**Highlights from Kevlin Henney’s #TechTalk**

Transcript

**Zoe:** Hello. Welcome, everyone. Welcome. My name is Zoe Cunningham. I'm a Director at Softwire, and I host the *Softwire TechTalk* podcast. Welcome to this Digital Lighthouse Tech Talk. On the Digital Lighthouse, we get inspiration from tech leaders to help us shine a light through turbulent times, so that we can learn, act, and change as a result for the benefit of our businesses. We believe that if you have a lighthouse, you can harness the power of the storm.

A warm welcome to all of our listeners. Today, we're going to be talking with technology consultant, Kevlin Henney. We will take questions, we'll host a Q&A at the end of the session. If you are not familiar with Twitter Spaces functionality, [chuckles] we'll be finding out what all the features are. There is a fantastic ability to turn on caption. If you have any hearing impairments, you can, from the settings function, turn on captions.

You can also send emojis if you feel very moved by our discussion. You have the ability to share this event on your own Twitter timeline from the right-hand side. On the left, you'll see there is a request button with a microphone. When we do the Q&A, this is the way that you can request to ask a question. We will monitor that and then we will bring people online to ask Kevlin questions.

Just to let you know, this is being recorded and will be available to listen back immediately after the event ends on @SoftwireUK Twitter for 30 days. We'll also be sending out the recording and some highlights in an email next week. If you joined us without registering on Eventbrite, we can add you to the list if you'd like to get the email. Just DM us @SoftwireUK.

Kevlin, our guest today, has a long career in software consulting. In addition, he's been the columnist for many magazines and sites, including The Register. He's the Co-author of two volumes in the *Pattern-Oriented Software Architecture* series, Editor of *97 Things Every Programmer Should Know*, and Co-editor of *97 Things Every Java Programmer Should Know*. You may also know him from his many speeches and keynote speeches at conferences. Hello, Kevlin.

**Kevlin Henney:** Yes. Right.

**Zoe:** Hooray.

**Kevlin:** Here is the discovery. [chuckles] Hello, Zoe.

**Zoe:** Hello. Yes. [chuckles]

**Kevlin:** We're going to talk about technology, aren't we? It turns out that when I moved my phone, it went from portrait to landscape mode and it dropped me out of Spaces, which is an exciting feature, nice on Twitter.

[laughter]

**Zoe:** Kevlin, were they programming for the past when they do? [chuckles]

**Kevlin:** Yes. This is one of the interesting things. There is this interesting thought I had. A few months ago, it occurred to me that, although I've been saying this for years, I've never quite encapsulated it quite so succinctly, that most of the time we are programming for the past. This is a reference to the fact that when a developer sits down and touches a piece of code, the chances are that that code was written in the past. It was written with a past assumption.

In fact, in a given eight-hour day, a developer is likely to spend seven to seven and a half hours, if not the whole eight hours, dealing with all of the issues that came from the past. That's not just simply a question of legacy code. It's legacy assumptions as well. Here, we have the assumption that, there is somebody from Twitter Spaces who developed this, had a very particular view about what mobile phones are and what their setup is. In other words, it was brittle with respect to their setup. We're stuck with that, which is an interesting consequence. More broadly, there is this question when people, actually, sit down to touch a piece of code, although we have this idea of technology is all about the future, we're building the future, most of the time people are pulled back into the past. That is legacy code, that is traditional assumptions. It's also legacy languages, legacy frameworks, it's all positioned very much in the past. I think perhaps if you are doing a startup, you are in the privileged position to possibly be doing something near the present, although it might not be the future.

**Zoe:** Closer to the present.

**Kevlin:** Yes.

**Zoe:** Is it a generational thing? Is it that older programmers also have their own they're bringing all the assumptions of how they've always coded things before or is it everyone?

