EPISODE 692

[INTRODUCTION]

[00:00:00] JM: The Windows Operating System is one of the most widely used pieces of software in history. Windows was started before there were any alternatives to a monolithic codebase, because this was before the internet was widely used by consumers and by developers in their software engineering process.

Networked computers gave rise to web applications and that made software engineers begin to rethink how to build everything. After the web was widely used in software development, software development got reimagined with Agile. Monolithic codebases got broken up into service-oriented architectures. Instead of going to a store to buy a box with software in it, users also were downloading software from the internet, and that software was regularly updated.

Software that's regularly updated needs to be regularly tested. Instead of a single round of testing for every round of boxed software that was shipped to a store, continuous testing and delivery gradually became the norm. The process of releasing and operating software become its own set of engineering challenges, and this was tackled by the operations or CIS admin team at a software company.

This made there be two different sets of engineers, those who were developing the software and those who were operating the software. The incentives of these two types of engineers were not completely aligned. The software developers wanted to build software quickly and release new features and the operators wanted things to be released more slowly, because if something broke, the operators, the CIS admins, they were the first line of defense for fixing those problems. These problems between development and operations gave rise to the DevOps movement, in which developers and operations started working more closely together and sharing responsibilities. Incentives became more closely aligned and new types of software was created to facilitate more harmonious relationships between developers and operations.

For example, continuous delivery pipelines. Continuous delivery pipelines were a newer kind of software built around the time of a movement towards widespread DevOps. Today, most enterprises are still undergoing a transformation from monolithic software release cycles to continuous delivery. This is often referred to as a DevOps transformation. A DevOps transformation requires the entire organization to reorient itself around faster software release cycles, and this could be a painful process. We've covered it in many past shows in several different case studies, and hearing case studies from these different enterprises can be helpful for figuring out how to reorient your own enterprise.

Microsoft is a useful case study in shifting towards DevOps. Windows is perhaps the biggest monolithic codebase in history, and the fact that Microsoft could re-architect Windows to be easier to work with, should provide some reassurance to other enterprises that are currently undergoing their own migrations.

Martin Woodward has been at Microsoft for 13 years and he joins the show to talk about how software delivery within the company has evolved. We discussed the move from boxed software delivery to delivery via the cloud and focused on a few specific longstanding products, such as Windows. Martin has been part of the effort to build Azure DevOps, which is a product that offers similar tools to the once Microsoft builds internally for DevOps as a service. We also talked about the specific difficulties that enterprises often have when moving towards DevOps.

Full disclosure; Microsoft is a sponsor of Software Engineering Daily.

[SPONSOR MESSAGE]

[00:03:54] JM: DigitalOcean is a reliable, easy to use cloud provider. I've used DigitalOcean for years whenever I want to get an application off the ground quickly, and I've always loved the focus on user experience, the great documentation and the simple user interface. More and more people are finding out about DigitalOcean and realizing that DigitalOcean is perfect for their application workloads.

This year, DigitalOcean is making that even easier with new node types. A $15 flexible droplet that can mix and match different configurations of CPU and RAM to get the perfect amount of resources for your application. There are also CPU optimized droplets, perfect for highly active frontend servers or CICD workloads, and running on the cloud can get expensive, which is why

DigitalOcean makes it easy to choose the right size instance. The prices on standard instances have gone down too. You can check out all their new deals by going to do.co/sedaily, and as a bonus to our listeners, you will get $100 in credit to use over 60 days. That's a lot of money to experiment with. You can make a hundred dollars go pretty far on DigitalOcean. You can use the credit for hosting, or infrastructure, and that includes load balancers, object storage. DigitalOcean Spaces is a great new product that provides object storage, of course, computation.

Get your free $100 credit at do.co/sedaily, and thanks to DigitalOcean for being a sponsor. The cofounder of DigitalOcean, Moisey Uretsky, was one of the first people I interviewed, and his interview was really inspirational for me. So I've always thought of DigitalOcean as a pretty inspirational company. So thank you, DigitalOcean.

[INTERVIEW]

[00:06:01] JM: Martin Woodward, you are a principal group program manager at Microsoft. You're a vice president of the .NET Foundation. Welcome to Software Engineering Daily.

[00:06:09] MW: Hey! Thanks for having me, Jeff.

[00:06:11] JM: You've been at Microsoft for 13 years, which is enough time to see a variety of evolutions of the company. When you started there, what was the release process for products? How did products make it from development out to the market?

[00:06:29] MW: Yeah, I joined Microsoft from a startup. We had a company sort of five of my friends doing Eclipse tooling, believe it or not, and we figured, "Hey, that would be a safe place to work where Microsoft won't want to buy us," but they came along and want to buy us anyway. So that was an early indication that times were changing, I think. I came from shipping to the web all the time. I was shipping into the Eclipse marketplace, but doing instant updates kind of thing.

When I arrived at Microsoft, it was still very much – This is in November 2009, and it's very much still a shrink wrap culture. I'm the program manager. So sort of my job is to ship the right thing at the right time and you kind of have to figure out what the right thing is and when the right time is, but it's a pretty easy job, ship the right thing and the right time.

So part of that involved being the box PM, so being the person, and that's literally what it was called, a box PM. So that kind of says everything you need to know. For the product that Microsoft had acquired, that I brought in to the company. So I had to go on a training course to learn how to make it so I was able to sign off on the hologram that was used for the DVD that that software had got burned on to. That was printed in Costa Rica and Dublin and the boxes and all that sort of stuff, you know, doing all my Ad-Law courses so I could approve the, all the text that went on the side of the DVD cover. When we were shipping software, you literally called your release done.

