

## A Personal Reflection on Agile Ten Years On

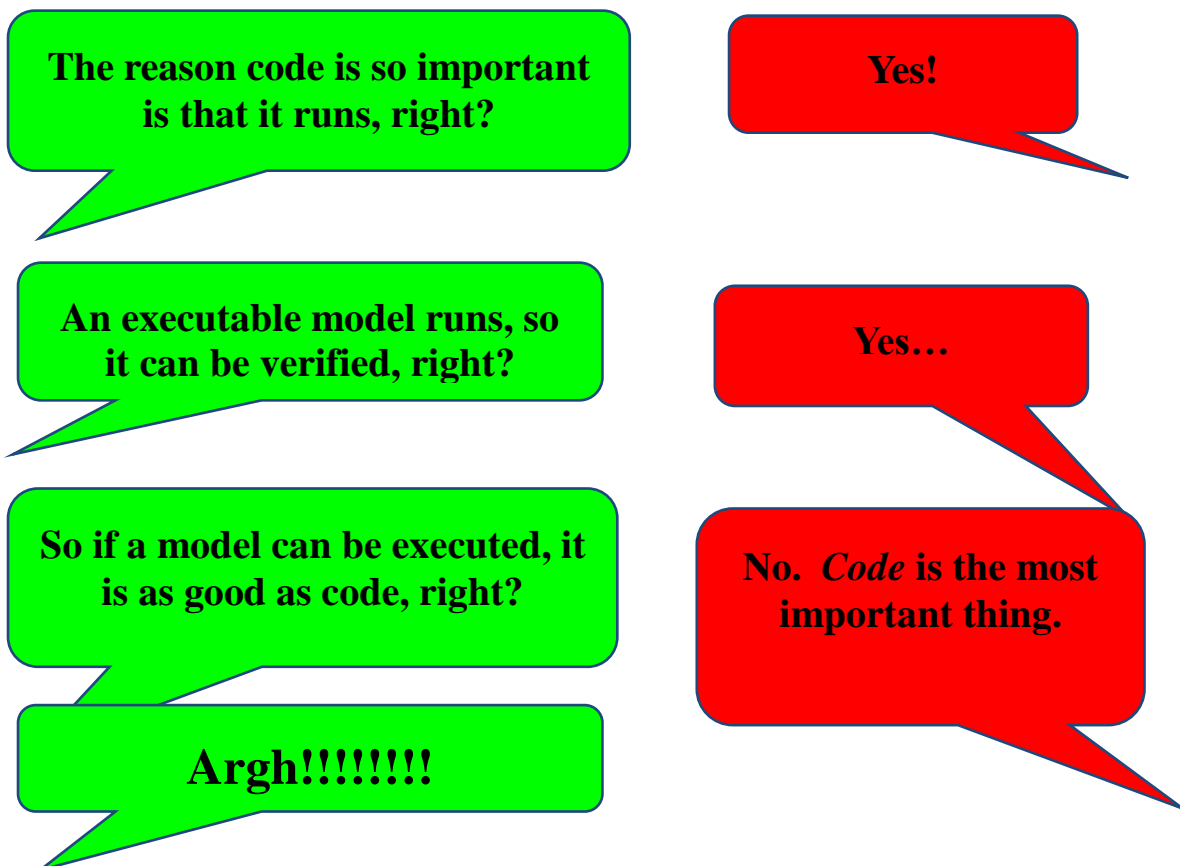
Stephen J. Mellor

I was astonished to be invited to what became the meeting that originated the Agile Manifesto because my work had always been based around building models. The *Structured Development for Real-Time Systems* trilogy with Paul Ward and the pair of *Object-Oriented Analysis* books with Sally Shlaer all focused on analysis and design, placing much less emphasis on the coding and testing favored by agile practitioners.

Moreover, I had recently read Kent Beck's *Extreme Programming* and I was horrified by the lack of emphasis on up-front thinking, abhorrence of models and the deprecation of documentation. However, it was clear that lightweight, extreme, or agile approaches were gaining traction, so I resolved to attend the meeting. The fact that it was in the Rockies in winter had nothing to do with it.

I introduced myself as a spy who hoped to derail their evil plans. While I found many of the positions taken regarding process to be sound (talking to the customer, for example, or timeboxing), the rejection of models was surely unjustified. Yes, there had been an overemphasis on "big design up front," but surely models had some value. Yes, writing documents (and models) had come to be seen as an end in itself, but surely the correct response is not to throw the productivity gains from modeling out with the dirty documentation.

So just why are models so bad? Because they don't execute, I was told. Yet my work over the preceding decade had been all about building executable models. Over the few days of the meeting I had the conversation shown in the figure below with almost all of the signatories, sometimes more than once.