**Kevlin:** I don't think there is a generational distinction, but I think it's now more amplified because software runs the world and everything that we do affects somebody else. In the past what was the scope of legacy systems was in one sense much smaller. The fact that perhaps, in fact, dealing with my interactions this morning, my frustrations, let's put it that way. My frustrations this morning with trying to book a hotel via their site and I was struck again by this classic problem.

I'm sure we've all experienced it where the website says something else, and it says one thing and then you find out, actually, there's probably another database in the background. In other words a classic legacy system, but there's multiple databases. I remember trying to change my address of my bank a couple of decades ago. That was almost impossible because it turns out that there was one database somewhere that had my information on. It was probably written in COBOL and it was the previous address.

It took about two or three years to get this one sorted. Now, that was relatively isolated in a very personal experience, but now it's much more available on the web. Actually, that's what I encountered with this particular hotel site. I tried booking yesterday for something. Tried booking, failed. Tried today, failed. Tried different browser, the usual thing and then, eventually, decided I would resort to old fashioned technology and rang them up, and they told me there was no availability.

I said, "Well, I'm looking at a page here that says there's availability, just worked out your book." "Well, we have the actual numbers." This company that I was dealing with, this hotel company had that multiple database problem and that's now far more visible elsewhere. They have the privileged access, whereas the shiny front end that everybody uses, actually, is working on scale data. This is not some eventual consistency problem. I think that one expects that 24 is enough for any.

**Zoe:** [laughs]

**Kevlin:** It should be enough to synchronize and so they have got a coherence issue. I think older programmers are just as bad and no better than younger programmers and vice versa with one exception, they've probably been bitten by it more times and are more likely to ask certain questions. These assumptions both of the users, but also the existing systems until it slows down, I think every generation is going to be slightly overwhelmed and surprised by it.

**Zoe:** I love that point, talking about slowing down, obviously, this is linked to the fact, right, there used to be fewer. [chuckles] The quantity of legacy code is just increasing so quickly as the rate of development increases. I suppose code just doesn't really get decommissioned in the same way or very rarely, we're just always building on top of things that are already there.

**Kevlin:** I think that is very true. In fact, a discussion earlier this year, possibly last year, these days who knows? The last two years all merged into a blur. At some point in the last couple of years, I, actually, went and checked when the term legacy came into being, and it started being used in this context, the context we now use it in the end of the 1980s. There was Lexico before that and people were already seeing that there were problems, but they didn't have a name for it. They didn't recognize that would, actually, be one of the fundamental issues we confront. The point that you make is that there's this notion that code doesn't get decommissioned, I think is an important one and it's not a necessary truth of code. I'm intrigued by this possibility, I'll come to that in a moment, but I remember running a workshop a few years ago and talking to this old guy who probably been there and seen it all. Very much an old-school embedded systems programmer, and we were talking about architecture. That's the workshop.

He made a throwaway comment, which I didn't understand, and he was really down and very negative on layered architecture. I thought, "Well, hang on. Layers?" One of the ways it's a complexity, otherwise it just becomes a big mush. Without layering, you don't really know where you are in a large system. It's one of the means by which we orient ourselves and say, "Oh, this layer is about this technology," or, "It's this team," or, "It's this domain."

We use these separations and I thought it's one of the fundamental ones, classic idea that goes back to the 1960s. I asked him a little bit, yes, probed a little bit further and it turns out his understanding of layering was quite different to mine, but, actually, surprisingly valid. What he called layering was where you built a new system on top of the ashes or the broken APIs of a previous system and keep on going.

**Zoe:** Like with the build thing in the bottom of the riverbed.

**Kevlin:** Yes. Just like whenever in cities like London, this happens all the time. Whenever you have somebody's putting up a new building, half the time construction stops because you come across all Roman remains or some Anglo-Saxon settlement. Quite, literally, we build our civilization on the remains of the previous one. That was his perspective on layering. It wasn't an abstraction, it was what we know considered legacy layering or rapid.

