"There's nothing so practical as a good theory"

interview with Dr. Mike Spivey

The statement above is the motto of Mike Spivey. He is known in the world of formal specifications as 'Mr. Z', after the successful specification technique Z, for which he wrote the reference manual. In November 1993, Spivey spent a few days in Eindhoven at the invitation of OOTI. He gave a lecture to the OOTI students and was the key note speaker at a symposium on formal specification methods (FSMs).[1] The editors of XOOTIC MAGAZINE *had a lunch interview with Spivey.*

Dr. Mike Spivey is affiliated to the Computing Laboratory of Oxford University. In 1982 he joined the Programming Research Group there as a graduate student, attracted by Oxford's reputation for combining high standards in theoretical research with direct application of research results to industrial practice. In 1985 he was awarded a doctorate.

Spivey has become known as the author of the standard reference manual for the Z notation, a specification technique developed at Oxford. The Programming Research group has successfully applied Z in many collaboration projects with industrial partners, two of the most important ones being with the IBM Laboratory in Hursley (UK) on enhancing their CICS transaction processing system, and with the Computer Research Laboratory of Tektronix Inc. in Portland, Oregon (USA), where Spivey spent a sabbatical year. Recently his research interests have moved from the foundations of mathematically-based methods for software specification and design to compiling high-level programming languages directly into hardware.

Introduction problems

What problems have you encountered with the usage of FSMs in general and Z in particular in commercial companies?

"The initial problem often is to persuade managers that it is worth making a tentative commitment to formal methods in order to find out how well they will work. Once that commitment is made, I am convinced that that technology will justify itself. But it is senseless just to have a tiny project to try them out, because such a project is always inconclusive. So, if you want to introduce formal methods into an organization, it is necessary to make the decision to make a serious trial. And that means a big commitment to time and money and a commitment to educate the staff to use the method. There is no other way of beginning. I think that is always the biggest problem."

Many software houses think that it just takes too much time to first write a formal specification of a system. Even if the implementation can be done faster with a good formal specification at hand, the total time of writing both specification and implementation is expected to be longer than writing an implementation only.

"Of course there are many circumstances where certain techniques do not pay off. There are many aspects of information systems that cannot be improved by formal methods. An example is: you cannot use formal methods to ensure that the screen 'looks' right. That is something that will always be to some extent experimental. You can use formal methods in user interface design, but there are some parts that are just not subject to this kind of discipline. May be they should be. Perhaps we need to know a lot more about psychology and visual methods. So there are circumstances where formal methods won't give you very much."

Specifying old code

"But what we have found is surprising. Most productive places where you use formal methods is where you have a big piece of old code that you have to work with. This is exactly the situation with IBM, but I have seen this in other places too. You have an old system, you must continue to use it, but you no longer quite understand what it does.

[1] For a report on the symposium "Specifying; a formality", see page 4 of this edition of XOOTIC Magazine.

And what IBM does is to write specifications for bits of the existing system that they are maintaining, so that they have a good understanding of what they have to keep working in the new version. So, in fact the most productive place for these formal methods has not been the 'blue-sky' projects where you start with a blank sheet of paper.''

But doesn't that imply that if you start a 'blue-sky' project with writing a formal specification that months or years later, if you have to update the code and you forgot the exact function of it, the specification is already there and you will save then?

''You can use formal methods at different levels of abstraction. IBM chose to use them at the module level. They haven't asked the question (and perhaps I should be disappointed for that) 'what is transaction processing anyway, how could we design a better transaction processing system'. But their business is not in doing that. Their business is in keeping the customers who continue to pay for CICS happy. Always in the commercial world technical issues are subsidiary to what the customer wants.''

Textronix experience

At Tektronix in the US you spent a sabbatical year. You worked there on the formal specification of an advanced oscilloscope. Tell us about that experience.

''We used formal methods to describe only some parts of the software for that oscilloscope. The oscilloscope contains about 1 Mb of object code. So, there is actually far more software in there than is visible from the surface. In fact, an important part of the software design is that you should give the electronics engineer the illusion that he is using an old-fashioned oscilloscope, unless he wants to use something digital. We used formal methods to describe parts of the user interface of this oscilloscope and also to describe the real-time operating system that is behind all the signal processing. This is a machine with half a dozen processors in it. We built a special-purpose real-time kernal based on data flow, that manages the signal processing.''

''Really, the most memorable experience for me is the meetings we used to have with the application engineers. I worked with the system programmers, implementing the operating system, and the application engineers were our users. They would propose situations where they thought our design was inadequate or they would ask us what would happen if such and such a situation would occur. And I found that, having written a formal specification, you could answer these questions directly. Even if you hadn't thought about a situation before, you could work out from the understanding the mathematics gave you what would have to happen.''

Spivey during the symposium "Specifying; a formality?" (Photo: Eindhoven University of Technology, Stafgroep Reproduktie en Fotografie)

Diagram-based FSMs

One can distinguish two mainstreams in the FSM world: those based on mathematical formulas, and those based on state diagrams and schematics. What is your standpoint in this discussion?

