Philos. Technol. (2014) 27:491–494
DOI 10.1007/s13347-014-0172-9

REPLY

Reply to Angius and Primiero on Software Intensive Science

Jack Horner · John Symons

Received: 15 July 2014 / Accepted: 20 July 2014 / Published online: 21 August 2014

Abstract This paper provides a reply to articles by Nicola Angius and Guiseppe Primiero responding to our paper "Software Intensive Science" (Symons and Horner, Philosophy and Technology, 2014).

Keywords Software intensive science · Computational models

We are very grateful to Nicola Angius and Giuseppe Primiero for their careful critical readings of our paper "Software Intensive Science" (Symons and Horner 2014). While both are in agreement with the specific results we present, their responses indicate the need for further work on our part to articulate the implications of these results for Philosophy of Computing and General Philosophy of Science. In our reply, we endeavor to clarify some potential points of confusion and explain where and why we disagree.

The central thesis of our paper is this: the high conditionality of typical *software*, as we restrict the term in that paper, implies that, *in general* (i.e., over all cases of interest), there is no known effective method for characterizing (i.e., articulating all necessary and sufficient conditions describing) the error distribution in software. It follows that to the extent knowledge in a scientific domain *directly depends* on "software," there is no known effective method for characterizing the error distribution of that scientific domain. We accordingly claim that these considerations motivate *defining* ***a*** distinction between software-intensive science (SIS) and non-software-intensive science (NSIS) in terms of a measure of *software* conditionality which we call "path complexity." Both Angius and Primiero agree that this feature of contemporary science has important philosophical implications and they indicate some important lines of inquiry that merit further consideration.

Both Angius and Primiero raise important questions concerning the integration of our arguments concerning path complexity with traditional debates in philosophy. Primiero is highly sympathetic to the main thrust of our paper. "SIS," he writes, "is here to stay, and we better have the conceptual and formal tools to understand it." (Primiero 2014) Of course, we are in complete agreement and concur with his account

J. Horner · J. Symons (✉)
University of Kansas, Lawrence, KS, USA
e-mail: johnfsymons@gmail.com

of the scale and diversity of the questions that arise once we recognize the importance of SIS.

Primiero's response develops some of the implications of the SIS/NSIS distinction for the traditional categories of ontology and epistemology as used in Philosophy of Computing. By distinguishing between the "ontology of the computing process" and the "ontology of the computed," Primiero argues that our account of path complexity "represent[s] a crucial way of measuring uncertainty in the ontology of the computing process." (Primiero 2014) This is not precisely how we would have characterized it. On our view, calculations of the scale of path complexity for a particular piece of software often admit of a clear answer. The uncertainty that our arguments present is due in part to the lack of an effective procedure for characterizing error distribution in SIS. The ontology of the computing process is, in part, the source of this uncertainty, but in an important sense, on our characterization path complexity is completely determinate.

Primiero argues that our result changes the way that we understand the ontology of the computing process and further complicates the challenge of understanding the relationship between the ontology of the computed and the ontology of the computing process. Primiero considers these challenges in the context of our distinction between NSIS and SIS. Here, he helpfully highlights the distinction between the role of scientific laws in NSIS and conditional statements in branching paths of software code in SIS. He observes that "the former seem to have a normative value in describing reality independently of our experience, while the latter seem to be accountable for a procedural, user-dependent characterization" (Primiero 2014 MS?). Primiero's comment here is suggestive. He sharpens the insight with the claim that "Laws of nature tell us what can be known of the world, given some general conditions; code instead seems to tell us what the user, in her local conditions, can do with the world" (Primiero 2014 MS?).

We concur with the spirit of his assertion insofar as it acknowledges the sharp difference between traditional scientific practice and contemporary software intensive science. However, we would be more cautious than he is in the characterization of the normative role of scientific law in traditional science and in the relevance of user-dependency in software intensive science. We regard our discussion of path-complexity as presenting an objective (non user-dependent) constraint on all inquiry that involves high levels of conditionality. We are therefore hesitant to identify the shift from NSIS to SIS as a transition from objectivity to user-dependency. We regard the norms governing NSIS and SIS as relatively continuous. Normative notions like objectivity, truth, simplicity, and the like are still important for users of SIS. What we argue is that SIS poses important limits on our ability to live up to our ideals. Perhaps our difference with Primiero is simply a matter of emphasis.