That I thought was a really interesting perspective. Like, "Yes, it's different, it's archeological layering rather than layering by abstraction." I don't think it's necessary the code should be this old or that old code should be a problem. I think that that is a curious dysfunction of the industry. It goes back to that point about we have not slowed down and consolidated. I think when everything's in a rush, you have no reason to stop.

When loads of money comes into the industry, as I said before, the software, basically, runs the world. We don't have a reason to stop, and consolidate, and take stock. I think many individuals feel that we probably should, but that's not the pace. The industries are always moving forward. That gives us, again, that idea that we're always forward-looking, and yet, actually, most of what we do, today's work is mostly created by yesterday rather than the future, if that makes sense?

It's based in part on an idea from John Seddon, a systems thinker, that a lot of what we do is failure demand as opposed to value demand. We are working because of something else not working or causing problems, so therefore we have to do more work to respond to that and build on that, rather than things that are, actually, genuinely novel and distinct. We are taking one step forward, but that's the overall effect. Actually, we're taking eight steps forward against the seven steps we just took back that then it feels a little bit like that.

**Zoe:** I think it's very interesting this idea that we're all caught in this massive pace of change and almost every one of them individual, you should say, "Oh, wow. We should all slow down and fix up all our code bases." Actually, as an individual, we are always just caught in it and we don't have that ability. What was the implications of that? In terms of, as a developer, what's your moral responsibility in terms of coding for the future?

**Kevlin:** Oh, that's a good question because I think that there is this idea, again, this is a difficulty we have with being human. Is that when we do try and think of the future, we either draw a blank or we tend to latch onto one particular idea of what the future will hold. It's normally a personal preference based on our experience. We see this in code bases that a lot of complexity that gets added is not simply cumulative.

That's the archeological layering that we're just talking about. Some of what gets added is speculation, and speculation is this wonderful capacity that human beings have to imagine alternative possible futures. That's a huge cognitive breakthrough. This is really powerful. The problem is that we then get attached to this and then put it in our code, and the code becomes littered with speculation. We sit there going, "What if?" We go, "Yes, that's great." What if somebody wants to--" I had this with one team many years ago, "What if we try and add a different persistence mechanism? What if we do this? Their code base was littered with interfaces and hook methods, and all bits and pieces, none of which were ever going to be fulfilled, or rather, only one future was going to, actually, happen. What they were carrying on a day-to-day basis with them was the weight, the cognitive weight of all of this speculation. They created more work for themselves or more importantly, other people who were coming after them in the future.

Whereas what we should be doing with speculation is using it to find the simpler solution, "What if we do it like this? What if this came to pass?" Well, I'll tell you what, I won't have that function because it'll be easier in the future when we know what we need, but there's also another element that that notion-- Either we over design, but we overdesigned for one version of the future, or we fail to account for probably most often shifting assumptions.

The moral responsibility here is very much, it goes back to what we were talking about before, is the way that pretty much most systems these days are connected. Now, that was not immediately obvious a few years ago. That was not obvious that everything would inevitably end up connected in the way that it is now. The idea is that, "Yes, we have PCs. Yes, we can connect to other servers, but we still ran applications."

Now, with the rise of open source software, little pieces of code that a developer somewhere puts together and if the open source is a gifted culture, generously gives to the world, then you don't know where that's going to end up. It might, actually, have some limitations. Refer to the left-pad incident in 2016, which was a piece of JavaScript that was for string padding.

Now, really, people shouldn't be using anything online for string padding. That's, I think you write in about two or three lines of code. Nonetheless, this was offered and it was withdrawn, and it pulled down a lot of sites with it. It was over a licensing disagreement. What I found interesting is investigating this, looking into this is, one, sadly, although it's a gift, the code was not great. It had some errors in it.

**Zoe:** It was maybe like second or third best present, not best present.

**Kevlin:** Exactly, yes. It's fine to put stuff out there, but it's that curious thing that sometimes when we put stuff out there, we don't know who's necessarily going to use it. We don't necessarily know how effective that code was. It was only when I, actually, tried to write a simpler version of it, just for a talk. I wrote some tests because that's the kind of thing that I do. I wrote some tests and I ran the tests and they failed for two or three cases.