You put that into a system which took a couple of days to sort of move binaries from one place to another, and then landed in Costa Rica or Ireland, and they started manufacturing it, putting it on to DVDs, printing them and putting them into boxes and shipping those boxes to suppliers who then solved them. That was very much the culture of the company coming from that kind of mindset, which is kind of expensive to fix things. So you start operating in the meantime between failure is the thing you care about. You don't want to ship – The thing you ship has to work because it's going to be printed on boxes and things. If you have to have a recall where it involves shredding DVDs and things, then that gets pretty expensive pretty quickly. So it's a completely different mindset.

[00:09:01] JM: So the distribution format informed how the software development process went. So it made sense to do a waterfall software development process, or would you call it waterfall?

[00:09:16] MW: Yeah. The most of the company was probably waterfall at that time. There were pockets of Agile. Our team ran in an Agile way, and so internally we're always potentially shippable. We would run a lot of dog food bills instantly and things. But having that mindset of a thing that gets released does lead you down actually into a waterfall instinct if you don't sort of fight it. It's more of it's the meantime between failures is the expensive thing, and so you have processes which optimized for not failing rather in a DevOps world and the modern world. You really are optimizing for how meantime to fix, how quickly can you fix issues rather than the meantime between failure. So it’s a very different – It’s a subtly different thing, but it leads to very different results.

[00:10:08] JM: When was the company able to focus its software development processes towards more of a cloud-based delivery model and when did that cloud-based delivery model start to inform the development practices so that it can move to a place where it was less of a waterfall, less of a shrink wrapped development model?

[00:10:31] MW: Obviously, Microsoft is a big company, and so different parts move at different speeds. I was over in sort of developer tools land, which kind of was pushing ahead here and we were very closely affiliated with the Azure people and what was sort of the Windows server people, and that became the Azure people. So they were obviously a lot more on the outside.

This was around November 2009, and I come from a little startup. I wasn’t sure – I had like the deal you sign when you join a big company like Microsoft that kind of requires you to stick around for two years if you want to see any of the money. But I was kind of thinking, “Well, we’ll see how it goes. I’m not sure how I’m going to cope joining this massive big company coming from cheeky little startup land.”

But it’s been amazing and that I kind of joined at just the right time, because it was just as that transition was kind of kicking off. So my team was doing the Agile. We were after the Agile. Then the rest of the group I was doing sort of started that had pilot kind of being of an Agile, and then they went full on. We’re going to change a whole sort of 500 people all go over from shipping in a more Agile way.

Interestingly, the key thing is that it wasn’t sort of change. We become more cloud-focused, which then means we can become more Agile. It was we saw we needed to be Agile to be successful. So we changed our business and our business models to allow us to be more agile. So we kind of drove one drive over, which is interesting.

[00:12:01] JM: Well, this was 2009. So I guess that was right around the same time that the DevOps movement was getting started. Were you reading about DevOps then? Was that information starting percolate to you?

[00:12:14] MW: I was in like the Git space and the Eclipse space. DevOps not so much as a term. The term then was a lot around application lifecycle management and things. So I come from sort of Agile and TDD, and then the transition from Agile to more DevOps practices, I want to say probably happened for my group around – Probably around 2014, because maybe – Yeah, that's when we kind of – Yeah, 2013. Around about then, because that's when we started running a live service that needed to be up 24/7 and all those sorts of things, and we started to need to learn kind of practices for keeping an always running cloud service running, and that's kind of when I started digging into that space.

[00:13:01] JM: I had trouble understanding what DevOps represented when I first started covering it, because I would just get these different definitions from different people and it seemed like –

[00:13:11] MW: Depends on what I'm trying to sell often as well, but yeah.

[00:13:15] JM: Yeah, exactly. I seemed to have something to do with Agile. It seemed to do with the continuous delivery. It seemed to have something to do with sets of tools. But overtime I think what I've come to understand is it was more like there were new tools, there were newer ways of delivering software, and those were informing a cultural shift, and then the cultural shift would also inform what new tools could be built, because there would be a change in culture and then people would say, "Wow! We really would like to do things in some new way. If only we had a tool for doing that," and then that gave a market opportunity for a tool to come into place. Then the tool would change behavior even more and then you would have this back and forth and it's really driven really rapid changes in develop productivity.

I imagine, you probably saw this to an acute degree because if you were coming from the Eclipse space, I imagine you were seeing – You had your eye on internal tooling, and I think of Eclipse, that was certainly a development tool that I used a lot back in the day. Well, back in the day as far as don't go back very far. But the tooling has – Eclipse was like a cutting-edge developer tool back then. Now there's like a bajillion developer tools you can have, and Eclipse was such a predominant one before.

Maybe you could about how internal tooling, when you were at Microsoft, you see the cutting-edge of internal tools. How did the evolution of tooling inform the movement towards what some people were starting to call DevOps? Maybe not in 2009, but 2010, 2011, whatever, whenever you start hearing about it?

[00:14:56] MW: Yeah. Regarding the DevOps thing first, we sort of – The way I think of it is the people process products is what we say, and that's an order of difficulty, but I'll carry on. Then the iterative continuous flow of value to your customers. How do you get value into your customers' hands as quickly as possible and keep iterating on that? A continuous flow of value. The DevOps movement is all about continual improvement, continue to getting better. Nobody finishes a DevOps journey. Nobody certainly is doing DevOps. DevOps is the journey to get – Is a journey to improve for getting this.

