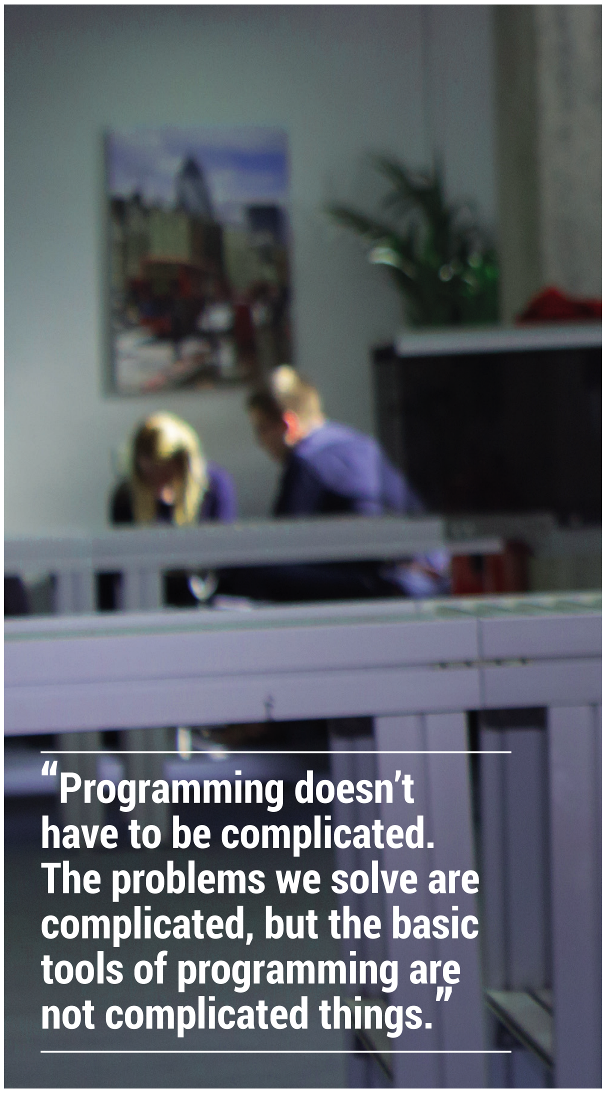
DAMIAN CONWAY

We meet the creator of a programming language based on Klingon and one of the architects of Perl 6. If only we could tell them apart…

Damian Conway is one of the Guardians of Perl (our term) and one of Perl 6's chief architects. But he's chiefly a computer scientist, a brilliant communicator and an educator. His presentations are often worth crossing continents for. He was the Adjunct Associate Professor in the Faculty of Information Technology at Melbourne's Monash University between 2001 and 2010, and has run courses on everything from Regular Expressions for Bioinformatics to Presentation Aikido (and of course, lots of Perl). Which is why, when we discovered he was making a keynote at this year's QCon conference in London in March, we braved train delays and the sardine travelling classes of the London Underground to meet him opposite Westminster Abbey.



"Programming doesn't have to be complicated. The problems we solve are complicated, but the basic tools of programming are not complicated things."

LV The main reason we wanted to talk to you is that we want to try to simplify people's experience of programming and computers. John Horton Conway said recently that his Game of Life is the blight of his life because he had gone on to do so much more interesting and important work. But what struck us by what he said about the attraction to the game is its simplicity and the fact that that goes on to teach things that you could not possibly imagine. So with that in mind, is there something like that for programming, how does that fit with Perl, and is Perl for people that think like that in the first place?

Damian: That is a huge question! There is almost an industry in making programming seem more difficult than it is. Programming doesn't have to be really complicated. The problems we solve are complicated, and at the scale we have to code things become complicated, but the basic tools of programming are not complicated things. And learning the patterns of use of those tools that work, that scale, that are robust, reliable and maintainable, isn't really that difficult. This is really not rocket science. This is not quantum mechanics. This is not that difficult.

LV But it can be. There was pride in the Perl community when you showed the Turing machine running in this much [gesticulation to show a tiny thing] code.

Damian: Sure, but that's a game. To me, that's just that I make this happen in that kind of way. It's been very interesting for me. I've recently been starting to put together classes on Perl 6, the new language in the Perl family. And the thing about Perl 6 is that it just feels like it's a lot more polished and smooth than Perl 5 ever was.

I mean, I love Perl 5 dearly, I do almost all my work in Perl 5, but Perl 6 has all of the same features but with the rough edges kind of knocked off of them. And what it gives you is the same thing that Perl 5 has always given, which is exactly the right tools to do the job you want to do and not get in your way. What I find when I change to programming in JavaScript or C++ or C is that the language itself gets in the way of my using the language.

