I have the pleasure being here with Dave Thomas. I just want to talk a little bit about Ruby and Rails and the success and the ongoing success, on how you have been part of that and some of your opinions. What do you make of the success of Ruby especially in the last years, so it's really started to take off?
I think it's exceptional and I am really happy it's happening, not specifically for Ruby, although I think it's really cool that a nice language like Ruby has broken out and made it to the mainstream. For me, the most important the most important thing that has happened in the last couple of years is that assumptions that people had about programming languages in the big world have changed; it used to be "I have to have my Java, or I have to have my C++, or my C# and that's my focus, that's what I do and it has to be typed, statically typed" and people have seen the success of Ruby, the success of Rails and said: "Oh, it is possible to use something different." and that's encouraged people to go out and explore, so we have a whole bunch of people that are jumping on the Rails wagon right now, that's great, but I think we also have people who are jumping on the python side of things, some people looking at squeak and the work that Avi Bryant is doing, there is a whole lot of renewed interest in alternatives and that's got to be healthy.
What's the source of the change, why are people more open now then they were 5 or 10 years ago, into looking at these more niche/specialized languages?
I think a lot of it has to do with just pure pain. They have been developing now in languages and frameworks, which started out simple, started out elegant and then gradually just accreted more and more stuff around them. To the point where now I don't even know how a beginner J2EE developer gets started, because the stack of books you have to read just to write a simple J2EE application it's probably 5-6 feet tall and it's a lot of work just to get in there, and then once you are in there it's only one of these fields that's changing so quickly and new things are being added to it all the time, it's very hard to keep up and then you say: "Ok. What benefit am I getting for this?" and the reality is not a lot. It's a lot of pain for not much payback. From an organization point of view it's not a particularly productive environment. People have to do a whole lot of house keeping work to put these applications up and running, or to develop them in the first place, and they are saying: "Where is my return on this? All the stuff I'm pouring in here, why aren't I seeing all these web applications coming up?" and then they point to other sites, the startup sites, where they're producing massively successful applications on the cheap. "Why can't we do this?". "Well, because they are not using Java, or they are not using C# or whatever it might be", "Well, what technology are they using?".

I think that combination of pain plus the carrot of seeing these successful sites now means that people really don't have the choice. If you look at Rails, I'm not necessarily saying Rails is the way to go, but if you look at Rails, most of the consultants that I know that are experts in both Java and Rails will tell you that they get between maybe 5 and 15 times performance increase, let's say 5. They can write their applications maybe 5 times faster, so for domains where Rails is applicable, and that's not all domains, but in those domains where it's applicable a shop can turn out an application with 20% of the effort, if they were to use Rails over say C# or Java. If that's the case then really you could say that developers who stick with the Java model and C# model are actually being negligent, because they are costing their shops so much extra money.So in that environment what I would like to see happening, or what I am seeing happening is people who are saying: "Ok, we're no longer monoculture, we're no longer a Java shop or a Microsoft shop, but instead we are responsible for developing applications for our company and we'll do whatever it takes, we'll use the correct tools."

We're in the beginning of the Ruby and Rails adoption cycle, but I see out a lot of the folks that are starting to get into Rails are not the early adoptive type, but more towards the mainstream types. Are they as capable of getting that 5 times performance productivity increase?
Absolutely. In fact I would argue that probably they would see a larger increase in in productivity, simply because the more exploratory developer the ones that came across to say Ruby first are probably also using tricks in their Java or their C# work to metaprogram as much as they can. They are probably using code generation techniques, they are probably using more dynamic-like typing then you would normally use in Java, for example using hash maps to pass stuff around rather than creating objects to do it. So those people are probably quite a few of the tricks that you do in Ruby as well, but for a mainstream Java developer who kind of does it by the book then doing it in Rails by the book will give you a massive productivity increase, so I think that's true. I watch people that come to our studios who may be Java developers or probably haven't done much Ruby coding before and they come away having written a couple of applications and they are just stunned that they have actually got this whole thing working beginning to end and that's great. I think it is definitely possible. I think you have got to be very careful not to go and fall into the trap of saying: "What's the question, it's a Rails applications whatever the question means to b, whatever the problem domain happens to be" Rails is a particular solution for particular set of problems and part of the trick is making sure you use it appropriately.
If we were to draw an architectural diagram we might have to put Ruby at the bottom, let's say as a horizontal base technology and Rails would be one of the pillars on top of it, but, then there would be a vacuum right now I think, in terms of what other types of major applications you would build using Ruby.
I don't know if there is a vacuum in terms of applications, there is a vacuum of exemplars, in terms of people come out and say: "Here, is our magic application". Ruby is a tremendous enterprise glue and increasingly enterprise development is less about writing fresh applications as it is about creating functionality within the existing applications, plugging them together into different ways and making them do things and Ruby is a incredibly good language for that, particularly now that we have integration of JRuby, we have a decent set of libraries for RPC style work, we can interface with just about any database etc. etc. Given a all that stuff, Ruby is a really great way of creating these kind of amalgam applications and there is no real name for that, there is not particular framework for doing that, so it's not particularly marketed as a something, but I think that's a very strong place where Ruby is right now.
Is it perhaps like AJAX, where the techniques and the technology existed but no one really knew how to identify them till it was given a name?
I think it's exactly like AJAX and maybe we should go and get drunk and actually come over the new acronym before whatever we're doing with Ruby enterprise, but I really think that's a really strong analogy. People are doing that all over the place, but they don't call it anything, they don't give it a name.
Do you think it's sane to try to push Ruby into all of the equations in your enterprise. People are talking about doing enterprise Ruby stacks and maybe it's about what you were referring to a second ago, this idea of using Ruby as an integration language.
I think anybody who tries to push any technology in as the solution is missing the point.
The vendors do that all the time, right?
Yes, but one of the things that we are blessed with in the Ruby world is we don't really have vendors right now, we have a few people coming along, but we don't really have that kind of "my mortgage depends on Ruby therefore I have to push it to every one of my costumers." That's a blessing because Ruby is just a tool and I don't think anyone should be ever told "You have to

