International Journal of Microsimulation (2017) 10(3) 204-208

International Microsimulation Association

The Code *and* the Model.

A response to "The Code is the Model", by Luzius Meisser

Matteo G Richiardi

Institute for New Economic Thinking at the Oxford Martin School, Oxford, UK.
matteo.richiardi@spi.ox.ac.uk

The main argument of Luzius Meisser's paper is in the title: "The code *is* the model". This is considered as a self-evident truth, that once understood allows to reap the benefits of agile software development. The paper then continues describing some good practices in programming, and advocates the adoption of new editorial policies for simulation-based studies. The paper deals with an important topic, is well written, and engaging. In particular, I agree with most of the recommendations for increasing the replicability of the results and writing simple and accessible code put forward in the paper (sections 4 and 5). However, I think the central claim of the paper suffers from some major misconceptions.

In a sense, what we define as a model is arbitrary. When a boy takes a stick and floats it in the water, the stick can be considered as the model of a boat. However, we can also imagine that the boy has a mental model of a boat —an object that floats, is long and narrow, and possibly does not capsize too often— and that the stick is the implementation of that mental model. Similarly, when considering an analytical model, we can of course say that the model is the set of equations that characterise it. As an example, consider the 3-body problem in physics, where the aim is to characterise at any instant in time the positions of three particles with different masses, given their initial position and velocity. The problem is fully described by three coupled second-order differential equations, so we could claim that these three equations *are* the model.

However, this is often a poor description of the underlying process of abstraction. There is the real world. There is some intuition —in the mind of the modeller— that some features of the real world can be abstracted from. And then there is the implementation of this intuition. In my understand-

ing, a model is made of both the assumptions and the implementation. The implementation part is in many respects the least interesting. The important bit lies in the assumptions, the process of abstraction that leads to the identification of the core elements of a problem, or the Platonic ideals of an object: "Consider that boats are long and narrow objects that float"; "Assume that the mass of three celestial bodies is concentrated in their centre, and that there are no other interfering bodies". While a narrow definition of a model as implementation alone —as Meisser suggests— is certainly possible, I do not think it leads to any deeper understanding of the motivations, the practice and the limitations of any modelling exercise. On the other hand, the layered interpretation of a model as assumptions + implementation allows to go further. For instance, the three-body problem has no analytical solution, and it must be simulated. According to Meisser, the algorithm for the simulation *is* the model, and forget about the three differential equations. But the three equation are much more "talkative" than the algorithm for simulating them. They convey more relevant information, in a more concise and clearer way.

One objection to distinguishing between assumptions and implementation is that the implementation is not assumption-free. This might be true, but it seems to me that there is a clear hierarchy of assumptions, where those relating to the implementation occupying the lowest level. When the results of a model depend on some technical details of the implementation, there is a problem. May be those details are not only of technical relevance, but discriminate between different versions of the model, that is they pertain to the upper tiers of the hierarchy, and should be discussed separately from the implementation. If the results truly depend on some insignificant detail of the implementation, may be the model is not that interesting.

Could it be that considering the code as the model is appropriate for models without an analytical representation? I have argued elsewhere (Gallegati & Richiardi, 2009; Leombruni & Richiardi, 2005) that simulation models always allow —at least in principle— an analytical representation, consisting in a set of well-defined functions that iteratively define in an unambiguous way the dynamic of the system. However, it is true that sometimes these functions are too numerous or too complicated so that not only they cannot be solved analytically, but they offer little additional insights with respect to the algorithms used to implement them. May be in such cases we can dispense with one layer: from "assumptions → analytical implementation → computational implementation" to "assumptions → computational implementation". Still, I find it valuable to think in terms of assumptions + implementation, rather than in terms of implementation alone. The appropriateness of a model is to be judged also with respect to those assumptions. While in principle it is possible to look at the code to find what these assumptions are, this is practically infeasible, and possibly very inefficient. As soon as the model becomes complicated enough, relying on a description at the lowest possible level —the code itself— is like presenting a black box with lots of wiring inside: while it is still possible for an engineer to open it and see how it works, the broader scientific audience should not be expected to do it. Once the assumptions on which the model rests are spelled out, either verbally or with the use of

mathematical language, an additional layer in model specification is introduced, with a corresponding need for validation: program validation, the validation of the code that simulates the model relative to the model itself (Stanislaw, 1986).

Are there cases where assumptions are not important in defining a model? Meisser brings the example of software, and indeed this is his main analogy. Software is not supposed to *represent* anything; software is supposed to *do* something. The distinction matters because if a model is, above all, the formal representation of an idea, it must be adherent enough to the original idea that the researcher wants to convey (model validity, in Stanislaw's parlance). This is something which is generally very hard to check in the code. On the other hand, if a model needs just to do things —transforms inputs into outputs— it does not really matter whether the way the model does the things it does is consistent with some a priori idea. The only thing that matters is that users find it useful.

