Rick Cattell Speaks Out on Patenting, Reinventing and Standardizing Things

Marianne Winslett and Vanessa Braganholo



Rick Cattell
http://www.cattell.net/

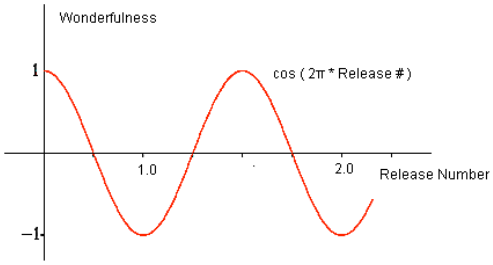
Welcome to ACM SIGMOD Record's series of interviews with distinguished members of the database community. I'm Marianne Winslett, and today we are in Tiburon, California, at the home of Rick Cattell, who is an independent consultant. Rick spent over 20 years at Sun Microsystems, where he was involved with many things that we take for granted today, such as ODBC, JDBC, and J2EE. Rick was one of Sun's first Distinguished Engineers, and his dissertation on compiler technology won the ACM Dissertation Award. So, Rick, welcome!

Thank you Marianne, it's good to be here.

What is a patent troll?

So I've been working a lot with companies that are dealing with patent trolls. A patent troll is a company who basically does no useful work, but they have patents and go after big companies with those patents. They either bought the patents or acquired them in some fashion and they threaten lawsuits. In almost all the cases I've been involved in, Fortune 500 companies will generally settle out of court because they do not want to spend a couple million dollars proving that the patent is not valid. In all of the cases I've been involved in, the patents are definitely not valid (in my opinion). The US patent office, since they've started granting patents on software in about 1990, seems to not understand software very well, so people get patents for the silliest things!

Wonderfulness is equal to the cosine of 2π times the release number [of a product].



Like what?

Like putting a cursor in a field on the screen. Also, multiple people get patents on the same thing! Right now, there are about 15 different patents on object-relational mapping from an object-oriented programming language to a database system. They use different and confusing words, so apparently the patent examiner thought they were different things. And as far as I can tell, none of these people actually invented object-relational mapping, which was invented as early as 1990 or 1989 in academia and in products, independently. It's tedious helping with this patent work, but it's also satisfying in a sense that I feel like these patent trolls are standing in the way of progress in computer science and in the way of people building real useful products -- they're just trying to make a buck. So I've been doing a lot of that work lately.

Okay so it sounds like a vampire sucking our creative blood, but who is the original filer of the patents? Are these big companies themselves? Or random evil individuals? Or startups that fail? Or what?

They are typically startups that fail. Startups almost always file patents on the things that they are doing. When they fail, their patents go on the auction block, or get acquired by another company in some fashion. Then they turn into live ammunition that's out there in a dangerous spot.

So do you think those individual startups thought they had come up with a new idea when filing these object-relational patents? Or just hadn't read the literature? Or what?

That's a good question, I often wonder about that. I've certainly talked to some people who think that they invented object-relational mapping and then I had to show them the previous work. It's a natural thing when you have a new idea. There are new ideas whose time has come and everybody sort of thinks of them at the same time. So here there was object-oriented programming, and there were relational database systems, and people wanted to connect them. Well, let's do object-relational mapping! So I would expect that a number of people think that they actually invented it, even though they didn't invent it first.

You actually told me earlier that "the system works in favor of invalid patents".

Yes.

That's awful!

Yes, here is the problem. I think this is probably not such a bad problem in other areas where the patent office has a hundred years worth of prior art and they understand it and it's well established what's new and what's not, but in software, things are very confused. So a patent troll with a patent threatens a lawsuit with a Fortune 1000 company or whatever. The company looks at the patent and on the surface it seems like they violate this patent. Proving that the patent is invalid is very difficult. Typically, a patent holder will bring a lawsuit in the Eastern District of Texas that is generally viewed as favorable to patent holders. The

problem, being faced by a big company, is they have to explain to a jury of laymen why some piece of prior art, which is a fairly complicated piece of software, invalidates this patent with a fairly complicated set of claims. That is difficult to do. They often view it as a roll of the dice -- they don't know what is going to happen when the jury makes a decision. It's a risk for them. So rather than spending 2 million dollars and go to court and fighting, they all settle. They all give $50,000 or $100,000 to the patent troll or whatever. Then the patent troll can go on to another big company with their patents.