Having said this, we concur with Primiero's insight concerning the relationship between epistemic confidence and software reliability. He describes our arguments as providing "a new way of approaching well-known problems of expertise from social epistemology." Insofar as contemporary experts are heavily dependent on SIS, then we concur that any traditional debates on expertise will require careful attention to the problems of error and path complexity in software.

Nicola Angius's comments (Angius 2014) on our paper help to clarify at least two general issues, which we consider in turn in the following. Both these issues depend in fundamental ways on the distinction between "software", as we have appropriated that

term in our paper, versus "models" associated with that software. The issues also variously concern how general any known software-error-reduction approaches could be.

1. What is meant by "directly depends" in the expression "a scientific domain that directly depends on software"? Angius argues that there are at least *some* kinds of *models* associated with software in scientific domains that have properties highly similar to properties of models associated with scientific domains that are not software-intensive. It follows that the specific appropriation of the terms "NSIS" and "SIS" in our paper in terms of path complexity does not characterize all possible relationships between software-intensive, and non-software-intensive, science in the non-appropriated sense of those terms.

   We agree with the conclusion of this argument, but note that:

   i. The conclusion concerns certain kinds of *models of* software, not "software" as we reserve that term.
   ii. Much depends on what is meant by "model" in the context of Angius's claim here. Some nontrivial "models" of software do not have the character he identifies. In particular, any "model" (in the sense of Chang and Keisler 1990) of a computing language that contains an isomorphic image of a Turing-complete language will have, by that isomorphism, the same error-characterization limitations we identify in our paper.
   iii. As Angius implicitly suggests, as we relax the requirements on what constitutes a "model" of software, we back into a rat's nest of problems. What, for example, do we mean when we say that a software system is a "model" of a specification? We can of course imagine specification languages that are isomorphic to a Turing-complete computing language used to implement to that specification, in which case we fall prey to (ii). But worse is true. A typical software system "specification" is written in a natural language that has a context-sensitive grammar (Chomsky 1956). A language that has a context-sensitive grammar cannot be mapped isomorphically to a Turing-complete language, thus (in the general case) failing a fundamental condition of computability (Boolos et al. 2002). Similar problems occur in the mappings among software design, abstract machine models, and test entities.
   iv. We assert only that our appropriation of the term "SIS" ("NSIS") delimits *a* distinction between science that intensively, and science that does not intensively, depend on software. There are, of course, some similarities between these two kinds of science (e.g., both require the use of experimental methods).

2. Angius 2014 claims, nominally in contrast to our view, that there *are* effective methods for characterizing the error distribution in software. He provides at least two arguments for this claim:

   v. At least static analyzers are effective methods
   vi. At least some kinds of testing by model-checking are effective methods.

Concerning (iv), we agree that at least some static analyzers, as exemplified in current practice, implement effective methods. Static analyzers are powerful tools for

detecting certain classes of software errors and, as Angius rightly observes, are unfortunately used too little in practice.

We note, however, that the scope of static analyzers cannot in general characterize the error distribution of SIS "software", because in general and by definition, *static* analyzers have no cognizance of the dependence of software conditionality on values (of test variables) that can be determined only by executing the software.

Concerning (v), we agree that within their domain, at least some kinds of testing by model-checking are effective methods. However, we again note that "effective" as used in this context is a property *of models* of software, not of software in the sense we have reserved that term.

In any case, we strongly concur with Angius's (and Primiero's (2014)) general observation that we must do what we can to try to control the errors in software, and there are many development techniques—widely under-utilized in practice—that help to reduce the occurrence of some of those errors. In addition, there are promising research programs (model-checking among them) whose objective is to help to control the error distribution in software. At the same time, we must realize that in general, characterizing the distribution of errors in SIS software has the same prospects, as Neurath put it, "of building our raft at sea."

References

Angius, N. (2014). Computational idealizations in software intensive science: a comment on Symons' and Horner's paper". Ms.

Boolos, G., Burgess, J., & Jeffrey, R. (2002). *Computability and Logic* (4th ed.). Cambridge.

Chang, C. C, & Keisler, H. J. (1990). *Model Theory*. North-Holland.

Chomsky, N. (1956). *Syntactic Structures*. Mouton reprint, 1972.

Primiero, G. (2014). On the ontology of the computing process and the epistemology of the computed. Ms.

Symons, J., & Horner, J. (2014). Software intensive science. *Philosophy and Technology*. doi:10.1007/s13347-014-0163-x.