So if you're doing DevOps right, what you're doing is you're continuously looking about how to improve and how to improve those processes. Then that's what leads to what you were saying in terms of you're constantly iterating and then you get faster and faster and faster, because it's accelerating. Then it's definitely people process products, because the people are the hardest thing to change and sort of the wetware in there is always the sort of the stickiest bit. So you've got to be able to fix the sort of people. You've got to change the processes about how those people work and then you've got to provide products which support those people and those processes.

The products is kind of the easy part and kind of you can make do with – You can make do with current gen of products, and then you find yourself building new gens of products. You buy products off a shelf, you can use open source ones, you can do whatever and assemble those together, which I guess is a weird thing to say for a guy that works in this space that sells these products, this tooling, but that's the easy part. But anyway, it kind of is. It's the people and the processes is the hard part. You just go by or you acquire some tooling from either open source or from a company like Microsoft or something.

So in terms of how we saw the change, again, it was one of these ones where it kind of it was iterative in itself. So we knew we needed to be fast. We knew we needed to ship quicker, because if you're trying to predict where the market is going to go in three years' time, which is what you have to do with boxed products, because it's so expensive to print and then manufacture and ship. If you're shipping every two to three years, you're trying to predict where the market is going to be in two, three years' time. Of that prediction, you're going to get it right 20% of the time and get it wrong 80% of the time say, but a roughly guess.

If you're able to ship twice that speed, then you're going to – Within the same period, within that same two to three years, you're going to have X amount more good features, because you've seen what works and then you fix it and you move on and you discard what doesn't. So we very much saw that that kind of thing was working.

When I joined the company, it would be fair to say – Again, this is me going from start Startland not really a Microsoft person at the time. I was fairly horrified at the state of internal tooling. You would think Microsoft would have the best internal tooling in the world, which hopefully it does now and it's always getting there, definitely. It's industry leading.

When I joined, that wasn't the case. It was kind of like – There's a saying in England and the U.K., the cobbler's child has the worst shoes. The shoemaker's children have the worst shoes kind of thing. Because they hired a bunch of smart people to do smart things. They were busy working on products and kind of working on the build and tooling that was building those products. It was very fragmented into different groups inside Microsoft. There was – You've got, say, like big teams, like Windows and Office and developer division, which is building on dev tools and people like that. Everybody would have their own build teams and their own tooling teams and their own reporting teams and they all kind of work differently.

Every area kind of have a very, very different ways of working different sets of tools, different methodologies, different restraints. If you were trying to move between areas inside the company, it was almost like just starting brand new in a new company. It was a completely different tooling set.

One of the things we've been doing is kind of driving a couple of metrics, kind of like improving, looking at the time it takes for a new hire to engineering. When is there – How long does it take for them to get code running in production? Then for a transfer for like somebody else in Microsoft, how long does it take for them to get up to speed and get code running in production?

The goal for new hire is two business days. That's where we wanted to get to. Bear in mind, when I started, it used to be three years. Because you would start on a team and then you got hit the next ship cycle depending on – Up to three years sort of thing, two to three years, because you have to get some code in there and it had to be built and then it would maybe get into a bit or something. But by the time it shipped, it could be two or three years between release cycles. Yeah, which is very different.

[SPONSOR MESSAGE]

[00:20:19] JM: For all of the advances in data science and machine learning over the last few years, most teams are still stuck trying to deploy their machine learning models manually. That is tedious. It's resource-intensive and it often ends in various forms of failure. The Algorithmia AI layer deploys your models automatically in minutes, empowering data scientists and machine learning engineers to productionize their work with ease. Algorithmia's AI tooling optimizes hardware usage and GPU acceleration and works with all popular languages and frameworks.

Deploy ML models the smart way and hardware, and head to algorithmia.com to get started and upload your pre-trained models. If you use the code SWEDaily, they will give you 50,000 credits free, which is pretty sweet. That's code SWEDaily.

The expert engineers at Algorithmia are always available to help your team successfully deploy your models to production with the AI layer tooling, and you can also listen to a couple of episodes I've done with the CEO of Algorithmia. If you want to hear more about their pretty sweet set of software platforms, go to algorithmia.com and use the code SWEDaily to try it out or, of course, listen to those episodes.

[INTERVIEW CONTINUED]

[00:21:54] JM: Now, I can imagine, this is really hard to ordain from the top of the company how people are going to develop software, because Microsoft's products span from the pre-internet era to the post-internet era. So if you have a product like Windows, or a product like Microsoft Word, you can't necessarily map a continuous delivery to those products as easily as something that was born on the web. It may probably be even something like Skype. Was there difficulty in establishing consistent standards for the different teams when you have these radically different constraints around the testing process and the sensitivity of the infrastructure that you're building and just the delivery model of those different pieces of software?

[00:22:52] MW: The difficulties are mostly, again, we told people process products. The difficulties are, first of all, with the people. Just getting them to understand, "This is how we've always done it. These are the processes we always follow with these tools." Getting them to sort of sit back and think about like is this the best way of doing this? Do we need to change our business model to help us sell our products differently?

When we first started shipping Windows, the way Windows were shipping was it would landed on your machine. Then when the internet came, we were able to sort of ship ISO images and things. To think about an ISO image, that's a very direct analog. It's literally a direct analog of the DVD that they were physically shipping to before. If you think about modern internet connected operating system, there's no reason why or has to ship in one image. Why can't you ship enough to bootstrap and then download individual components? Which is kind of how Windows is moving and how it's moved in terms of there's a lot more store apps being there, there's a lot more things that get individually updated rather than just being part of the core OS.