Surely, there must be a better way to do things.

Yes, I've put a lot of thought into that. I often thought that the jury of peers should actually be people knowledgeable in computer science or whatever the patent is about, but that's not the system we have had for a couple hundred years. It would be easy to explain invalidity to someone with a bachelor's degree in computer science.

Well how do they do it in other science and engineering? So, what's the secret there?

It's the same problem, but for some reason software is more complicated. Even with a new drug, it's pretty clear whether it is or isn't the new drug. In automobiles… where people can generally understand how the parts fit together, you can explain it to them. In software, it's complicated to explain it to a layman.

So do you think we should get rid of software patents?

That's a good question. I think they serve a good purpose between companies that really have invented something and want to protect that intellectual property. I think the threshold for the obviousness of patents and the innovation in the patent has to be higher than it is right now in software, and then they would work better.

How would that happen?

That's a good question. I often ask attorneys about changing the system. There was an attempt in 2006 to improve the patent system, but it's still broken in my mind. It's a system we've had for hundreds of years… 200 years? So it's hard to change, especially because when congress considers a change, there is certainly a lot of invested interest in existing patents so there will be a lot of lobbyists in there. I don't have a good solution or I would have written a paper about it by now.

Back when people first started talking about object databases there were a lot of arguments about "my data model is better than yours". That's a really hard kind of argument to make. It is kind of religious in nature. Was that the right argument for them to be making?

I don't think so. I think the right argument is to say "people like object-oriented programming languages and they need to make their data persistent". These systems solved that problem. And then later with object-relational mapping, they actually solved that in a different way, storing the data in a relational database system. That is the strong argument for object oriented-databases and object relational mapping, respectively.

So you helped change that argument.

Yes.

What did you do?

Well, when object-oriented databases first came out, I was concerned that there was no standard like SQL for relational database systems so they would fail just because people didn't want to count on something that had a different API in every other company. So we made some progress on that. And then later at Sun, I was involved in standards in various ways. We can talk about that later if you want.

You can mention them. We might come back to them.

So I learned something from the ODMG object database standard experience that I applied to JDBC when I was at Sun trying to get a dozen different companies to agree on a standard.

Yeah?

Standards typically turn into a design by committee that is very confusing and perhaps inefficient and clumsy to use. The system that I used with JDBC was that there was a specification lead (that was me) and I took input from everybody. We often took votes on things, but there was one person responsible for the integrity of the document and the standard, so I believe it came out better as a result. The same thing got applied in other areas of the Java community. I wrote up what I learned from JDBC and passed it among the Sun management about how to make a successful, simple yet powerful standard. Those ideas eventually evolved into the Java community process, which I believe has worked pretty well over and over again in different standard arenas.

Do you think something like that could have worked for SQL?

It did work for SQL, because in that case, IBM was the specification lead for something that became the standard. And in fact if you look at almost every successful standard like Unix, it was originally done by one company or one small group of people and then was adopted as a *de-facto* standard. The standards that were designed by a committee have often failed. You can find some exceptions, but generally, the best standards were initially done by one person or one company.

What about the Web and the W3C, where they were working on standardizing things that don't exist yet?

That's hard, but it can be done. You can do standards by committee and try to do innovation, but it's difficult to do innovation in a group.

Back to the "my data model is better than your data model" argument… In a sense, that was the argument for Java and they actually won that argument.

Yeah, Java is a nice programming language and it has a pretty good model, but I claim that it didn't succeed just because it was better. I think it succeeded because it was in the right place at the right time. In fact I was very frustrated about the time Java came out because at Sun we were stuck with C and C++. After being at Xerox PARC working with languages that did automatic garbage collection and were safe, I found myself making stupid errors: forgetting to free memory, clobbering memory… errors that are very difficult to fix. Java wasn't the first language to solve that problem. There were other languages that were equally good, yet they were not popular. They didn't take off in the way that C and C++ did. So, here with Java, finally there was a language that was good and it was popular and so I could expect there to be lots of libraries and activity around it. So I got excited about it around 1995. I left my work on relational and object database systems at Sun, and focused on Java. Of course I then went back and did JDBC, so I was still in the database area. Those were exciting times -- there were only about a dozen people in the Java group when I joined and it grew very fast.