I thought, "That's funny. This code is used across tens of thousands of sites and nobody's ever, actually, bothered testing it." That was a slight shock. I think there is a bit of a more responsibility there and sometimes it's accidental. One of the reasons I do quite more tests than I used to for talks is particularly one company. I used to throw together example code, where we'd be talking in a session, talking things about, "Well, I though they should-- Okay, you need something like this. We'd sketch it out, but I try and offer them code to make it a little more real."

Now, this particular company I have in mind, I have a long relationship with. There I am two years later doing a code review and suddenly there's this kind of disconnect in the code style. I'm looking at it going, "Well, this is really different to all your other code. It's not consistent with it. In fact, it's also really familiar. This is my coding style. This is the coding style I have when I'm not trying to fit in with somebody else's coding style."

As I've written this as a standalone example, I wasn't trying to fit in with their style, but it's there in their production system. This happened twice more at this company. That code I had given to them as just like, "You need something like this and become this." Pretty much from that point on, I wrote tests for everything that I gave them, even if it was throwaway code because it turns out one of a phrase we use in software a lot is, throwaway code.

Very rarely does code ever get thrown away. You can ask somebody, ask your colleague, "Throwaway code, you were working on a couple weeks ago, "Do you still have it?" "Oh, yes, I do," but they didn't throw it away. The throwaway metaphor is very clear on this. It turns out this stuff can't last forever. There's an accidental more responsibility I discovered I had in handing over code. It can't just be a suggestive, "Oh, something like this. Actually, it needed to be actually production ready." We have these kind of ideas of changing context. We get what I think is a whiteboard context suddenly becomes a production context. What I also think is that we are surprised by changes and we need to keep track of them. I worked on some systems number of years ago, SCADA systems that manage that, basically deal with electricity network. We put in pretty light security because we didn't think anybody would ever be done enough to put something as fundamental as the electricity grid on a public network. We genuinely thought that was not a thing. We just thought you'd have to be mad to do that.

**Zoe:** [laughs] I knew where this story's going. [laughs]

**Kevlin:** Yes. Well, fortunately, before it had a very bad ending, that realization was the bandwidth also got better. One of the reasons we cared a great deal about, I put together this protocol, it was very, very lightweight and it wasn't very secure. We had all these discussions because we were pretty much communicating down wet pieces of strip. Using a security-based encryption would've, actually, chewed up our bandwidth, but we thought that shouldn't be a problem. Fortunately, bandwidth got better as the realization dawned, but, actually, people were going to put this on public networks and that was a possibility. That was a change in context that we had not estimated.

I will say though, in our favor, we not only anticipated the year 2000 problem, we also anticipated the year 2038 problem. We were ahead of most of the industry, but we had a one blind spot, which was how people connect and whether or not we need this level of security, and that was a shift in context. I think see this every day, particularly now where an algorithm has a cloud footprint. [chuckles] Everything's getting into the cloud, it turns out the efficiency of your code- [sound cut]

**[pause 00:21:40]**

**Kevlin:** -has a consequence that goes beyond your PC, for example.

**Zoe:** Well, I think also, there's this small-- you're talking about small changes in context, but we also get hit by large changes in context that we can't predict, which, obviously, we've just all been through a really large example of the pandemic and how we had to respond, I guess, in a physical sense, people, to the presence of this pandemic. What was your experience through that?

**Kevlin:** I think that was quite interesting because I think the pandemic is one of those experiences that affected everyone differently. It's everybody has a slightly different story. For me, in theory, I work for myself. In one sense, there's the idea of like, "Wow, you're working from home in that sense, so perhaps it doesn't affect you. What turns out that what I do is, actually, on people's sites.