However, a model is not a software object. A software is typically used many times, for different purposes. I use the scanner app on my phone to scan all sort of documents, and use R whenever I need to manipulate data. While it is true that the same model can be applied, possibly with minor modifications or changes in the interpretation of the variables, to different settings, models are typically used to make a point. Point made, job done: the model can go and rest in the attic. Moreover, the point is often of more general validity than the model, and it possibly holds also when some of the assumptions used to prove it are relaxed. For instance, David Ricardo explained his idea of comparative advantage with the aid of a simple model where England and Portugal traded wine for clothes. Ricardo used a model to make the point that trade can be mutually beneficial also when one country possesses a superior technology in all the goods involved, but his contribution lies in the idea, not in the model. The mechanism of comparative advantage is indeed quite abstract and universal, and arguably always at work whenever an exchange is concerned. Whether it can quantitatively overcome other mechanisms that possibly work in other directions, and make trade less or no beneficial at all for at least one of the subjects involved, depends on the details of the model, and is the subject of an exterminate literature that has originated from Ricardo's work.

Meisser builds his plea for agile development on the software analogy. However, my impression is that once the software analogy is debunked, little scope remains for agile development. We can think of three steps in model building: (1) model specification (in the mind of the researcher), (2) model construction (in the lab/computer of the researcher), (3) model description (in the paper). I have doubts that agile programming can help in step (1) —model specification: it is not true that researchers wander around modifying their code until something interesting comes out of it —this would definitely be a bad scientific practice. Agile programming might perhaps render step (2) —model construction— more flexible and efficient, but I do not see it as impacting step (3) —model description— in any significant way. Software objects do not need a description: they need a user guide. A software, as long as it is continuously developed, is a dynamic object; a model is a static one. In a paper, model description happens at a stage when the model is already carved. Model description is in many respects akin to the

autopsy of a dead body.

Said differently, the goal of a paper is not to facilitate further extensions and adaptations of the model, but to describe the model in a concise way, amenable to replicability. Using the code itself to describe the model might satisfy the latter requirement (replicability), but definitely does not satisfy the first. Agile programming notwithstanding, it is difficult to go beyond the point beautifully made by Miller and Page, quoted by the author:

> The actual computer code itself is a complete specification of the model, but there is a big difference between a complete specification and an accessible one.

Of course, saying that the code alone is not enough does not mean that the code is not useful. What matters is that the model is appropriately described irrespective of whether words, formulas, diagrams, pseudo-code or actual code —possibly, a combination of all these— are used to describe it. Alike the solution to energy generation comes from an appropriate energy mix, the solution to the problem of describing the model can only come in terms of an appropriate mix of techniques, which vary depending on the nature of the model itself. Simulation models definitely require pseudo-code, and possibly the code itself, to be made available to readers.

However, there is a practical issue with respect of the publication of the code. Code, as the author also notes, can be very long, and it normally does not fit a paper. The author suggests to circumvent this problem by referring to some publicly available repository, like GitHub. As an editor of a scientific journal, I welcome submission of the code, which I pass on to the referees, and even more I welcome when the code is made available on a public repository, so that both referees and readers can analyse and use it. But as an editor of a scientific journal I cannot take responsibility for material published elsewhere. The model must be described in the paper published by the journal, in a way that is possibly amenable to replication, at least qualitatively. The paper cannot contain only a very high level description of the model, as Meisser suggests, while referring to GitHub for all the details. What if GitHub is hacked and all the uploaded material lost? But I take the point, and I think that asking scientific journals to publish the code of the model —or at a minimum an executable— as an appendix to the paper, together with some code which recalls the model under suitable parameterisations and produces the published results, would be fair. The IJM is prepared to adopt such a publication policy (see the Editorial to the current issue of the journal).

To summarise, I think of models as comprised of assumptions and implementation. Agile software development might increase programming efficiency and help with the implementation —although my impression is that the benefits can be fully reaped only by professional programmers. However, the code alone is in general not a good way to describe neither the assumptions, nor the implementation.

REFERENCES

Gallegati, M., & Richiardi, M. (2009). Agent-based modelling in economics and complexity. In R. A. Meyer (Ed.), *Encyclopedia of complexity and system science.* New York: Springer.

Leombruni, R., & Richiardi, M. (2005). Why are economists sceptical about agent-based simulations? *Physica A*, *1*(355), 103-109.

Stanislaw, H. (1986). Tests of computer simulation validity. what do they measure? *Simulation and Games*, *17*, 173-191.