement for the ICL simulator project to make any software-development artifact of that project accessible to the general public. It is not surprising, therefore, that, even if they exist, many of the artifacts identified in the consensus software engineering standards are not publicly available in the ICL simulator project. In our judgement, however, it is highly likely that ICL (2020b) is closer to the actual ICL covid-19 simulator project baseline than any other publicly available archive; accordingly, we chose ICL (2020b), along with the published articles and reports identified in ICL 2020c, as the baseline for the analysis reported here.

Assuming ICL (2020b) as the baseline for our assessment, Sects. 5.1.1–5.1.6 describe, at a high level, the major features of each phase of the software engineering process described in ISO 2017 and assess how well, within the limitations described above, ICL (2020b) conforms to that standard.

5.1.1 Specification phase

The principal function of the specification phase of a software project is to generate an agreement (called the specification) among stakeholders that states what objectives a software system must achieve. Among other things, the specification is intended to reflect the results of the negotiation of stakeholder values. In the case of epidemiology simulator development projects, such tradeoffs can concern negotiations of the tradeoffs between the rights of the younger and the elderly, or tradeoffs in optimizing on the social-distancing directives/guidelines that collide directly with other activities that all but require person-to-person physical contact. (In several stakeholder communities, these tradeoffs (as of mid-2020) have yet to be resolved.) In some policy-making venues, furthermore, the general public is a stakeholder and thus can legitimately claim a right to have, in a timely way, access to all policy-related artifacts such as simulator rationale, design, and implementation (a view institutionalized, for example, in UK Government Office for Science 2010):

> 73. SACs [Scientific Advisory Committees] and their secretariats should aim to prepare papers in accessible language. Where issues require technical discussion, consideration should be given to separate, and additional, production of a 'lay summary' to ensure that all matters are accessible to all interested parties regardless of specialist knowledge. (UK Government Office for Science 2011, 18).

Examples of publicly negotiated specifications include the Internet protocol standard (Internet Engineering Steering Group 2020), the GPS signal specification (US Air Force 1995), nuclear reactor control simulation (Oak Ridge National Laboratory 2020), and pacemakers (Boston Scientific 2007).

Justifiable decision-making typically requires transparency and explainability – even insuring, in some cases, some level of lay understanding.[21] Policy makers cannot be expected to be able to evaluate models and simulations at the level of technical detail, but modelers should be transparent with respect to, for example, the degrees of uncertainty involved in their predictions. In complex decision-making problems facing policy makers, modelers must therefore represent the extent to which their predictions should be believed. Trusting experts is unavoidable and

[21] European Union law establishes a right to explanation in relation to the use of technology in important decisions affecting individual citizens. See for example https://eur-lex.europa.eu/legal-content/EN/TXT/?qid=1465452422595&uri=CELEX:32016R0679 Rectital 71 (accessed June 8 2020). French national law establishes the right to explanation in the 2016 *Loi pour une République numérique*. See also Morely, Cowls, Taddeo, and Floridi (2020). Such a right is not categorical, however. In the case of code and documentation that contains information whose disclosure could compromise national security, access to these artifacts must be restricted.

fully appropriate in certain domains, especially those with high technical content. However, as we discussed above, expertise in technical, scientific, or engineering domains (such as epidemiology), does not imply expertise with respect to societal goals and values.

Relevant value considerations and assumptions shaping the development of the simulation should be explicitly stated in the specification to the extent possible. The degree to which precautionary or other values enter into the choice of parameters, data sources, etc. should also be captured in the specification, because they can affect our understanding of the meaning of the predictions derived from the simulation.

Unfortunately, there is no publicly accessible specification for ICL (2020b). Ideally, future iterations of this and related simulators ought to be generated according to publicly negotiated specifications. At the very least, the specifications stipulated by the modelers themselves should be made available to the public.

Modelers and their funding organizations might protest that epidemiological simulation is a time-sensitive project whose urgency precludes such public deliberation. We contend, however, that the trust invested in epidemiologists by the public and their political representatives in these contexts means that they must be able to provide a well-articulated and understandable specification. A clear specification will explain the purpose and assumptions of the simulator in ways that will help ensure its trustworthiness and will permit all stakeholders to evaluate its import for their decisions.