In other words, I run workshops, I run training courses, I do consultancy, and I give talks. That's geographically, although I am based at home, that is my official business address, the delivery of a lot of what I do is based on site. That had a huge effect on me because initially a lot of people counseled about work because there was no sense of how long this is going to go on for.

Then that vacuum, reversed where everybody suddenly realized, "This might be going on for a bit longer." Actually, we probably still need to keep people, give them the social sense. Suddenly I found myself doing, funny enough, a lot of talks. Company wanted a lot of talks to say, "Here's the thing, give our developers something, other than just sitting at home working on stuff."

For me, that changed the balance of my work, but also changed the nature of my work in terms of running workshops, and training courses, and consultancy. I spread it out because that's something that I don't need to be geographically co-located for. Suddenly I discovered that the shape of my work changed. The interesting thing was getting jetlagged without leaving my own house.

That was an interesting one. I'm not entirely sure I recommend it, but it was a low calm footprint. Doing something in Australia meant me getting up very, very early and, actually, finished, working in half days. I finished my half day when my family got breakfast. That was an interesting one, but for other people, they have been able to continue their work at home. Some people who are already working from home, in other words, developing, their primary focus was development or they were already distributed companies, or quasi distributed, and they were already halfway there. Now, obviously that doesn't talk about the mental health consequences of this rather strange collective experience we've had. From a functional day-to-day point of view, there was like less disruption because their primary focus of work was already contained, was already handled by the distributed nature of saying, "Yes, they could already get their code."

They had already been accessing their code from wherever they were. That was already something they'd been doing. They'd already been working on the system they were working on using the tools that they were working on. There was clearly a bit of a disruption, but not as much as other people experienced who were very, very office based, and for whom the company setup was very geographically based.

There's a very strong office presence in some places and perhaps a little weaker in others. There's this huge disruption that has been mediated, if you like, by technology, and coming out of it, you end up with some companies have said, "We're going to do mix of work from home and offices." The role of the office or being together physically co-located has shifted a bit. Whereas other companies want to hit time machine button, go back and say, "Actually, we want it exactly as it was three years ago, and that's how we're going to work.'' Other companies have decentralized completely.

We're in an industry that loves to talk about disruption and this is a real disruption. This is not-- I often joke about the idea, often people like to talk about Uber being disruptive and it's just like, "That's not disruption. Be able to get a taxi via a handheld device was-- Yes, I think we knew that was going to happen. From a technology perspective, there was nothing disruptive about that. When we read the Uber files, it turns out the disruptive elements of Uber were, actually, their business practices which were somewhat questionable. Let's just put it like that. That was the disruptive element.

When a real disruption came along, I think we were one a little bit surprised, but also in one sense, curiously well prepared at one level. I think technology had this happened 10 years ago, it would've been very different. I'm thinking of the schooling experience of my kids or my younger one at least, and, actually, university experience of my older one. 10 years ago, the ability to continue education in any form whilst not present in school, would've been far, far harder.

Although we currently still have a haves and have-nots situation, it would've been much more the have-nots that were in the majority in terms of access to appropriate technology and the space in their home where they could be studying. That is the number of digital devices per household now versus 10 years ago is radically different. 10 years before that, even more radically different. People are still using dial up and stuff like that 20 years ago. There is a huge shift in what technologies allowed us to do to weather this.

I still think that the consequences of it, I don't think we fully work those out. Some companies are being very open, very forward looking. Others are trying to come down with one rule that simplifies everything. You've got to be in the office these days, or you've got to be in the office all the time, or you've not got to be in the office at all. We're not quite there as in having negotiated a recognition of how much flexibility people want, but also how much sociability they want. We've learned that human contact matters. We've also learned that work can be flexible for certain disciplines and technology as one of those disciplines.

**Zoe:** Everyone's affected differently as an individual.

**Kevlin:** Yes.

**Zoe:** Something was taken away from us and some people have gone, ''Huh, do you know what? It turns out I never needed that and some people have gone, "Oh, my goodness, it's back. It's back. Don't ever let it go away again."