I spend all my time coding around either limitations in the language or a particular mindset that makes you do it in one particular way, and that's equally true in Perl 5 on occasion. Perl 5 has got real deficiencies that are only just, in this very year, finally being addressed. It's insane, for example, that in Perl 5, until the release that's probably coming out in May, we haven't had parameter lists. Now this is an advanced technology that was pioneered, what, 60 years ago, and we still haven't got them. And so everyone who's writing subroutines in Perl spends most of their time simulating the behaviour necessary for a parameter list. So finally, with Perl 5.20 coming out this year, we have parameter lists.

Every language that I code in, I find these issues. A really good one is, this afternoon I'm talking about regular expressions, and I went through 20 different languages that supply regular expression mechanisms. And in about 18 of them, the regular expression mechanism is bolted on the side, so you can't write a regular expression, you have to write a string, which then gets translated into a regular expression.

And that irritation leads to mistakes too. You don't put the right number of backslashes in, 'cause it's a string, and you've got to backslash all the backslashes to get a single backslash.

But, to many of us, Perl looks like a regular expression.
Damian: [laughs] Yeah, but this is kind of the same thing. If I had just gotten up on stage this morning and just shown you Klingon sentences without explaining the structure of them, the syntax of them and how they come together, then it would just look like line noise. Alphabetic line noise, but line noise. And the thing about Perl is, in the very early design of Perl, a decision was made that there would be lots of syntactic differentiation. In most programming languages, there's only a relatively small amount of syntax. There are identifiers, there are a couple of operators and there's probably a method call mechanism, and then we do everything with that.

In Lisp it's even more extreme. In Lisp there's just comments and atoms, basically. But in Perl the decision was made very early on that we would use as much of the keyboard as possible, so that once you knew what a particular element in the Perl syntax meant, it would stand out for you immediately. So when I read Lisp, and I can read Lisp and write Lisp, and I've taught Lisp, but there's always this mental gear shift that has to go on because the language isn't helping me see what the different components are. And I find that equally true in Python, which is a lovely language and has many many benefits. But to me, in Python, everything looks like a method call, because everything is a method call. Losing that syntactic distinction makes it really really hard for me to pick up on what's going on.

Now, the problem with that is that it only works if you know the distinction in the syntax. So people coming into Perl get lost in this sea of ampersands and stars and all sorts of other symbols that we use in the language. And until you get past and it sort of goes into your hind brain and it just translates immediately, 'ah yes, that's a scalar variable', 'ah yes, that's a type blah, blah, blah', it doesn't make sense. It looks like line noise, and I fully agree.

"In most programming languages there's a relatively small amount of syntax."

So do you think it's better for people who want to learn

"Never settle for just being a Perl programmer or just being a Java programmer or just being whatever."

programming to dive into Perl straight away?

Damian: I don't think it is. To be perfectly honest, I think Perl 5 at least is a lousy first language. And the reason I think that is that learning to program isn't just about learning syntax. It's about learning at six or seven different levels at the same time. So the purely lexical level of what character do I type here, the syntactic level of what that means, the semantic level of what does the construct that this represents mean, the algorithmic level of how do I put these things together to make things work… for me it's like when I was learning to juggle or to drive a car or any other complicated multi-level activity. If you think about learning to drive a car, it's not just about how do I steer or how do I push the accelerator pedal, it's also about how aware I am on the road, how I'm aware of what the car is doing, how do I anticipate what's happening next, how do I navigate at the same time and how do I listen to the radio as well. And for me, coding is exactly like that.

For nearly a decade, I taught the introductory programming class at our university, and I was forced to teach it in C and C++ and Java and whatever it was. But the key is always the same. You have to give them a way of focusing on one level of abstraction at a time. And so the more syntax that the language that they're using has, the harder it is for them to focus on the level of what does this mean, what does it do and how do I make it do what I want. I think from that point of view there have been many CS programs over time that have taught Lisp as their first language. I think, in one sense, that's a really good thing, because I can tell you the syntax of Lisp in three minutes, and from then on it's just trying to understand how the mechanisms work and how the algorithms work.

So I don't think Perl 5 is a good language for that. I think Perl 6 is a better language because Perl 6 doesn't need as much syntax to get the basic stuff done. There's of acres of syntax in the background but you don't need it early on.

LV The UK government has decreed this year as the Year of Code. Its representative said that it was possible to learn some code in an hour. Talking to Robert Lefkowitz on the subject, he thought that programming is at a similar stage to when spaces were introduced between words in Latin script, which opened up reading to more people. And, similarly, stirrups were fundamental to the feudal system because they enabled riders to wield a sword and shield.

Damian: Or the zero in the number system.

LV Yes, exactly. So is that a relevant question for Perl, or is it better suited to Python or JavaScript, say, and should we just be teaching people concepts before we teach abstraction?

Damian: Wow!

LV Sorry, I've had too much coffee this morning.