5.1.2 Logical design phase

The objective of the logical design phase is to generate an abstract description, called a Logical Design Document, of a system that satisfies the requirements of the specification. Understanding what "satisfaction" means in the software development process is not simple and it involves considerations beyond the scope of this paper. For an explanation of the notion of satisfaction in the context of software development projects, see Symons and Horner (2019).

The abstract description that satisfies the specification assumes no particular implementation in hardware, software, or human procedures. Various languages can be used to express the logical design. In current practice, the Unified Modeling Language (see, for example, Rumbaugh, Jacobson, and Booch 1999) is often used for this purpose. No software is generated during this phase. There is no publicly accessible Logical Design document in ICL (2020b). This is not unusual for software used in scientific inquiry, but it does violate the consensus standards for software deployed in high-risk/reliability contexts.

5.1.3 Physical design phase

The objective of the physical design phase is to generate a concrete description, typically called the Physical Design Document, or Detailed Physical Design Document, of how specific machines, software, and human processes, and their interactions, will satisfy the requirements allocated to them from prior phases. The

software-specific component of the Physical Design Document is often called the Software Design Document, or SDD. (For a detailed description of an SDD, see US Department of Defense 1988). Assuming ~ 50 software statements per page of source code, this document typically contains ~ 10 pages per page of source code. No software is generated is generated during this phase. There is no publicly accessible SDD for ICL (2020b). However, a few items that would be contained in an SDD are included in the inline comments of the source code in ICL (2020b).

5.1.4 Implementation phase

This phase implements on actual machines, and in software and human procedures, an operational product that satisfies the requirements allocated to it from prior phases. The software developed during implementation phase is typically required to satisfy certain programming-language-specific standards (sometimes called "coding guidelines"). These standards prescribe programming-language-specific practices that are, and proscribe practices that are not, acceptable. (Such requirements are often stated in, and inherited by allocation from, the specification.) The primary role of these programming-language-specific standards is to minimize programming-language-specific coding errors.[22]

By manually analyzing ICL (2020b) along with the reports and papers nominally associated with that archive we determined that the source code in ICL (2020b) was intended primarily to study the effect of "interventions" (e.g., school closings, social distancing) and population-distribution details on the course of a pandemic. Based on our analysis of inline comments in the source code, and on the style of the code itself, the code in ICL (2020b) appears to have descended from a multi-thousand-statement simulator written in the C language by one developer in the early 2000s. In its current form, the code is almost entirely implemented in the C language subset of C++. For example, ICL (2020b) makes no use of C++ classes or polymorphism.[23]

By applying the static source code analyzer *Understand* (Scientific Tools 2020) to the source code in ICL (2020b) we found that the code consists of ~ 1000 declarative/definitional, and ~ 10,000 executable, statements, distributed across approximately 30 files. Half of these statements are in a single file that contains the source for the simulator's main routine.

The complexity of software serves as a rough measure of the intelligibility and the maintainability of the code (Symons and Horner 2014). All else being equal, software engineering attempts to minimize the complexity of a software system while satisfying all other requirements on that system. There are many way to measure software complexity. One of the more widely used measures of software complexity is McCabe complexity. Informally put, McCabe complexity is the number

[22] For examples of such standards, see Hatton (1995), Evans (2003), Perforce (2013), Google (2020).

[23] We made this assessment by reading the ICL (2020b) source code, and by analyzing the ICL 2020c source code with the documentation tool doxygen (van Heesch 2020) and the static source code analyzer Understand (Scientific Tools 2020).

of distinct execution paths through the code (McCabe 1976). Statistically, the frequency of errors in software is an increasing function of McCabe complexity (Basili and Perricone 1984). ~ 50% of ICL (2020b) has extremely high McCabe complexity. Most of this complexity comes from deeply nested "if, then" statements, primarily in function main, the understanding of which requires the reader or developer of the code to maintain awareness of complex chains of conditionality.