**Kevlin:** Yes. I, definitely, had that reaction myself. In other words, I've adjusted my work patterns and I enjoyed the fact that, actually, I got into the habit of traveling and I was just like, "Actually, I don't need to travel that much. I can reduce that, but I do human contact." It's trying to find that balance, it turns out there are a whole load of things that were not necessary and now I know which ones are necessary or I feel are fundamental to the way that I work. It gave us an accidental moment of clarity.

**Zoe:** Lets avoid disruptions and, obviously, I appreciate that to look forward at all, we fall into the exact trap that you mentioned earlier in that by predicting things we can get fixated on them. It is looking pretty likely that there will be an upcoming recession.

**Kevlin:** Yes.

**Zoe:** What is the answer? [laughs] I mean, what answers, I suppose, can technology bring and what's the impact going to be on people working in technology?

**Kevlin:** Ah, that's an interesting one. Predicting the future is always interesting, but at the moment, all of the signs are, definitely, in a number of countries, but particularly the UK, recession is very much on the cards. All the indicators point in that direction. What is interesting from the technology point of view, is I think a lot of people working in technology are going to be insulated from this because technology is the thing that is still on the rise.

There is still a lot of demand for new software, new sites, as it were to either automate things or make certain things more possible. The connectivity kind of feeds itself, if you like, and that's one of the main drivers is not just people wanting a software application, it's the fact that everything is connected to everything else. It doesn't matter if you are a health service or you are trying to send people pieces, if you are not online, then you are aware in one sense.

The demand for people in software and associated technologies is going to continue rising in spite of, and perhaps in some quarters because of the recession. That's the thing. The one thing is that this now gives a number of technologies, perhaps an accidentally privileged view is that they may not see the recession that is happening around them, they may not see the cost of living crisis that is around them in the same way that other people do and experience it.

I think there's a lot that can be done, and, certainly, streamlining access to services where they exist is one of the things I regard as incredibly important. We have this wonderful magic that we've created that allows us to access all the world's knowledge and pretty much all of the services in theory, but in practice, we are still frustrated by bureaucracy and barriers that get in our way.

People are trying to overcome those and they can't do it politically then it turns out that technologically is the way that people are often going to go. I don't have anything specific to offer, I think, and I wouldn't be very cautious about saying technology is the answer of things because I think it's also created a number of problems. It a blade with at least two edges, quite possibly more.

**Zoe:** I love this. Technology is the answer and it's also the problem. [laughs]

**Kevlin:** Yes, pretty much so. Yes.

**Zoe:** You get a pandemic, you solve it with technology. You get a reflective, you solve it technology.

**Kevlin:** Yes.

**Zoe:** Kind of a decline. [crosstalk]

**Kevlin:** Some of these situations, they were technology or demand of one kind or another that led to it. These causes are not very direct in many cases. Sometimes they're indirect, but they do fall a path.

**Zoe:** Something that's very closes to my heart and I feel like touches on all of the challenges in technology or challenges of working in technology and trying to do the best job and get the best outcome for our stakeholders, which, like you say, for a lot of pieces of technology is the whole country or very large, or help the whole world in some cases. Something I'm very keen on is, how do you do the best of that, that you can? How do you help bring stakeholders with you and I suppose, get the right decisions made?

**Kevlin:** Yes, this is an interesting one because I think it's a business question that goes back at least as long as people have realized that there's a question to be answered there. I think a lot of it comes down to communication, which at this point, I can imagine a motivational poster popping up in the background. It's more to do with the quality or the nature or the purpose of communication. I think, we don't just want more communication, we want to be very clear about what it involves. There's that element of the idea of working, whoever the stakeholders are is that idea of working with them. Whereas in a lot of businesses and particularly where we are dealing with manufacturing or physical items, not even just manufacturing. I mentioned pizzas earlier, actually, that that also fits into this, where we have the idea of physical handoff and in software. Software is knowledge work.