use Ruby for this". I think it's just the question of developers need to make intelligent choices and they need to have experienced a whole range of tools to know which one is the best for this particular job.
Pragmatic Programmers just put out an Erlang book. So are we going to be here in a few years talking about Erlang and the success of Erlang?
I hope we're here talking about the success of some language that makes it easy to write concurrent programs because it's a desperate need that we have right now from all sorts of reasons. First of all if you look at it form a Java perspective, every Java program is by nature multithreaded and if you actually ever expose that threading out to the developer level, the chances are pretty good that it's being done wrongly. So there is a whole bunch of programs that are working there by coincidence, using a threading model and that's not a poke at Java, it's the same in C#, it's the same in Ruby, Ruby has a thread model and no one knows that use it either.
So what makes you think that they would get it right in Erlang?
Because you don't have to worry shared state. Fundamentally, shared state is what makes threads hard. In threads you have memory that is shared between the different threads and you just synchronize access to it so you have to know where you have to put the synchronize key words and when to use a queue, it's really, really hard to do and what's worse it's almost impossible to debug. Languages like Erlang and I am not necessarily saying Erlang is a solution, but I think it points the way to the solution. Languages like Erlang have a different model of concurrency, in Erlang there is no shared memory, instead everything is done through a message passing. They also have a different model of error handling, in that they tend to say: "Don't do it!" Individual processes function focus on their particular function and then you have totally separate sorts of processes who are responsible for monitoring the world and working out if things have gone wrong and fix it up and getting it working again. so error handling in Erlang is a lot easier too, and as you know, writing multithreaded code with error handling and getting it right is very, very tough. So Erlang is a really interesting language for solving that problem and then on top of that we have another problem and that is Moore's Law is kind of breaking down, this idea that you've always got more performance just over the horizon.

It's true, but now you no longer get it in the same architecture as you used to get it. In the old days I used to get a faster processor, and more memory, and then I'd be happy happy, now I get a 2 core processor or a 4 core processor, or a 32 core processor and suddenly, yes, I've got more aggregate CPU cycles then I used to have, I can't use them, unless I can run things in parallel. So then I'm back to this problem. How do I do that? Do I have to use threading etc? And Erlang gives you a very elegant way of handling that, so it's a lot easier to write multi-core applications in Erlang than it would be in Java or Ruby or C# So long term I think if we're going to continue to rely on this ever growing increase in processor power as an industry we're going to have to switch across to environments that take advantage of that.
Right now Rails is the dominant web framework certainly in Ruby and in some places it's almost becoming the dominant web framework period? So given the benefits you get out of being a dominant player like that what do you see happening in the next few years in terms of competition for Rails?
I think what we are going to see or at least I hope what we're going to see is people realizing yet again there is no one single best solution and that maybe we can look to have smaller and more focused frameworks or solutions for individual styles of problem, and that Rails itself just won't keep growing to be the monolithic thing but instead will spawn a kind of philosophy of frameworks, where we're going to have lots of small, maybe micro frameworks for particular areas, particular problems, not necessarily always in Ruby, so maybe Java, maybe Erlang maybe

Lisp maybe whatever it is. Again, the important thing is focusing on getting stuff out that works for the custumer and whatever tool you use it should just be the best tool for the job. In terms of Rails itself, clearly there is a big push over the next six months or so to consolidate the support for REST and this idea that we should be writing our web applications using a kind of RESTful paradigm.

That's going to be very interesting to see how it plays out. They did an initial prototype, it's not really a prototype it's an implement that's in there based on the Atom protocol and the way that Atom can handle resources remotely using the HTTP verbs. That's has been pretty successful for a category of applications, particularly resource based applications It's been harder to map on to more general applications, but they are looking for ways of doing that so that is going to be very interesting to track, to see how Rails 2 handles a more general category of application while still keeping the simplicity and the benefits of the RESTful approach. We are also starting to see the stronger decoupling of presentation stuff, so that I can write, for example, a single application that very easily responds both to human being on a browser or another application sending an XML request.