A simulator typically requires that the user enter input values for the parameters that are relevant to the model underlying the simulation. "Manual" analysis of the ICL (2020b) source code and its input files reveals that in order to generate a simulation, one must enter 40–50 distinct data-items, such as boundaries of geographic/jurisdictional regions, populations, and intervention types and dates, among others. For the most part these are data are intended to be derived from public health sources, but in some cases it is less clear how these assignments are determined. The high number of parameters in this simulator and its resulting complexity cannot be avoided at some level if the model is to assess the effects of even the intervention regimes that have already been deployed by various countries. As a result, ICL (2020b) is unavoidably more difficult to comprehend, correctly use, calibrate, and maintain than lower-fidelity epidemiological models such as SIR (for a description of SIR, see Vynnycky and White 2010; Nowak and May 2000). Arguably, only the authors of the code can reliably use it in its current form. For the stakeholders, transparency with respect to these parameters is important in order to ensure the trustworthiness of the simulator.

ICL (2020b) performs little to no sanity-checking (such as plausible-range-of-value testing) of its inputs, relying instead on users and external data suppliers to perform essentially all data curation. In actual practice, the data used as input to ICL (2020b) has proven to be of highly variable reliability.

With the exception of the high-complexity portion of the code mentioned above, the ICL simulator is, as of 15 September 2020, being modified in a way that is generally in accordance with at least some of ISO 2017. The scope of those modifications has to date been relatively limited. Based on time-stamps in ICL (2020b), the code has experienced, on average, average annual change traffic (number-of-statements-of-software-changed/total-number-software-statements in the system) of ~ 5%. This fraction is typical of software that has undergone relatively minor modifications, not of software that has been wholly re-engineered (Boehm 1981, 543; Boehm, Abts, Brown, Chulani, Clark, Horowitz et al. 2000, 28).[24]

It strongly appears, furthermore, that the ICL covid-19 simulator is in the process of being re-engineered from the C++ language to the R language. Among other things, this re-engineering has replaced several large segments of C++ statistical-methods software with what is intended to be equivalent high-level public R library functions. There is no publicly available documentation about how the project has shown, or intends to show, whether that the C++, and the nominally corresponding R, code agree (or whether they should). It is worth noting that the public

[24] This assessment was based on manually analyzing the source code in ICL (2020b) and our experience with software engineering standards and practices.

R-library curation protocol does not assign responsibility to anyone for ensuring that the functions in the library perform correctly or are suitable for any particular purpose. This fact has direct implications for the reliability of the re-engineered product (which might, but not because of any binding legal reason, be more reliable than the original).

5.1.5 Test phase

This phase determines whether the product generated in the Implementation phase (Sect. 2.5) satisfies all requirements allocated to the software. Testing is typically performed at various software-build levels. There is no publicly accessible Test Plan, Test Report, or official Regression Test for ICL (2020b). (ICL (2020b) does contain some test files, but what quality-control role those files are intended to support is not identified in ICL (2020b)). Eglen 2020 reports the results of porting, without modification, the source code in ICL (2020b) to two small supercomputing platforms. Using test files provided by the ICL covid-19 simulator team (it is not clear these are the test files included in ICL (2020b)), the ported code produced results that were the "same" as the results of some tables in ICL 2020b ("Report 9"). It should be noted, as Eglen 2020 does, that these demonstrations show nothing about the correctness of ICL (2020b).

5.1.6 Maintenance phase

This phase iterates the phases described in Sects. 5.1.1–5.1.5 after the product is deployed, as needed. Maintenance policies and procedures are documented in a Maintenance Manual.

There is no publicly accessible Maintenance Manual for ICL (2020b).

In summary, ICL (2020b) was not developed in accordance with the requirements of ISO 2017, or any comparable software-engineering-practice standard, tailored for high-reliability/safety–critical applications. As noted above there is compelling empirical evidence that failing to adhere to such a standard typically leads to at least an order of magnitude higher frequency of errors than if standards like ISO 2017 were followed (Boehm 1981, 381–386).

6 Discussion and conclusions

Evaluating, understanding, and controlling the quality of, software used in epidemiological simulations requires us to address a complex of interdependent normative and technical questions. In this paper, we have explained that epidemiological simulators are de facto integral to epidemiological policy making. Justifiable policy-making in epidemiological crises such as the current COVID-19 pandemic involves trades among diverse values. Some of these values, such as tradeoffs between the rights of children and the rights of the elderly, lie outside the scope of epidemiology proper. Some of the values, including the need to

assess the objective effects of various interventions, clearly lie within the scope of traditional epistemology. Furthermore, normative considerations play a role in determining what counts as an epidemic and what counts as an acceptable form of public health intervention.