We are normally trying to shape and understand somebody else's domain bringing to it knowledge about a technical domain. In other words, "Okay, this is how you do your business. These are your objectives. Here, I've got this concept of architecture and these frameworks, and these programming languages, and experiencing this. I'm trying to take that knowledge and trying to work out how can that knowledge meet this business and shape it and help move it forward."

It's very flexible stuff. It's not product that you hand off. It's thought stuff. It's knowledge. There's this idea of genuinely working with somebody rather than receiving a handoff. Now, historically, I've seen this in a number of cases where people say, "Oh, well, I hand off the requirements to," or we ask them to build us. They're working for the same company. This is always interesting. Whenever you have them and us and in the same company, it shows you, it's just like, "No, it's all you."

You are all one. It just happens that, and you get with developers as well. The developers talk about the business as if it's separate, as if they're outside when they're working for the company. In most cases, they are the business. This is something I've done work in the past for banks, and I remember there was, 2000, there were a few reports floating around from these self-styled white paper based companies floating around about IT as an overhead and all this, rather, it's an add on. It's an overhead. It's a separate entity.

I remember thinking at the time, it's like, "If you don't have IT, you don't, actually, have a bank. You don't, actually, have a business." Whatever is true then is even more true now. There is no them and us. It needs to be understood in that sense that, "Yes, you are working. You are on the team, basically. If you are a stakeholder, you need to consider yourself more to be on the team. Knowledge, it's not a pizza that you hand off to somebody else.

Knowledge is very fluid. Quite literally, very fluid. There's quite a good way of demonstrating why handoffs don't work in this case. Particularly in large companies that are hierarchy based. If you imagine cupping your hands and there's, obviously, a lot easier for people who will be doing this video, but you can imagine this. Cup your hands and I'm going to pour some water into your cupped hands, and you can retain that.

There's a little bit that leaks out, one, that's not all the knowledge that I have about my domain, but I'm going to give you some of it. Some of it going to leak out, then you are going to pass it on to somebody else and say, "Here's what I've understood from Kevlin." They're going to receive it in their hands and they will get a little less. If you keep doing this, you end up with, the person at the end just gets a wear handshake and nothing more.

What's happened is that loss of knowledge. You get that you retain that by all being in the room at the same time. This is the thing that we saw many approaches trying to do, is the idea of a sprint demo in Scrum, but that's still at the end of something. There's this idea of working much more closely together. Extreme programming, actually, in the late '90s gave us an idea of what this should be like. They had this idea of an onsite customer, or the customer is on the team.

You have the stakeholder there all the time. I've seen this in a couple of places where people were, I think the term used, I think it's formed by the wayside, was peer programming. You had a domain expert and then you had a programmer and they were together. Clearly, one knew the domain and the other knew the code. They were, basically, having an extended conversation that was handed into code and then run. That level of cooperation is a very different thing.

**Zoe:** Working as a team.

**Kevlin:** Yes.

**Zoe:** Yes. Essentially, just like all the technologies that we're working with, we are all interconnected and interdependent. I think it's so important and so important to remember that from the people level. That's been so fantastic. Thank you so much, Kevlin. I have not enjoyed a lunchtime this much for a long time. Absolutely, fantastic.

**Kevlin:** Thank you, Zoe.

**Zoe:** I'd like to remind everyone that we have some more air coming up at *Softwire,* and you can see those on our website. On the 20th of October, we're going to have a talk on artificial intelligence in our Softwire Manchester office. You will need to be physically located in Manchester for this one. We have a speaker, Sean Williams, who is the Founder and CEO of AutogenAI, talking about the technology that underpins what they're doing and the changes in AI development, which is super exciting.

We then, on Thursday, the 3rd of November, we've got the breakfast event covering accelerating digital innovation in the media and entertainment sector. All the details are on our website, www.softwire.com. Yes, thank you all so much for joining us and enjoy your day.