Yes, but it was difficult mostly culturally. I would say that the company is moving at different speeds depending on the products that they're servicing. Even within a group, like the Office group. You've got the people who are building the office web apps who are literally bleeding edge JavaScript developers right there at the very front of how to build really deep complicated web applications. You've got other people who are building the client side tooling that we all know and use. Then you have the work within the Office team to try and have a – Not completely rewrite everything every time to have a codebase that's kind of shared and how to do that.

Those are long-term architectural things that take many, many, many releases to get there and you kind of have to – Making baby steps to get there, because you're trying to do continuous value. Trying continuous flow value, because you can't just stop and rewrite everything and throw all the old thing away. You've got to be making customers lives better every single day.

[00:25:10] JM: Could we go a little deeper on maybe if you have some like a case studies – I don't know, Office, or Windows, or Minecraft, or Bing? Any teams that come to mind for how they shifted their development model?

[00:25:25] MW: Yeah. I mean, the one I know best is my team, which builds the Azure DevOps tooling and how we kind of shifted from shipping an on-premises product called Team Foundation Server on a DVD, which is installed – Which people install client side. So then running in the cloud as a bunch of cloud services. Our transition sort of architecturally has gone very much from – Is very much influenced by where we came from, but also we still ship the on-prem product on the cloud versions from the same codebase.

So if we were cloud native, born in the cloud kind of thing, then we'd be using all the platform as a services from Azure and from your cloud provider, because you get much greatest density of compute there. So you get much greater cost savings. It's all about the amount of electricity you're burning at the end of the day and how much about you're turning into real work for your end customers.

Because we have a product that ships on-prem as well as the cloud, the most dense unit of deployment we can kind of get is probably containers, but we still have some services, which aren't containerized and are running as VM images and multiple sort of sharded VM and images. That's one transition. Then the kind of the thing we did there, and our was very much an agile transition first followed by moving to a cloud service and then the sort of learning how to run a service and building that live site mentality. Customers are experiencing in the live site is all that matters.

When you're used to shipping products, it's very easy to think about features and things like that, what's getting checked in. It doesn't matter what's checked in. It matters what experience that customer is having right now in the world, and building that mentality was probably the biggest shift for our team, because we had to learn how to run a service now rather than how to ship a product.

There're other groups like Windows that do that less. So they've moved from shipping once every few years to shipping – They do ship multiple times per day in terms of the things like signature updates and store and service updates and then they sort of have a service – Windows now have a servicing update every week and then you'll know about sort of patch cheese day and things like that and the Xbox updates that go out monthly. Then every sort of couple of times a year, they do these big feature updates, like Redstone 3 and things like this. That's kind of the free conceivers that they change from those teams. But from our team, we had to go over next full hog and DevOps and learn how to run a live cycle, which is a huge change.

An interesting one that our customers we speak to, they already know how to run live services, because they're internal IT in a lot companies in a lot of the enterprises and all that sort of stuff that used to running services, we really, really weren't. So that was a big culture change for us.

[00:28:28] JM: How did you do that cleanly? Because I've talked to a lot of companies that have – They've gone through some – That's even more a complex migration than many of the companies I've talked to, because you're going from this product that has to – Constraints that you need to run on-prem as well as in the cloud and you're building it for customers. A lot of the companies I've talked to, they're just building like an internal application, like an insurance, like they're an insurance company for example and they're trying to move to the cloud, but they're not selling software to developers.

You also want to keep your software in a place where it's at least comprehendible enough so that, as you said earlier, if I'm a new developer and I'm joining Microsoft and I'm working on Azure DevOps, then I need to be able to understand the stack or at least the portion of it that I'm working on in a couple of days so that I can ship code very quickly.

How do you do a migration like that where you want to keep the product accessible to the users that have been using it for a long time while also updating it in a way that it makes it easier to ship in the cloud for example?

[00:29:47] MW: Yeah, the classic DevOps way of finding a bit the hurts the most and then repeating that until it stops hurting as much and then seeing what bit hurts the most next. It took us – Where are we at now? As we're recording this, it's the end of September 2018. So go out in October 2018 sometime? We started this journey in, say, 2010. So we've been doing it 8 years and we're a long way from done. Continuously getting better.

In terms of the stack, making the stack easy for engineers to understand, there were a few sort of epochs, if you like, that had to come in, a few c-change events. There's only so much you can do from continuous iterations. Occasionally, you need a big change. Moving a lot of our engineers over to Git as the version control system was a very deliberate decision to make it easier for new engineers coming to our stack to be able to know how to check in code and know how to do a poll request and all those sorts of things.

Through a team like Windows, that was a ridiculously technically hard problem. They have – What's the last number? There's like over 500 gig of source and about 6,000 repos. The big data is the windows core repo, which is where Windows as you know it, the thing that you look at if you're running on a laptop or whatever, the operating system. That's a single repo that's got 360 gig git repository at the center of it, which is just nuts.

Anyone that knows git would tell you trying to do that is just stupid. Remember, I got drafted in, and I was like the third person on the ground when the Windows team were like, "Hey, we want to join forces and we want to a single engineering system across our whole company and we want to base it on top of a modern stack, and we'd quite like to use git." I'm like, "You are insane. Why would you ever think that that's a sensible idea?"