Our analysis began by reviewing some recent work on the role of computation in philosophy of epidemiology. We then highlighted relevant research on the epistemology of computational modeling and simulation. From there, we described a consensus framework for software engineering that has developed over the past four decades in the software engineering community. The purpose of this framework is to provide a principled approach to balancing development cost and schedule against the possible harms of using software in high-risk venues. Within that framework, we evaluated the publicly accessible simulator archive of the Imperial College London (ICL) covid-19 simulator (ICL 2020b). Our assessment shows that ICL (2020b) does not satisfy the standards for safety–critical software identified above (ISO 2017).

We have explained why the norms from high-risk engineering contexts should be adopted in epidemiological contexts that have direct and significant public policy implications. In all projects of this kind, we urge teams to adopt methods that support transparency, explainability, and reproducibility within the framework of consensus safety–critical software engineering standards. We urge journals and funding agencies to require that published results include access to a baseline instance of relevant software along with all the relevant documentation in order to ensure reproducibility and transparency. More specifically, we contend that epidemiological simulators should be engineered and evaluated within the framework of safety–critical standards developed by consensus of the software engineering community (ISO 2017, tailored for safety–critical applications).

This analysis serves as the basis for our recommendations for software engineering standards for future epidemiological modeling projects. Furthermore, we recommend that these standards be mandated by funding agencies for epidemiological contexts that have direct and significant public policy implications.

Acknowledgements This paper benefitted from discussions with Jorge Soberón, Dick Frank, Dick Stutzke, Tony Pawlicki, Eric Winsberg, Ramón Alvarado, and Larry Cox.

Funding JFS was supported in part by the US National Security Agency Science of Security initiative contract #H98230-18-D-0009.

Compliance with ethical standards

Conflicts of interest The authors declare no conflicts of interests or competing interests.

Consent to publication The authors consent to publication.

References

Alvarado, R. (2020). *Computer simulations as scientific instruments*. PhD. Diss: University of Kansas.

Barberousse, A., & Vorms, M. (2014). About the warrants of computer-based empirical knowledge. *Synthese, 191,* 3595–3620.

Basili, V. R., & Perricone, B. T. (1984). Software errors and complexity: An empirical investigation. *Communications of the ACM* 27, 42-52. https://doi.org/10.1145/69605.2085. Open access. Accessed 2 June 2020.

Baxter, A. J., Scott, K. M., Ferrari, A. J., Norman, R. E., Vos, T., & Whiteford, H. A. (2014). Challenging the myth of an epidemic of common mental disorders: trends in the global prevalence of anxiety and depression between 1990 and 2010. *Depression and anxiety, 31*(6), 506–516.

Beisbart, C. (2017). Advancing knowledge through computer simulations? A Socratic exercise. In M. Resch, A. Kaminski, & P. Gehring (Eds.), *The science and art of simulation I* (pp. 153–174). Berlin: Springer.

Berge, C. (1973). *Graphes et Hypergraphes. English translation: Graphs and hypergraphs*. Amsterdam: North-Holland Publishing Company.

Boehm, B.W. (1973). Software and its impact: a quantitative assessment. *Datamation*, May 1973, 48–59.

Boehm, B. W. (1976). Software engineering. *IEEE Transactions on Computers*, December 1976, 1226–1241.

Boehm, B. W. (1981). *Software engineering economics*. Upper Saddle River NJ: Prentice-Hall.

Boehm, B. W., Abts, C., Brown, A. W., Chulani, S., Clark, B. K., Horowitz, E., et al. (2000). *Software cost estimation with COCOMO II*. Upper Saddle River NJ: Prentice-Hall.

Bonita, R., Beaglehole, R., & Kjellström, T. (2006). *Basic epidemiology*. World Health Organization.

Boschetti, F., & Symons, J. (2011). *Why models outputs should be interpreted as predictions. In international congress on modelling and simulation (MODSIM 2011)*. MSSANZ: WA.

Boschetti, F., Fulton, E., Bradbury, R., & Symons, J. (2012). What is a model, why people don't trust them and why they should? In M. R. Raupach (Ed.), *Negotiating our future: Living scenarios for Australia to 2050* (pp. 107–118). Canberra: Australian Academy of Science.

Boston scientific. (2007). *PACEMAKER System Specification.* https://sqrl.mcmaster.ca/_SQRLDocuments/PACEMAKER.pdf. Accessed 25 September 2020.

