International Journal of Microsimulation (2017) 10(3) 202-203

International Microsimulation Association

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

Dan Tang

Improbable Worlds Limited, London, UK.
danieltang@improbable.io

Although the word "model" is used in quite a broad sense over the various scientific disciplines, we will restrict our consideration here to models that can have a meaningful implementation on a computer.

There are two related but distinct questions that are raised by this paper:

1. Should we use source code as a way of defining a model?
2. Should we use agile software development to write computer implementations of models?

The suggestion that source code should be used to define a model is a response, I believe, to an unfortunate tendency in the modelling community to obscure the details of a model in imprecise, often incomplete, natural language descriptions or to present verbose descriptions of algorithms that could better be expressed formally. In these instances we would certainly gain by being more concise, explicit and comprehensive in our definitions of models. However, while source code is an appropriate way to formally define a model when it consists of an easily understood and exact algorithm, this is not always the case. For example, Maxwell's equations can be considered to be a model of electromagnetic phenomena which can conveniently be expressed as a set of four short partial differential equations. A computer program to solve these equations, however, would be much longer, much more difficult to understand and would necessarily require many numerical approximations which should not be considered to be part of the model itself. Many probabilistic or statistical models, too, are not well expressed using any of today's popular computer languages. Complex joint probabilities can often be expressed in code by allowing calls that return a sample from a random number generator, but it is very hard to write, for example, an efficient computer program that marginalises or conditions such a joint probability. Probabilistic programming languages hope to lift this constraint by allowing us to easily express the conditioning of random variables in source code, but this is still an area of active research. For now, mathematical notation remains the most convenient method of expressing models that involve conditional probabilities. Any model must also be a model of something; that is to say we must provide a semantic interpretation of the model's state space in order to interpret the model's output and use the appropriate real-world data during calibration and validation. I would argue that this semantic interpretation should also be considered to be part of the model, and this necessarily requires the use of natural language. So, while we as modellers should make an effort to define our models as precisely and formally as possible, we should be free to use whatever form is appropriate to make the definition succinct and understandable. In general, this will require some combination of equations, code and natural language description. This mode of definition allows the possibility of incorrect and buggy computational implementations of the model, but the alternative of considering the code to be the model's definition does not make those bugs magically disappear, it just makes the definition not quite what the author intended.

Whether we should use agile software development to implement models is more of a practical than a theoretical question. It is true that the general level of quality and modularity of code coming out of academia is sadly quite low, but I have my doubts that imposing agile processes on researchers will make them better software engineers. I have software engineers in my research and development team, they are comfortable with agile development so I let them use agile and it seems to keep up development velocity. However, I think I would have a harder time getting PhD students or post-doctoral research assistants to work in sprints and I suspect any such attempt would only succeed in stifling their creativity and alienating them. What I have seen to work much better is to let researchers do research and, when there is a need for a non-trivial piece of software, get software engineers to write the software. The software engineers should be producing well-defined tools that allow the researchers to do their research quickly and easily without having to write large, complex pieces of code. The researchers remain free to work in their own individual, often idiosyncratic, ways while the software engineers, for now at least, choose to use agile development; although I strongly suspect that agile won't be the last word in software development.

Tang *The Code is the Model, Sometimes. A response to "The Code is the Model", by Luzius Meisser*