Damian: No, these are fantastic and deep and important questions. Let's go back to the very beginning. Anyone who believes you can teach programming in an hour has no idea about what programming is. I think that I finally thought that I was a confident programmer maybe about four or five years ago, so after about a quarter of a century of coding. I felt that I was an ordinary good programmer by that stage. I don't think you can even teach HTML in an hour, to be brutally honest.

LV That's one of the very examples they gave.

Damian: No, no. So there's a fundamental misunderstanding about how complicated a task it is that we do when we do programming and how quickly one ought to be able to do that task. And I think we do a disservice if we try and throw people in at the deep end. And a lot of language choices throw people in the deep end. I would, for example, put JavaScript or Java in that same category.

If you try to teach people Java, just think about the Java 'Hello World' program, you see it online all the time. The Java 'Hello World' program has a class declaration and then it has a method declaration, it has the loading of libraries that make the thing work, it then has the method call chain to actually do that. And in order to even understand the presumably simplest of all programs, you have to understand

Java at about four different levels of abstraction. You have to understand a lot of very sophisticated concepts, including things as simple as what's the difference between static and non-static. Now, a lot of good programmers would not be able to tell you what the difference between static and non-static really is. So, a language like that, which is often touted as being a relatively simple language, actually isn't.

"Anyone who believes you can teach programming in an hour has no idea about programming."

So you can just dive in a change things?

Damian: Yeah, and that's what people do. They don't learn to program, they learn to evolve or mutate existing programs, and that's not the same skill set. And, frankly, a lot of Perl developers are like that as well. Their only exposure to Perl is in existing large-scale scripts on which their entire organisation depends. And all that they're asked to do is go in and make a small change to that. They're not asked to develop, to design, to build, to implement. It's strictly about "let's twiddle".

When you're looking for a language to actually get people up and running, you need a language that doesn't get in their way, that allows them to think about the abstractions of how to express this series of instructions clearly and unambiguously. In Perl or Perl 6, Hello World is literally "say 'Hello World'". The thing is, I can teach someone to do that in 30 seconds, not an hour, and I can go from there if I'm very very careful about what I introduce them to next. There are other languages where you don't have to be quite as careful because there just aren't that many constructs, and they have pitfalls as well.

"Ruby on Rails makes it possible for not very strong developers to build fairly sophisticated systems."

What's important is that we do need good programmers, we do need people who can do this stuff, because our entire society will utterly fall apart if we do not have people that can maintain our software. We are not a society that can survive if our software goes down. But to think that we can teach them in an hour, or a day, or a week, or a month or even a year, or the three years of the standard program, is highly optimistic.

Does that mean that, in some way, computer science has failed if we still want people to become expert scientists, when the future promised us some pseudo-code that we could just transfer our thoughts to the computer?

Damian: Yes. The future promised us a lot of things, didn't it! I'm still waiting for my flying car.

But should programming become a commodity? Eventually, for a large number of people, it will be. We will find ways whereby people can set up their environments and have the behaviour they want. But there's a fundamental mistake there in thinking things that are that complicated can be reduced down to something so simple.

Considering the context of the conversation, what do you think is the ideal path? Is there an ideal language to start with? How would you recommend people get started if you want to take them to Perl nirvana?

Damian: Perl nirvana! I can probably only go by my own path and by the paths that I've shown to my of students over time. And for me, the most important thing was diversity. Not being stuck thinking this is one way that we code. And I don't care if it's the one way of Python, or the one way of Ruby, or the one way of JavaScript, or of Java, or C , or C++ or anything. I think the important thing is that if you want to become an experienced programmer, you need to be exposed to an enormously wide range of ways of thinking about coding. You need to be exposed to functional programming systems and imperative programming systems and object-oriented programming systems and declarative programming systems and concurrent programming systems. Because it's only by opening up your mind to these different views on the same reality that you really see.

It's like back in the early days of physics where everyone either just thought of light as particles or just as waves, and there was this enormous fight over which one is it. Well, the answer is both. And is programming a purely functional activity or a purely object-oriented activity or a purely imperative activity or a declarative activity? It's all of them. And what I try to do in all of the syllabuses that I ever put together and what I try to do for myself in my own ongoing learning is find new ways of thinking about what it is that I do. How can I do functional programming in C, for example? How do I do object orientation in C? Well you can do that it. It's not easy, but you can do it. So, for me, it doesn't matter what tool I'm looking at, what I want to know is how can I think of this problem in a way that makes the solution obvious and simple and correct and robust. And often that's just I need to look at it entirely differently. And so what I would encourage every young programmer, and every old programmer as well, is never give up.

Look at the new languages that are coming out. Look at the Clojures, and the Scalas and the Darts and the Gos, and all of the different languages that are constantly coming up. See what they have to give you in the way of insights about what programming actually is. Because the only way you're going to eventually understand what this elephant looks like is if you feel the various parts of it individually and realise that they are simply parts of a greater whole.

Brilliant. Thanks Damian.

Damian: My pleasure.