Broadbent, A. (2012). *Philosophy of Epidemiology.* Palgrave.

Burge, T. (1993). Content preservation. *The Philosophical Review, 102*(4), 457–488.

Burge, T. (1998). Computer proof, apriori knowledge, and other minds: The sixth philosophical perspectives lecture. *Noûs, 32*(S12), 1–37.

Dicker, R. C., Coronado, F., Koo, D., & Parrish, R. G. (2006). *Principles of epidemiology in public health practice; an introduction to applied epidemiology and biostatistics*. Atlanta: U.S. Department of health and human services, centers for disease control and prevention (CDC) office of workforce and career development.

Durán, J. M. (2018). *Computer simulations in science and engineering*. Concepts-Practices-Perspectives: Springer.

Eglen, S. (2020). CODECHECK report comparing ICL 2020c and some tables in ICL 2020b. https://zenodo.org/record/3865491#.XuIc-W5FyUk. Accessed 11 June 2020.

Evans, D. (2003). *Splint Manual*. V3.1.1–1. https://splint.org/manual/. Accessed 10 June 2020.

Frérot, M., Lefebvre, A., Aho, S., Callier, P., Astruc, K., & Glélé, L. S. A. (2018). What is epidemiology? Changing definitions of epidemiology 1978–2017. *PLoS ONE, 13*(12), e0208442. https://doi.org/10.1371/journal.pone.0208442

Freedman, L. (2020). Scientific advice at a time of emergency. SAGE and covid-19. *The Political Quarterly, 91,* 514–522. https://doi.org/10.1111/1467-923X.12885

Google, Inc. (2020). Google C++ style guide. https://google.github.io/styleguide/cppguide.html. Accessed 2 June 2020.

Hartmann, S., & Frigg, R. (2005). Scientific Models. In S. Sarkar & J. Pfeifer (Eds.), *The philosophy of science: An encyclopedia* (Vol. 2, pp. 740–749). New York: Routledge.

Hatton, L. (1995). *Safer C: Developing software for high-integrity and safety-critical systems*. New York: McGraw-Hill.

Horner, J., & Symons, J. (2019). Understanding error rates in software engineering: Conceptual, empirical, and experimental approaches. *Philosophy & Technology, 32*(2), 363–378.

Imperial college london (ICL). (2020a). Report 9: Impact of non-pharmaceutical interventions (NPIs) to reduce COVID-19 mortality and healthcare demand. https://www.imperial.ac.uk/mrc-global-infectious-disease-analysis/covid-19/report-9-impact-of-npis-on-covid-19/. Accessed 28 May 2020.