What was Java originally intended to be used for?

That's a good question. James Gosling and his group had been looking at devices like set-top boxes where they wanted to be able to send programmed material to a box in a remote location. Yet you wanted it to be secure and not crash the box if there was a faulty program. So Java took off with the Internet even though that wasn't the original intent. There were some sharp people at Sun that figured out that this was a way to convey programs in a browser and jumped on it.

Okay, so they started using it on the client side?

Yes. And my contribution to Java was thinking, "wow, this would also be a great language for servers", because there, you are also trying to build websites with some application server logic that you need to be safe and that you need to be able to easily move around. So I started this group doing what I called Enterprise Java at the time, which is now known as J2EE (or Java Enterprise Edition) to build a set of APIs, a platform for building server side Java programs which required additional functionality that wasn't necessary on the client.

People look at the wrong things when they are trying to tune [database systems]. They don't realize that the number of machine instructions that are used in a database call is now a critical factor they are trying to minimize.

How have things changed in the past decade, for database applications?

Two things. One is that the hardware has changed. RAM has gotten much cheaper, so that you can store a lot of your database in memory, and flash also gives you a way to make very fast reads and writes to databases, which was not possible with disk. The other thing is a change in the market in that there are a lot more people out there trying to deal with really big databases with lots of users because every web company is at least dreaming of having millions of users and they want to be able to scale up to that. So there's a lot of interest in scalable database systems today, which is where I've been focusing in my consulting practice.

What's the right way to respond to those two trends?

I think there's a lot of excitement around scalable database systems. There's also a lot of confusion around them. There are perhaps a dozen new database

systems just in the last 18 months to 24 months[1] that are so called NoSQL database systems or are scalable SQL database systems. So there is plenty of work for consultants like myself who are familiar with the various products and their strengths and weaknesses. Customers just need to make a careful choice of what are the requirements and which database systems really satisfy those. That's the age-old story in software, making sure it solves your problem.

Did you say a dozen new products?

Yes, I would say that.

The system works in favor of invalid patents.

That's a lot! So it is really responding to this need.

Yes, and most of these systems are open source. So there are communities of people who work around them trying to tune them. In my experience, a number of the systems are still immature. The bugs haven't been shaken out and they have some performance characteristics that are lacking when you go beyond the simple case they were originally designed for. So I think we're going to see some fallout in this industry and I've been doing a lot of benchmarking of systems with my clients to look at which ones actually scale, instead of running into all kinds of communication bottlenecks when there are more than 4 servers.

So I would love to ask you which systems perform best on your benchmarks but one of the lessons that database researchers have learned is that it's not good to talk about that. So I won't ask you about that but I will ask you about what are the types of scaling that the systems need to do and what are the good ways of achieving it.

So, in some ways that hasn't changed in the last 10 or 20 years. You can achieve horizontal scaling across multiple servers by doing replication. This gives you scalability for reads and also gives you a way to recover from crashes. And by doing partitioning of the database across multiple servers, you can split the load of writes and reads over multiple servers.

That's not new, right? I remember learning about that in grad school. So what's new? I guess nothing is new under the sun. That's what the patent story is. It's nothing really new, but there must be something new here?

Well, with the two things I mentioned… more people are interested in this. They care and there are some open source systems built around these ideas. The other thing is that RAM is cheap and flash memory is cheap and that there are new mechanisms to do faster communication in between machines that will reduce the overhead. Also, you get more cores per computer now than ever before, which allows you to do better vertical scaling on each machine.

So what is vertical scaling?

Vertical scaling is using all of those cores that you have on a modern computer effectively. In the bad old days, you weren't too far off just having a single process per machine that processed all the database calls. Now you can't afford to do that if you want to utilize all the power of the computer that you have in front of you.

So are we still in a world where we are trying to minimize the number of disk accesses?

No, that has changed for most database systems. Oracle, Informix, Ingres, all major database systems were originally designed with the goal of minimizing the number of disk accesses and now that story has changed. There are papers about that like the one "It's Time for a Complete Rewrite"[2]. The things you're trying to optimize in a relational database have changed, so it's time for a complete rewrite. Some of the NoSQL systems are doing their style of rewrite and other people with products like Clustrix and VoltDB are going in a different direction with relational databases, taking advantage of splitting the data across multiple machines, using RAM effectively, and so on.