And that's kind of exciting too because it means that writing big monolithic applications, I can more easily roll out a lot of functionality as a group of small applications that federate and cooperate to produce a result. Even down to the point where now simple captures where I put out like a string of graphics or characters and I say: "Type this in so I know that you're not a robot." People are federating down to a level where you have a capture server that purely serves the images for a caputre and then validates the response. That's glorious, it's a wonderful of decomposing your applications and so I hope that part of the things we see in the future with Rails is a better understanding of how to create those kinds of architectures where lots of small applications cooperate to produce lots of interesting results.
So we talked about the future of Rails. What about the future of Ruby? The 1.8 line of Ruby is kind of approaching its end and at the same time 1.9 introduces enough changes that some people might not necessarily be expecting to change over to it any time soon. How does that transition play out?
I think that one of the interesting things about the way Matz is handling Ruby 2 is how slowly he's taking it and that's not particularly deliberate in terms of "I want to be slow", it's more that he is trying things. So he'll put a feature in, he'll experiment it for a couple of months saying: "Maybe that didn't work out the way I wanted it to", pull it back out, and try something different. So he is taking it very slow deliberately approach to that and the result of that is that the community knows what's going on because all is this happening in public so it's very easy to see now where the incompatibilities are between 1.8 and 1.9, for example.

There aren't that many. The biggest change that we'll see in Ruby 2, I think, is the scoping of block parameters; it's probably the biggest change in terms of incompatibility. Other than that I wouldn't expect a large effort to move from Ruby 1.8 to Ruby 2, and you've always got the 1.9 branch to experiment with along the way, so I don't think that is going to be a major problem.

There are people that complain it's too slow, but I think Matz is taking a very realistic view that he has a lot of people now that depend on this, he is not just going to throw new features in on a whim, but he is talking it very deliberate approach to it, so I salute him for that. I don't think it's going to be a dramatic transition at all. The other thing I am seeing in terms of Ruby is suddenly

we are seeing a whole bunch of alternative VM implementations. We started out with the VM written by Matz and for 13 years or something like that, that was the VM.

The problem is that Ruby is a really hard language to parse, but as parser generator technology has moved on, people are finding it easier to write something that will parse Ruby and so now we have YARV, which has actually become the VM now for Ruby 2, we have JRuby, Microsoft has got IronRuby and I know of at least 2 other Ruby implementation projects in the works where people are producing VMs either for specific areas like the JVM or to solve specific problems. So I think we are going to see some really interesting stuff happening on the VM front over the next couple of years. JRuby, or at least the ability to run Ruby on the JVM, is really exciting.

I went to the Euro RailsConf. back in September, and saw Charles Nutter bring up my deploy application on JRuby and that was really cool and then he integrated it was some entity beans written in Java and it was totally seamless and that was significant because now you are looking at Java shops that have millions of lines of existing Java code, their own schemas, their own business rules, their libraries, the bought in 3rd party code and suddenly it's not obsolete, suddenly they can actually have that interoperability between Ruby and their existing Java applications seamlessly and that gives them the opportunity to put a toe in the water, to experiment, try things out and if it works then great, if not they just carry on doing what they are doing and there is no technology risk there, so I think all of these alternative VMs are a great thing for the language. What I would love to see and I don't think I'll see it is more work towards a specification what Ruby actually is so we can ensure compatibility and portability between all these different implementations. But right now we have a dramatically big test suite - tens of thousands of tests and so that at least guarantees that at a functional level they all do the same thing.

If you think back to what you were thinking about Ruby 2 years ago have your expectations come true, have they been less of what you expected, or have they been more than what you expected?

Interesting. When I first wrote the first edition of the Pickaxe book back in 1999, it was written because I love the language and I never expected to see people taking it up. It was purely for me, one of those passion projects, I just wanted to get the word out and the reception to that first edition of the book was about what I was expecting because it sold, OK, it carried on for a long time just selling every month, but that was my expectation. The uptake over the last 2-3 years has blown me away. It's just so cool to see developers all over the world picking up this tool simply because it seems to me that people coding Ruby are happier and the fact that these people have that opportunity to be happier makes me happy, I am really pleased with that, so in that way it way exceeded my expectations, I'm so, so happy that it's happened.

Why does coding in Ruby make you happy?

Coding in Ruby makes me happy because it's one of the shortest paths between my brain and a computer. I can think of something and I can express it very succinctly and typically fairly elegantly in Ruby without all the kind of extraneous fluff that you need in most other languages, and that makes me happy. It also makes me happy because I like playing and I like trying new things out and I like experimenting: "Does this work?" and Ruby has sufficient depth that I can do that. I am still learning new things about Ruby every day and that is good fun, so it makes me happy because I can be productive, but also makes me happy because I learn while I am using it, it's just not static.