Imperial college london (ICL). (2020b). https://github.com/mrc-ide/covid-sim/blob/master/src/, Accessed 10 May 2020.
Imperial college london (ICL). (2020c). COVID-19 scientific resources. https://www.imperial.ac.uk/mrc-global-infectious-disease-analysis/covid-19/covid-19-scientific-resources/ . Accessed 3 June 2020.
Internet engineering steering group. (2020). Official internet protocol standards. https://www.rfc-editor.org/standards. Accessed 25 September 2020.
Ioannidis, J. P. A., Cripps, S., & Tanner, M. A. (2020). Forecasting for COVID-19 has failed. *International Journal of Forecasting*. https://doi.org/10.1016/j.ijforecast.2020.08.004
ISO/IEC/IEEE. (2017). *ISO/IEC/IEEE 12207:2017. Systems and software engineering–Software life cycle processes.* https://www.iso.org/standard/63712.html. Accessed 26 May 2020.
Kaplan, J.M., & Valles, S.A. (2019). Reflecting on what philosophy of epidemiology is and does, as the field comes into its own: Introduction to the Special Issue on Philosophy of Epidemiology. *Synthese*. 10.1007/s11229-019-02252-3
Keller, E. F. (2002). *Making sense of life: Explaining biological development with models, metaphors and machines*. Cambridge, MA: Harvard University Press.
Kermack, W. O., & McKendrick, A. G. (1927). A contribution to the mathematical theory of epidemics. *Proceedings of the Royal Society A., 115*(772), 700–721.
Koopman, P. (2014). *A case study of toyota unintended acceleration and software safety.* Briefing slides. https://users.ece.cmu.edu/~koopman/pubs/koopman14_toyota_ua_slides.pdf. Accessed 26 May 2020.
Kreps, S. & Kriner, D. (2020) Model uncertainty, political contestation, and public trust in science: Evidence from the COVID-19 pandemic. Science Advances 25 Sep 2020: eabd4563 DOI: 10.1126/sciadv.abd4563
Landler, M. & Castle, S. (2020). Behind the virus report that jarred the U.S. and the U.K. to action. *The New York Times*, March 17.
Lewis, B. (2020). A series of tubes: Imperial college's covid-19 coding is unintelligible. https://thecritic.co.uk/a-series-of-tubes/, Accessed August 19, 2020.
Leonelli, S. (2011). Packaging data for re-use: Databases in model organism biology. In P. Howlett & M. S. Morgan (Eds.), *How well do facts travel? The dissemination of reliable knowledge*. Cambridge, MA: Cambridge University Press.
Leonelli, S. (2012). Introduction: Making sense of data-driven research in the biological and biomedical sciences. *Studies in History and Philosophy of Biological and Biomedical Sciences, 43*(1), 1–3. https://doi.org/10.1016/j.shpsc.2011.10.001
Leonelli, S. (2016). *Data-centric biology*. Chicago: University of Chicago Press.
López-Rubio, E., & Ratti, E. (2019). Data science and molecular biology: Prediction and mechanistic explanation. *Synthese*. https://doi.org/10.1007/s11229-019-02271-0
Lynch, W. T., & Kline, R. (2000). Engineering practice and engineering ethics. *Science, technology, & human values, 25*(2), 195–225.
McCabe, T. (1976). A complexity measure. *IEEE Transactions on Software Engineering* (4): 308–320. doi:https://doi.org/10.1109/tse.1976.233837, Accessed 27 May 2020.
MISRA. (2004). MISRA C:2004. https://www.misra.org.uk/misra-c/Activities/MISRAC/tabid/160/Default.aspx. Accessed 26 May 2020.
MISRA. (2008). *MISRA C++ Guidelines for the use of the C++ language in critical systems.* https://www.misra.org.uk/Activities/MISRAC/tabid/171/Default.aspx. Accessed 26 May 2020.
Morely, J., Cowls, J., Taddeo, M., & Floridi, L. (2020). Ethical guidelines for COVID-19 tracing apps. *Nature, 582,* 29–31.
Morrison, M. (2009). Models, measurement and computer simulation: The changing face of experimentation. *Philosophical Studies* 143, no. 1 (2009), 33–57.
Morrison, M. (2015). *Reconstructing reality*. Oxford: Oxford University Press.
Myers, G. J. (1976). *Software reliability*. New York: John Wiley.
NASA. (2004). *NASA Software Safety Guidebook. NASA*. https://standards.nasa.gov/standard/nasa/nasa-gb-871913. Accessed 26 May 2020.
Nowak, M. A., & May, R. M. (2000). *Virus dynamics: mathematical principles of immunology and virology*. Oxford UK: Oxford University Press.
Oak Ridge National Laboratory. (2020). Virtual environment for reactor applications (VERA). https://vera.ornl.gov/resources/. Accessed 25 September 2020.
Perforce, Inc. (2013). High Integrity C++ Coding Standard. V 4.0. https://www.perforce.com/resources/qac/high-integrity-cpp-coding-standard, Accessed 2 June 2020.