Some of the age-old rules still apply and people forget that. People forget that if you're going to scale to a dozen or a hundred machines, you can't have any manual intervention. You can't have the database go down and have an operator come in and fix it. The system has to be self-repairing. When a server fails, it has to be replaced online automatically. If you have to change your schema, you can't bring a hundred or a thousand machines down while you upgrade your

[1] Editor's note: recall that this interview was conducted in 2011.

[2] STONEBRAKER, M., MADDEN, S., ABADI, D., HARIZOPOULOS, S., HACHEM, N., HELLAND, P. The end of an architectural era: it's time for a complete rewrite. In: VLDB, 2007. pp 1150-1160.

schema. It has to be always online. That is something that I often find: clients in a startup that are not familiar with. They say, "okay, we'll just use more servers". They forget you also need high availability and continuous online operations.

Are there any new research issues there that people haven't already looked at?

Well there are in the performance arena, I believe, because a lot of the performance is counter intuitive. Where is the time actually going in a complex system like that?

> ***It would be easy to explain invalidity [of a software patent] to someone with a bachelor's degree in computer science.***

Counter intuitive? Like what?

So people look at the wrong things when they are trying to tune. They don't realize that the number of machine instructions that are used in a database call is now a critical factor they are trying to minimize. They don't realize that the inter-machine cost can be fairly expensive. It can take a thousand machine instructions just to move one byte from computer A to computer B. So your database design has to be built for minimizing the inter-machine calls with a new sense of urgency that wasn't the case in the past.

I see. Sounds like it's time for a rewrite for the textbooks also.

Yes, I would say so.

You spent much of your career at Sun, which didn't have any database product. That's counterintuitive too, but you exerted a lot of influence from there, including on the standards like we were talked about and de-facto standards. Could you have been as influential if you were in a company that did have a database product?

Yes and no. I did have some advantage, being in a neutral position. For example, by working with various database companies at Sun, which didn't have its own database product, I was able to get a lot of cooperation on tuning for their database systems with our operating system. Also with the ODMG with the object databases companies, I was in a neutral position. Even with Java, I was in a relatively neutral position, so that helped.

In retrospect, working for Sun did limit my career in many ways because we didn't have a database system product that I could work on. In fact, when I came to Sun, I was hired by Eric Schmitt to start a database group and the first thing I decided is that we actually didn't need a database group to build a database system. At that time, IBM, HP, DEC and even Apollo had their own database system and my conclusion was that it was better for us to just work with all of the vendors and they worked harder to have the best performance on our platform and to sell on our platform instead of selling against say, IBM, who had both a database system and a server they were trying to sell.

I flipped my position on that around 2000, 15 years after I went to Sun, saying, "now maybe it makes sense for Sun to have its own in-memory database cache in front of these relational database systems and that in-memory cache can actually evolve into being an in-memory distributed database system". Unfortunately around 2001 and 2002, Sun's profits were falling and there was not a lot of money to start a whole new project. I actually wasted some time at Sun thinking each year that I was just 6 months away from getting funding for building a distributed in-memory database system but it never happened and I gave up around 2007, when then I went out on my own.

Speaking of being out on your own, you're self-employed and doing research. I've never heard of a self-employed person who is doing research. So how does that work? Are you your own funding agency?

I am, I guess. I think every software consultant actually has to do some amount of research doing a benchmark here, doing a study of the details of an implementation in order to be familiar with what they are talking about. So I might be doing a little bit more of that than others, but I think that is actually necessary in order to give advice on database systems.

Speaking of advice, do you have any words of advice for fledging or mid-career database researchers or practitioners?

Yes, that's an area near and dear to my heart because I started writing this book that's been on the back burner for a long time. It's entitled, "Things I Wish I Learned in Engineering School". When I got my PhD, I went

out into research and then went out into industry building products. I learned over and over again that people waste a lot of time, spending many months or years building something that you could have known was not going to succeed, if you had had some experience with things that companies do wrong.

So how can you tell if something is not going to succeed?