So there was a kind of movement as to, "Well, maybe we should modularize windows." I'm like, "Yes! That's exactly what you need to do if you want to use git, because there's no way you're going to fit in one repo." You're going to need to break up this legacy – You know, I'm going to say legacy code. I shouldn't call it that. A code base that has 25, 30 years of heritage break that up and turn that, and then you can have manageable repo sizes.

It turns out that's harder than just fixing git to making git work for really big repos, to have mono repos in terms of a computer science problem, because these layers where if you don't have the abstractions built in, if you don't have these interfaces and contracts built in, then trying to modularize things after the fact, trying to unpick that hairball of interdependencies is just too hard and too expensive and requires you to stop work, which is no good. You can't. You just have to keep shipping features to customers. We went and we fixed git, if you like. We modified git to be able to scale up to that size. Obviously, with git being git, we then contribute those changes back into the community to make sure everybody else can also scale that big, because we got an open source. It's no good as modifying an open source thing and taking it and then modifying it to make it work really big. If we didn't contribute those changes back to that open stream open source project, then the git project itself, the open source git project could diverge in a different route and then we'd be left with our little orphan solution that's different that nobody else uses and then we're back to how we were before with now a fork of git rather than "real git". The whole point of using git was to make it easy to bring people in from the outside and to make it easy for people to move between the companies. Interestingly, contributing the changes back to the open source git project was a key requirement of why we're adopting git in the first place, which is cool.

[00:33:50] JM: There was a show I did a while ago with somebody from – He was a Linux kernel maintainer and he was talking about the fact that, for example, you can't use GitHub to manage Linux, because the Linux kernel is too big, and I guess they have their – They also have specific collaboration constraints. So because collaboration for the way that Linux is built is so distinct, they have this strange email system that they use to collaborate with each other.

Anyway, long story short, building an operating system is hard and you have a lot of people and collaborating on it is really difficult. But what were the change? What specifically were the changes that had to be made to git in order to accommodate building Windows on git?

[00:34:41] MW: Yeah. Right. The Linux codebase is sort of a 640 mega repo, and then the Azure DevOps codebase is a 3 gig repo, and then you get up to Windows core, which is like 360 gig. So you're jumping up – Every time you go up an order of magnitude, you normally need like a complete re-architecture, re-platforming kind of thing.

On that Linux thing, I know Linux has its own communication mechanism. Well, Linux does work fine in GitHub as a thing. It's the way that the Linux community sort of communicate their patches and emails and stuff. There's a certain amount of – Again, I'm not directly in that community as a maintainer or anything. But there's a certain amount of, if you can figure out this process, then kind of you meet the bar to be able to contribute to the kernel, sort of self-selection thing going on there. It's almost deliberately not too easy to contribute, because then the quality of contributions is higher.

Whereas when we've got open source projects certainly have a really broad reach being able to contribute to them is vitally important to us, which is why GitHub kind of won that space, because they invented the poll request and they made that such an easy process for people to be able to contribute to open source projects now and for maintainers to be able to maintain them, which is what led to the large part, what led to the success I think.

What do we need to do really to get to the Windows, to get to be able to host Windows team? Git is a distributed version control system, which means that it doesn't require a central server. In many ways we kind of had to break that to get it to work. We kind of have to take the distributed nature out of it. What we did is virtualize the underlying git repository so that when you do a clone, when you get your source code down, you don't actually get the entire source code and all the history ever down, because that would be too big and would never fit – You wouldn't be able to buy an SSD big enough to fit that all. Never mind transfer of a network or make it work.

We had to do some changes to be able to virtualize that under the hood and then put those changes into core git itself to be able to support that kind of way of working. That was a thing called – It was called GVFS. I think it's – But then that was confusingly the same as a known virtual file system. I think they've changed it now to virtual file system, VFS for git or something. I should check what the open source name is. That's an extension to git, which kind of virtualizes the storage of the underpinnings.

But when we did that, that allowed us to do some of the neat things, like we could – We're taking to some extent the distributed nature out of a DVCS tool, which is a big weird, but it still works the same way it works. It downloads the binaries on demand from a central server when it needs to rehydrate the objects. A bit like sort of how Dropbox and Onedrive and things like that kind of work with these virtual pointers.

Then it also allows you to do some caching of those things locally in your local Office. One of the differences between, say, the Linux project and the Windows team is the Linux kernel is maintained by a disparate group of individual contributors. You know, there's some in pocket. Like there's a pocket of them over in Red Hat and there's a pocket who work actually in Microsoft as well. There're pockets of them. But mostly it's people who are individuals spread across the world.

The Windows kernel is maintained by teams and they're mostly in Redmond and a few other places around the world and in specific buildings within those specific geographical areas. It actually allows us to cache the copies of those files that they need local to them. So you end up with an interesting thing where even though the Windows to the core Windows team based in Redmond, and their Azure server, which serves that code base, because this is all stored in Azure DevOps, Azure repos, which is in Azure. But we deliberately pick the Azure datacenter closest to them to store their particular project, which is like West U.S. or something.

[00:39:01] JM: Just to give more architectural color on what you're describing. So if I understand correctly, what you changed in git to accommodate a giant product like Windows is this virtual file system. So if I'm a developer, normally I would have my entire codebase on my local machine so that I can explore that codebase as I want to and I can click around in it and then I can understand the sub modules and the sub-sub modules and son on and I can have lots of rich metadata in my development environment so that I can introspect classes and things like that.