Pincock, C. (2011). Modeling reality. *Synthese, 180,* 19–32.
Rierson L. (2013). *Developing safety-critical software: A practical guide for aviation software DO-178C compliance*. CRC Press.
Roddis, W. K. (1993). Structural failures and engineering ethics. *Journal of Structural Engineering, 119*(5), 1539–1555.
RTCA, Inc. . (2012). *DO-178C*. RTCA: Software Considerations in Airborne Systems and Equipment Certification.
Rumbaugh, J., Jacobson, I., & Booch, G. (1999). *The unified modeling language reference manual*. Reading, Massachusetts: Addison-Wesley.
Ruphy, S. (2015). Computer simulations: A new mode of scientific inquiry? In S. O. Hansen (Ed.), *The role of technology in science: Philosophical perspectives* (pp. 131–148). Dordrecht: Springer.
Sackett, D. L., Haynes, R. B., & Tugwell, P. (1985). *Clinical epidemiology: A basic science for clinical medicine*. Brown and Company: Little.
Scientific Tools, Inc. (2020). *Understand*. https://scitools.com/. Accessed 26 May 2020.
Sharma, D. (2020). What does it really mean to 'follow the science'? *The Pharmaceutical Journal*, PJ June 2020 online, online | DOI: https://doi.org/10.1211/PJ.2020.20208008. Accessed August 10, 2020.
Sider, D., & Ward, M. (2020) A fight over data infiltrates Trumpworld's response to coronavirus. *Politico.* https://www.politico.com/news/2020/04/10/trump-coronavirus-data-modeling-179226. Accessed September 28, 2020.
Smolinski MS, Hamburg MA, Lederberg J, eds. (2003) *Microbial Threats to Health: Emergence, Detection, and Response*. Washington (DC): National academies press (US). Appendix E, Computational modeling and simulation of epidemic infectious diseases. Available from: https://www.ncbi.nlm.nih.gov/books/NBK221490/. Accessed May 7, 2020.
Stevens, A. (2020). Governments cannot just follow the science on COVID-19. *Nature Human Behavior, 4,* 560. https://doi.org/10.1038/s41562-020-0894-x
Stevens, H. (2017). A feeling for the algorithm: Working knowledge and big data in biology. *Osiris, 32*(1), 151–174. https://doi.org/10.1086/693516
Symons, J., & Alvarado, R. (2019). Epistemic entitlements and the practice of computer simulation. *Minds and Machines*. https://doi.org/10.1007/s11023-018-9487-0
Symons, J., & Boschetti, F. (2013). How computational models predict the behavior of complex systems. *Foundations of Science, 18*(4), 809–821.
Symons, J., & Horner, J. (2014). Software intensive science. *Philosophy & Technology, 27*(3), 461–477.
Symons, J., & Horner, J. (2019). Why there is no general solution to the problem of software verification. *Foundations of Science*. https://doi.org/10.1007/s10699-019-09611-w
Tabish, S. A. (2007). Is diabetes becoming the biggest epidemic of the twenty-first century? *International Journal of Health Sciences* 1(2), V–VIII.
UK Government. (2020). https://parliamentlive.tv/Event/Index/7baf6cbe-6016-4c8e-bd79-f2919ad7d215. Accessed 15 May 2020.
UK Government Office for Science. (2010). Principles of scientific advice for government. https://www.gov.uk/government/publications/scientific-advice-to-government-principles/principles-of-scientific-advice-to-government. Accessed 27 May 2020.
UK Government Office for Science. (2011). Code of practice for scientific advisory committees. https://assets.publishing.service.gov.uk/government/uploads/system/uploads/attachment_data/file/278498/11-1382-code-of-practice-scientific-advisory-committees.pdf. Accessed 27 May 2020.
US Air Force. (1995). *Global Positioning System Standard Positioning Service Signal Specification.* 2nd Edition. https://www.gps.gov/technical/ps/1995-SPS-signal-specification.pdf. Accessed 25 September 2020.
US Department of Defense. (1988). *Data Item Description DI-MCCR-80012A: Software Design Document.* https://continuum.org/~brentb/2167a-did-sdd.html. Accessed 28 May 2020.
US FDA. (2002). *General Principles of Software Validation: Guidance for Industry and FDA Staff*. https://www.fda.gov/regulatory-information/search-fda-guidance-documents/general-principles-software-validation. Accessed 26 May 2020.
van Heesch, D. (2020). *Doxygen*. https://doxygen.nl/. Accessed 26 May 2020.
Vynnycky, E., & White, R. G. (Eds.). (2010). *An introduction to infectious disease modelling*. Oxford: Oxford University Press.

Weisberg, M. (2012). *Simulation and similarity: Using models to understand the world*. Oxford: Oxford University Press.
Winsberg, E. (2010). *Science in the age of computer simulation*. Chicago: University of Chicago Press.
Winsberg, E. (2019). Computer simulations in science. In E. N. Zalta (Ed.), *The Stanford encyclopedia of philosophy* (summer 2015 edition). https://plato.stanford.edu/archives/sum2015/entries/simulations-science/. Accessed 20 Dec 2018.