''Drawings are very useful. But, it is not possible

to formulate mathematical arguments that manipulate drawings."

So, you are on the side of the text-based FSMs?

"Well, I am on everybody's side. But the telling advantage for describing systems by formulas rather than by pictures that have a formal meaning is that formulas are subject to manipulation. And this is a crucial thing for the effectiveness of formal methods."

"The naive view is that what you should do is prove that the code implements the specification. This is an enormously expensive process and the more we try to do it, the more convincing it becomes that it is bound to be expensive. But that is not the only use for symbolic argument. A much more important use is to derive from the specification other statements about the system that you hope to be true. E.g., after describing a banking system, you could derive the statement that the total amount of money should be constant. You could try to prove that statement from the specification. These things give you confidence that the specification says what you hoped it would say. So the specification becomes more than simply a statement of what is required. It becomes a network of statements that are connected. Those connections, which can be made by mathematical argument, establish a sort of shared understanding among the people who have to use the specification. And that is something that mathematical formulas can do for you that pictures cannot. What people do with picture-based specifications, like StateMate, is run them on examples. It gives you a nice on-screen animation of what happens for that particular example. But that's all you can do: you can play with it."

But doesn't that help with the introduction of FSMs? A lot of people do not know anything about math and do not want to know either. If they sense that FSMs have anything to do with mathematics, they will simply not use FSMs.

"I am not saying that diagrams should not be used. I am only pointing out an advantage that mathematical formulas have that diagrams can never have. So, there is a need to use both."

"One disadvantage of these animated specifications is that it is very easy to convince somebody that the animation they see is right. However, this is based on surface aspects. What you have to do with people who are not able to read the formal specification is try and conduct a dialogue with them which is focussed on concerns that a specification should properly address. An animation of the system might be a useful vehicle to start such a discussion, but it is not a vehicle that carries you all the way to the goal."

Does Z has an animation side?

"No. The choice was that we would allow the full generality of mathematics in specifications. Good Z specifications rely on that. What will happen in a certain situation might be described in an invariant which is always true. And these invariants cannot be automatically realized by an animation. Of course you can write a prototype of the specification, but we still regard that to be an implementation of the specification and not the specification itself. That is quite important. If you try to write your Z specification so that it could be automatically realized, you immediately shut the door to all sorts of ways of specifying that are very useful and attractive. People work on animating Z specifications, but I do not have a great deal of liking for that work, because the specifications that can be animated are very restricted. And as soon as you write a specification with the knowledge that you want to animate it, you put yourself in a restricted world where a lot of things that I think are important are no longer available to you."

Executable specifications

How do you feel about executable *specification languages?*

"There is room for a wide variety of styles of specification. There is sometimes a problem where actually the best sort of specification would be a program written in a very-high programming language. I am thinking about, for example, symbol manipulation, which is algorithmic in character. It would be difficult to give a specification for some problems that is not executable. But there are classes of problems where specifications are vastly simplified by using a specification language that is not executable. So, there is room for both."

Is an executable specification *not a contradiction in terms? Somewhere in an executable specification you have to take decisions that are not strictly part of the specification, but have to be taken for implementation reasons.*

"Sure, that is the really big danger. It is offset to some extent if you were to use a specification language that is subject to mathematical reasoning, because then it is not necessary to put the implementation considerations into the specification.

There is a very attractive style of program development that some of my colleagues in Oxford are working on, and many people in The Netherlands are working on this too, and that is using transformations on functional programs as a way of moving from a specification to an efficient design. For problems that can easily be described as functional programs, it is a very attractive thing, because you can run the specification''.

''Let's first say what the big attraction to executable specifications is. You have specifications that you can run, and for which the transformation between specification and program can be done with very simple reasoning. In particular the functional programming community uses equations as their way of expressing the transformations. So you have a very easy logical system to work with and it is easy to automate those transformations.
What is unattractive is what you just mentioned. If you want to use a specification as a prototype, it's efficiency does become important. If you want to show a customer the specification running to give some idea of what the system will be like, one of the things that will hit him like a brick wall is if it takes two minutes to do the simplest operation. The difference between two minutes and two tenths of a second is not very many orders of magnitude; that is the sort of efficiency improvement one could get by chosing the right data structures. So, chosing the wrong data structures in order to get a working prototype is going to make the prototype useless in that way.''

''Specifying things by giving global invariants: very attractive when that is a central facet of a system you are describing. And that is something you have to give up in order to make it executable, unless you restrict yourself to global invariants that can be mechanically solved. And that is a very great restriction. The reason why making things executable makes global invariants unavailable to you is that if you describe a change to one part of a state, what the implementation would then have to do is solve the invariant to find out what must happen to the other part of the state. That is the general problem of solving mathematical equations and there are no general algorithms for doing that. Only in a restricted domain is it possible and we do not want to restrict the mathematical domain.''