This intellectual passing-in-the-night comes about because we had differing ideas about what the word “model” means. Some signatories viewed models (if they thought about them at all) as sketches, to be drawn on napkins for communication purposes then thrown away. What drew the most ire was the view of a model as a blueprint to be thrown over the wall to developers who did as they were told. I held neither of those views. I thought models ran.

Though we were executing models in 2001, it was with our own action language. I heard repeatedly at the meeting that you could not write a program to say “Hello World!” in the Unified Modeling Language (UML). And although you could, in fact, say “Hello World!” in UML, it certainly wasn’t easy. In fact, it was an advertisement for writing code. This had to be addressed if modeling was to become widely viewed as executable.

In 1997, when the UML was accepted as a standard, there were but seven actions you could carry out in a model, and one of those was “uninterpreted string.” (You can certainly forgive the signatories for thinking that models couldn’t execute!) I proposed a standard Action Language for UML. Some folk immediately suggested standardizing on Java. Or Smalltalk. Or ...

But an existing programming language is insufficient because it at too low a level of abstraction. It therefore mixes (to some degree) application concepts with their implementation. For example, when we build a particular data structure (a list of Customers, say, with pointers to lists of Accounts), we have made a decision about implementation. In the UML, we would instead express this as an association, but we would not specify its implementation.

Moreover, when we come to operate on this data in code, we have to take into consideration the implementation we have selected. For example to sum up all the account balances for a specific customer, we are required to traverse the linked list. It would be simpler to say (somehow) “sum all of the account balances for this customer.”

It’s not so much that we want to be able to execute models in UML—we could just add code to achieve that. It’s that we want the higher productivity that comes from operating at a higher level of abstraction away from the implementation. We also want to be able to change our minds about the implementation without changing the description of the problem. The UML model would have the same application meaning if we put Customers and Accounts into a database implementation.

Over the following twelve years, we (the members of the Object Management Group, the organization that standardized UML, and especially Ed Seidewitz from Model Driven Solutions) have produced a series of standards that define the kinds of actions permissible in UML; defined a subset of UML for execution; defined the semantics of that subset precisely; *and* (finally!) defined a standard Action Language for UML.

In that language, the example above would be written:

```
myCustomer.account.balance -> reduce '+'
```

where “reduce” is an operation that “reduces” the collection of balances produced by traversing from myCustomer to all their accounts by applying a binary operation, in this case ‘+’ repeatedly until only a single value remains. Of course, that can then be translated into an implementation using linked lists. Or a database. Or anything else. There are no lists in the specification, nor any traversal of them, just a very compact statement of what needs to be done. (By the way, one design requirement for the language was that it be easily accessible to Java programmers, so much of the syntax looks like Java. The example here was chosen not to obfuscate but to illustrate the power of moving away from the implementation.)

The other signatories were kind enough, back in 2001, to write the manifesto using the word “software” (which can include executable models), not “code” (which is more specific.) As such I felt able, in good conscience, to become a signatory to the Manifesto while continuing to promote executable modeling. Ten years on we have a standard action language for agile modeling.

We rarely see the words “agile” and “model” in the same sentence, but they are not at all in conflict. Rather, modelers can learn a lot from the agile folk (building tests for models early, for example) and those following an agile process can benefit from the increased productivity and eased customer communication that come from sharing executable diagrams with their pair-modeling customers. Surely that’s a win for everyone.

---

For more detailed information, including critical references, see <http://modeling-languages.com/blog/content/new-executable-uml-standards-fuml-and-alf>

---

Stephen J Mellor is an independent teacher and consultant focused on methods for the construction of real-time and embedded systems.

He is the author of Structured Development for Real-Time Systems (way back in 1985), Object Lifecycles, Executable UML, and MDA Distilled. He is also (perhaps surprisingly) a signatory to the Agile Manifesto.

Until recently, he was Chief Scientist of the Embedded Software Division at Mentor Graphics, and founder and some-time president of Project Technology, Inc., before its acquisition. He participates in multiple UML/modeling related activities at the Object Management Group, and was a member of the OMG's Architecture Board, which is the final technical gateway for all OMG standards.

Mr Mellor was the Chairman of the Advisory Board to IEEE Software for ten years and a two-time Guest Editor of the magazine, most recently for an issue on Model-Driven Development. He is also adjunct professor at the Australian National University in Canberra, ACT, Australia.

He is a regular speaker at Software Education conferences and events, and delivers MasterClasses for Software Education customers in Australia and New Zealand

For more details please email [info@softed.com](mailto:info@softed.com)

[www.softed.com](http://www.softed.com)

---