The problem with doing that in a gigantic repo is that you simply cannot fit all of the repo on your machine. So you build a virtual file system, and the virtual file system indexes the metadata, or it has some metadata files that can tell you the relevant contents of those files so that if you actually open the file, only then will it be requested from the server on the fly, but it will seem a transparently like it's on your machine from the beginning, even though it is going over the network and pulling down the file, which can be really painful if you're somebody who's opening lots of different files and you haven't seen the files before. So you have to build probably like systems of caching, and of course it's really important that you – If you want to cache the entire repo on some data center that's the closest geographically to, like you said, the Microsoft, the team in Redmond.

[00:40:40] MW: The office.

[00:40:40] JM: Which illustrates this really interesting difference between Windows and Linux, where you could service those developers by just caching a piece of a version of the Microsoft Windows repo close to them, whereas Linux, people are all over the world. So caching a version of Linux close to everybody, maybe a bit more tricky.

[00:41:03] MW: Well, that's exactly what git does. The actual problem is slightly harder than – What you described is exactly correct. But I make it even a little bit harder, because as a developer, when you do a git clone of your repository from GitHub or Azure repos or whatever, you're not just pulling down your current version of a codebase, but you're pulling down every version ever of that codebase. You're pulling down the full history. Again, for a team like Windows, with 25 years of history, that's even worse.

The Linux team – [inaudible 00:41:34] You know what I mean? He was fundamental in the creation of git, and it works really well for the design for what the Linux kernel maintainers needed. Because they're individual contributors working distributed remotely, it is – They are caching an entire copy of the entire repository and all the history locally on their individual machines. That's fundamentally how git worked from day one. That made sense for the Linux kernel maintainers, because that's exactly how they work. That's now how the Windows team works. So we're a large enterprise team. There they have groups of people, very tightly geographically coordinated – Co-located, sorry. We kind of tweaked the way it was stored to enable us to do that. It's a bit slightly clever and that it has to – We had to do some changes into git as well into how it crawl trees and things, and git would break, because things were – You would hit lots of end factorial problems and stuff. So we had to change ways certain things were reevaluated to support really large trees.

I can give you some link for the show notes if you want that dig into some of the rather really glory details. It was some fundamental changes so – It's because the teams were very, very differently. Even things like branches, normally git would break, just breaks if you have more than, say, a couple of thousands branches. You think, "Who would need a couple of thousands of branches?" Well, the Windows team. Maybe I can just go look it up now. The Windows team got like 7,500 developers – There you go. Who are coding on those repos. Each person's doing – They work, do trunk-based development mostly.

So if you're doing a feature branch and then merging into master, you've got at least 7,300, 7,500 branches already, and that's just one develop working on one thing and then deleting that and remembering to delete their branch instantly. That's not even having any teams of developers working together before they then merge their thing in.

So we had to build some stuff into the server , inside your repos, to kind of hide branches by default. When you clone from Azure repo – Well, this is server side magic we ended up putting into Azure repos. When you clone for WDD code, the Windows codebase down, you only get the branches you care about, i.e., branches you've created or your team are working on, if you know what I mean. You don't get everybody's branches. There's a few little neat hacks like that we had to do as well.

While I'm here in the number, I'm just looking at a dashboard here. It's 7,500 people. In the most recent update of Windows, there were 4 million commits, so 4 million changes happened to that Windows codebase in a single – What was it? 6 monthly update? Which is crazy, isn't it, really? There were like – Just the build time alone for that update was 1,200 machine years of build time to build all the incremental builds that happened. Doing software at scale is just not – It really is, it's quite mind-blowing when you start digging into it. It amazing it all works, isn't it?

[00:44:41] JM: It sure is. Immense gratitude for that. So this is an extreme example that we're talking about, but it does illustrate the lengths to which developers are willing to go to get a better development process, and the DevOps movement is all about that. It's all about, as you've said, people, process and products. Yeah. It's worth doing this load of work, because otherwise you're not going to be nearly as productive as if you make that painful shift. This is what companies all over the world are realizing and at the same time they're moving to the cloud and they're also – There's companies like insurance companies that are going from being an insurance company to being a software company, and that shift is difficult. So there are all these difficult shifts that are happening. If you're on the Azure DevOps team, then you're in a perfect place to see the tooling that needs to be developed to allow these people to work more fluidly.

What does an average enterprise that is making this kind of shift? The average enterprise is not changing git to accommodate Windows.

[00:45:58] MW: No. Now should you. If you do, then you're clearly crazy. Do you know what I mean?

[00:46:02] JM: Right. They're talking about like breaking up their monolith at the insurance company and getting it into the cloud, which is its own set of really difficult challenges, but you can solve those in a much more generalized fashion than fixing git for the Windows team. What kinds of tooling does that kind of enterprise need?

[00:46:24] MW: The key thing I always try to emphasize when I'm talking with customers who do real jobs rather than this stuff I had to do is people get hung up on, "Oh no, I have to completely refactor my entire codebase to Kubernetes or whatever," the latest hotness in developer trends. Then by the time they've completed that migration, like there's another [inaudible 00:46:45] to migrating, or I need to move everybody over. I need to train everybody in Agile. They think of it in like big bangs. Don't. The whole point at DevOps is to just be continuously improving.

What is causing you the most pain right now? What's slowing you down the most in terms of delivering value in customers' hands? I'll be honest when I'm talking to a lot of enterprises. Mostly now people have kind of got – They've got source control kind of under control, which when we first started, when I'm old enough to remember when source control was just like zip files backed up, and even for a large team, working on the main frame often didn't have source control. You were just using sort of datasets in a certain file structure. That's how old I am. So we've mostly solved that problem now.