''What I am saying is that all these disciplines of specifying are attractive. What one needs as a software engineer is a familiarity with all of them, so that you can choose the appropriate one to whatever type of problem presents itself. Sorry, if that means that software engineers need more education. But education often turns out the cheapest solution in the end.''

Can a real specification be executable? Or, in other words, isn't every executable specification in fact an implementation, may be in a very-high-level programming language?
''I don't think there is a contradiction here. I think that a high-level programming language, and even a not-very-high-level programming language, can be used as a specification language. You just have to realize what the strengths and weaknesses are. I have recently done work in which I wrote a specification in Pascal with abstract data types. It worked very well, because this development made a very nice Pascal program. The structure of the Pascal program is elaborated if you reveal more and more detail about the data implementation. So, for that development it was a good choice, but for something more abstract that starts with something that is not naturally a program, that would be a bad choice. The short answer to your question is this: You say that an executable specification language is nothing but a very-high programming language. I say that a programming language, any programming language, is nothing but a specification language that happens to be executable.''

Future

Do you think there will be once a specification language as widely used as now some programming languages, say C? Or do you think there will be many domain-specific specification languages?
''The seventies were a time when people believed that there would be a perfect programming language. The Ada project for example was in some respects motivated by that dream. It is a dream that has turned out not to come true. Since the eighties we now know that there is room for a wide variety of programming languages. Some of them we might think as computer scientists have no right to exist, but they do.
Likewise I think that an emphasis on specification *languages* rather than on specifications themselves is a distraction. That is really the point of view with which we started the Z effort. We started this against the background where a lot of people were advocating the use of first-order algebraic equations as a specification language and people had competing specification languages. What we tried

to do in Z was ignore all of that. We simply wanted to use the ordinary structures of discrete mathematics as our specification vehicle. The history is that we developed linguistic constructs to make that as easy as possible. And those linguistic constructs have in a sense become a specification language. But that has never been the main focus of the research in Z. And in fact, I would say that I am much more interested in specification than I am in specification languages.''

Where will be in ten years from now?
''I think the most pressing item is general programming, the sort of area we want to address with Z. Not urgent now is further development of specification languages. What is also not pressing is the development of tools that do mechanized reasoning, because there is no indication that I have seen that such tools really help in industrial specification problems. What I think is pressing is the developments in the use of data refinement techniques, to understand the archetypical designs that are used in programming. Of course data refinement techniques are well known by programmers, but they are nowhere well defined. I think that what we are looking for now is a way of making it practical to define the implementation of data structures in a mathematical way. Not by doing lots of lengthy formal reasoning, but by being able to document design decisions. Lots of the technology is already known, but there is a gap between what is known in theory and what turns out in practice. And that's the gap that I think should and will be closed over the next decade. There is more and more emphasis, in every engineering discipline, on the use of documentation that allows an outside observer to understand the whole design process. For example for safety reasons, where you want your company to have a demonstration that nothing that was foreseeable has been neglected. Or perhaps so that the problem of maintaining and adapting designs is made easier, because the original design team left behind a record of what they did. Both of those are really pressing problems and I think formal methods certainly can help with those. I would like to see them begin to help in a very practical way. □

Interview : Frank van den Berk, Erik Jan Marinissen.
Text : Erik Jan Marinissen.

Towards Ph.D. in Design

Continuation of page 8

prevent the title 'Ph.D.' becoming a mere bauble.

Van Lint was disappointed to meet so much academic conservatism in his university. He pointed out that EUT is a university of *technology*, not of *pure science*. The principle idea, he added, was simple. Eindhoven University hosts an excellent institution for designers, namely the IVO. When it appears that an excellent student is working on a promising design project, then it should be possible to offer the student the opportunity to continue this project and after successful completion receive a Ph.D. The academic rules that apply to this amongst others state that someone has access to a Ph.D. when he 'as proof of skill of independent practice of science has written a thesis or constructed an experimental design'. This clearly comprises Van Lints initiative.

Someone from industry added that to him a traditional four years Ph.D. project generally is not relevant. He agreed that one of the criteria for a Ph.D. must be that the scientist or designer is independent. Currently, however, this independence often grows to the level that the scientist does not involve anyone in his research, and that the resulting thesis is of academic self-interest only. For a university of technology it should also be valued that someone can combine good research and good design. The translation from theory to application is an important aspect that is too often ignored in Ph.D. theses.

Conclusion

In tentative provocation, it is my opinion that there are quite some theses for which universities should rightly be ashamed. On the other hand, there are quite some designs on which especially a university of technology can rightly be very proud. Somehow it must be possible to award these skillful designers with a Ph.D. □

Ir. Ad Peeters completed the post-masters programme Software Technology in 1990. He is working towards his Ph.D. thesis at the department of Computing Science at Eindhoven University of Technology and part-time affiliated to Philips Research Laboratories in Eindhoven. He is secretary of XOOTIC.