Well my book is a list of rules to follow, advice to follow to avoid the errors that people commonly fall into. For example, rule #1 in the book is that most organizations, most startups, most projects, try to do too many things. They're almost always trying to do too many things. The landscape is littered with startups that are trying to do too many things. In fact I've consulted with for at least one of them where they didn't take my advice on it. So Steve Jobs for example, whom you think is the epitome of a great leader and a successful leader, started NeXT computer, which failed. He tried to do his own hardware from the bottom up. His own operating system, a new programming language, a new window system, a new programming environment, new tools, new ideas… and he was trying to do it all at once with finite resources, competing against existing players like Apple and Microsoft on Intel, which had established application bases. So he was working uphill in more than one way. He violated multiple rules in my book. He was trying to do too many things. And he was trying to displace an established market player without enough "better" to displace them.

Yeah there aren't enough places for being the enough better to displace the established leader.

So my book has four chapters. One is about successful organizations: why organizations fail and succeed. A second is about technology, errors that people make in technology. Like error #33 where you come up with a new idea that has a 30% chance of success and base it on another new idea that has a 30% chance of success. Now you have like a 10% chance of that your system succeeds. The third chapter is about successful products. There's a rule in there that says "wonderfulness is equal to the cosine of 2π times the release number".

What??

What this means here is that when you initially come up with a new idea, it sounds wonderful and you throw in all this other new stuff that sounds wonderful and you finally get to a proposed release. At Release #1 you're at the top of the curve in terms of the wonderfulness of what you're going to do. And then the product gets released and you discover all of the problems that people come in with. There are issues and there are bugs and you realize that there are missing features that were necessary. So you go down the wave again, and then the second release begins where you say, "Oh, we can fix all of that". We're going to put in this and that and it's going to be wonderful again. We go back up the curve until Release #2 comes out. So this is something you should be aware of. If you go to a startup, everyone is really excited about this first release. When you listen to Steve Jobs talk about his product, for example, he has this reality distortion field and everyone in the room will be convinced that this is going to be the best thing since sliced bread. So you just have to be aware of these trends when you put out a product.

The final chapter is about career advice. For example, you need to spend time increasing your effectiveness. For example, one of the best things I ever did was take a typing class in 8th grade. I didn't realize at the time how useful that was going to be to me. It's useful as a programmer to spend a lot of time learning about tools and what you need to do because if you've got it all in your head you're not constantly stalled, not realizing there's a solution to the problem you have or a better tool to solve the problem that you have. There are 80 rules in the book, twenty in each of those chapters, about things that you ought to know.

Now where can I read it? Or did you just get that?

On my website, Cattell.net, there is a presentation that I have given at the University of Illinois and a couple other universities that summarizes some of the ideas. If they are interested in giving me comments on the manuscript, then they can send me an email and I'd be happy to share it with them.

Among all your past research do you have a favorite piece of work?

That's a hard question. I like the work I did back in Xerox PARC, on new kinds of user interfaces to database systems where you can see things laid out spatially or browse around the databases by clicking on things. I did a little bit of work on that at Sun initially but I was in the wrong place. I was in a hardware company trying to do an innovative software product. That would've been fun to have spent more time on.

If you magically had enough extra time to do an additional thing at work that you're not doing now, what would it be?

In database systems, I'd like to have a little bit more time to experiment with the properties of these distributed systems. I'd like to setup a lab of a hundred computers and experiment with the characteristics and see what happens when you change the way database systems work, but that's a bit too big of a project.

No, it's a fine answer because the question was if you magically had enough time (and money, I guess), right? If you could change one thing about yourself as a computer science researcher, what would it be?

I think I made a mistake in my career in not choosing to focus on either doing research or doing products and trying to do everything at the same time for my entire career. It would be good to do one and then do the other and then the one. For example, I often considered starting a company myself and doing a new kind of database system. I didn't want to make the sacrifice of the time out of my personal life to do that. It also would put my research work at a standstill. In retrospect, it probably would have been good to go out and make some mistakes starting a new company (maybe more than once). Then go back into research or whatever.

Okay, thank you very much for talking with me today.

Thank you! I've enjoyed your interviews on SIGMOD. This is a great service you're doing for our community.

Thank you!