We've mostly through the 2000s figured out continuous integration and unit testing to a certain extent. So we can be continuously integrating and not constantly having to fight the merge debt, the merge pain, which was slowing teams down. We've mostly got continuous integration kind of figured out, or people are frightened of it and they don't like to touch it, because it's scary and it might break, but it kind of works.

The next stage is often around deployment, being able to automate how you get your bits from check in to actually the thing that needs to ship into the customer's hands. Then the next level of pain is getting the things into the customers' hands. So if it's continuous deployment, continuous delivery is where I'm seeing a lot of people, is their primary pain point in terms of improving the flow of value.

Again, you listen to people like us chatting away and talking about these massive web scale services and delivering to cloud and blah-blah-blah. Great, that's awesome, and you can kind of see how that can be done in a continuous delivery model, because you're delivering to one point. But I get to my Eclipse days where I was doing Eclipse updates and talking to a lot of people who were doing – Still doing Windows desktop apps, or line of business applications and things like that that don't even ship into the website. How do they do DevOps and can they do DevOps? The answer is they definitely can, but they need to make some architectural investments to make it really easy to – For their end users to pick up the latest version of their bits and get it running on their machines. So look at store applications or look at things like Squirrel, which is the sort of installer technology that's used behind the S code or Slack and all those sorts of things.

Continuous delivery and technologies, and so that is things like continuous build and release. So Azure Pipelines is our product to help you with that space, but then these things like Jenkins, and Travis and people like that in terms of doing this continuous integration, continuous build, continuous deployment pipelines. Then you've got your how do I get my thing from there into my customers' hands. So often it's installing on to a web server. Great. How am I going to make sure I could install into a web server with zero downtime and those architectural problems, or how do I get it and stored into the customers' hands? How do I publish into the app store quickly? How do I get that desktop application updating? So they are the kind of the spaces to invest to.

[SPONSOR MESSAGE]

[00:50:29] JM: This episode of Software Engineering Daily is sponsored by Datadog. Datadog integrates seamlessly with container technologies like Docker and Kubernetes so you can monitor your entire container cluster in real-time. See across all of your servers, containers, apps and services in one place with powerful visualizations, sophisticated learning, distributed tracing and APM.

Now, Datadog has application performance monitoring for Java. Start monitoring your micro-services today with a free trial. As a bonus, Datadog will send you a free t-shirt. You can get both of things by going to softwareengineeringdaily.com/datadog. That's softwareengineeringdaily.com/datadog.

Thank you, Datadog.

[INTERVIEW CONTINUED]

[00:51:24] JM: So I go to these conferences and when I'm at the conferences, I always walk around the expo hall, because one thing I like about the expo hall is you see the interaction between developers and the products that they're purchasing and also the things that they're not purchasing, and you get a sense for how that tastes of developers are evolving. What I've seen, there's conferences like Velocity, or the DevOps Enterprise Summit, where DevOps is a big focus. There are all these vendors for things like code hosting and project management boards and dashboards, and then there's the actual deployment systems and then there's monitoring. There're all these different things, and then some of the vendors have bundles of them together. You can get a variety of things all in one.

[00:52:15] MW: We would be in that category kind of thing.

[00:52:17] JM: Right. So it makes for an interesting build versus buy set of decisions, because you can build an integrated provider, or you can buy an integrated provider. You can stitch together open source tools together with random vendors. You can piece together entirely different vendors into one thing. How does a developer make these kinds of purchasing decisions, or a CIO, or a CSO? How do they figure out how much integration they want between these different systems and where should the integration points be?

Since you were heavily involved in the product architecture for Azure DevOps, you must have thought about the purchasing process that this kind of developer wants to go through.

[00:53:01] MW: Yeah, and there's always a difference between the purchasing process that the developer wants to go through and the one that the CIO wants to kind of mandate and things, which is always fun. The key thing is you need to make some bets on sort of technology trends. Equally, you don't want to be – You want to be able to hedge your bets.

For instance, moving your team to git, is it to just get itself is a great bet? Because now you have lots of vendor choices and now you're going to have a more competing with you for the best git hosting experience, and you know that there's going to be lots of innovation. Those are the sorts of things you want to be doing and identifying these key areas where I can make a bet on a technology, and you can tell I'm not a sales person, can't you though? But anyway, I'm avoiding sort of vendor lock in. I'm giving myself the most choices and the most flexibility. You do that and then you got also vendors.

The same is happening with sort of Docker to a certain extent and everyone sort of standardize on the Docker kind of file format, the Docker container format and moving towards now sort of Kubernetes is the way of managing the swarm of Docker containers, but there's plenty of competition in that space as well and you're not getting vendor locking. If you have a Docker container, you can move it to any cloud hosted provided and on-prem very, very, very easily.

Yeah, that's the way I make it, but that's kind of fit in how we've changed the product recently, because we've moved from – In the old days, it was very much a sort of suite of software that was built – It was kind purchased and mandated from the top, thou shall use. So that's why sort of we have the suite – Like a quite product from a company like Microsoft. Here's a turnkey solution. Buy this and all your problems is solved. 5,000 people in your company can use it tomorrow and it's all good.

Moving much more to a model that buy the capabilities you want, which is how a cloud is. If you think about it, developing environments naturally reflect the thing that you're developing for. When we were building desktop applications and client applications, then you have the big

IDEs, like the Eclipses and the Visual Studios and like the things before that. They naturally reflected the thing you are building.

In the cloud, you need a much more distributed heterogeneous model that needs to be spread and needs to be a lot more pluggable. That's kind of like – That's cloud native, because that's the thing that you're building for, is like that. So the way you build it is also like that.

So we very deliberately kind of broken our product into core bits of functionality that are very, very open and pluggable with other things. So it works great if you want to buy all from us, awesome, great, and we'll make that super easy and super cheap for people.

If you want to use our pipelines produce, if you want to use Azure pipelines to deploy to take code from Bitbucket and deploy it to AWS, then we should work just as well for you there, which is what we've deliberately gone and done. So we have the integrations on both ends and we work with Amazon for them to build the things that deploy bit to AWS. Yes, we have an awesome experience for deploying to Azure, but we also have to have an awesome experience for deploying to AWS and GCP, because that's how people buy these tools. They're not going to buy something if it gives in vendor lock in. So we deliberately build tools that are giving you lots of choices and trying to use open source and industry standard kind of ways of interchange as much as possible there.

[00:56:45] JM: You're describing this, like the DevOps experience that you want to have is very much like the experience you want to have in a cloud provider. So in a cloud provider, you log in and there's like 5,000 different services that the cloud provider offers, and then you can go into the marketplace and find a bunch of more services that are offered by a variety of other vendors. Like if you're on Azure, you can find Cloudera. You can easily install Cloudera using the marketplace and then – But you're saying that for DevOps, you kind of want the same thing. You want this buffet of options that you can integrate with easily and it's kind of a marketplace kind of experience, but you also have centralized, I guess, standard opinionated ways of doing things that are in accordance with how Microsoft sees DevOps.

[00:57:40] MW: I think that's what most customers want, and that certainly seems to be from when I talk to them. They don't want to be locked in, but equally they find all the choice quite overwhelming and they'd quite like some opinionated guidance as to what they should go down. Even with the cloud providers, like you're looking – Say you go and you're doing your just basic stuff, like the CDN, you can go in to Azure and get there, the Azure CDN, or you can go to Azure and buy Akamai or buy one of the other CDNs, or you can go to the other – Or you can go to AWS and buy their CDN, that CDN'ing stuff that's stored in Azure. It's all there and it should be interchangeable and plug and play.

Because the whole point is you want to iterate. You don't want to be throwing stuff away. You want to be iterating rapidly and you want to solve the thing that's most painful for you right now. If what's not painful for – You don't want to have to change things, which are working just great for you. So leave them alone until they become the thing that's slowing you down the next time and just keep incrementally improving, keep incrementally adopting. You find that these massive improvements.

We were talking about Microsoft's delivery at the start of the show and it was like every two or three years kind of thing is when stuff got shipped. We ship 7,800 times a day now. The 7,800 deployments a day come out of Azure Pipelines in Microsoft, because that's what we use for Microsoft alone. And that's just madness, but that took us 10 years to get here. Like the first time we improved, we shipped every six months rather than every three years. Then we shipped every six weeks, then every three weeks, and now multiple times per day. Just slowly make those incremental improvements. Don't feel like it's completely, "Oh no! I have to rebuild everything and start from scratch." What's the thing that's causing you most pain right now? Improve that, repeat, is my advice.

[00:59:35] JM: Just to wrap up. Yeah, that's a great piece of advice, but do you have any other pieces of advice that since you're talking to a lot of different customers – I mean, the thing that I always hear from enterprises is that they're having a lot of travel with is testing around their big ball of mud monolith, for example. That is a huge inhibitor to getting to continuous delivery, because if you can't test the big ball of mud, then you can't really continuous release it and that just becomes a big ball. That's not really any advice that I can give. But some commonality I've discovered talking to different enterprises. But what advice would you give to these enterprises who are adopting DevOps?

[01:00:13] MW: Yeah, starting a little bit is what is the main thing. But you talked about testing there. That's really an interesting one, because especially with – Again, when I had a real job, you're trying to get the business to stop doing business for a day and come and test something and then go back to doing the business again, and that's always a hard sell, because they're busy doing business and making the money that pays your wages.

So making the architectural changes possible so that you can do things like progressive exposure. You can rapidly fix what's running in production and move towards and testing and production. That's why the people are doing – That's why the people who were doing more testing and production we're finding are being more successful. That's not to say disable testing and just go deploy everything to production tomorrow, because you'll deploy some horrible mistakes. You need to get there slowly and incrementally get there. But being able to expose the proportion of user base to a thing, a new thing first, is a great baby step. Being able to compare for a proportion of your users the traffic of the old code? What results I've already got from the old code, which is what you give the end users who were in production. Compare that programmatically with the data that would have come from your new code and see if you get the answers.

But if you start investing in those kind of practices and that kind of infrastructure, then that allows you to iterate more quickly. That's why investing in unit tests was worthwhile, because it allowed you to do continuous integration, which allowed you to iterate more quickly and have confidence when you do big code refactoring. But if you haven't got the unit test and you're scared of refractor, so how do you address that and maybe being able to test code in production or test code using production traffic is a great way forward for you. Continuous deployment, continuous integration, getting things into your customers' hands as quickly as possible. Then how quickly can you speed up, ramp up your engineers? When I bring a new engineer into my company, how long does it take them to have code running in customers' hands and how happy are my engineers. The more frequently engineers ship, the happier they are. That's like proving in the data from the latest stack overflow survey. How happy are your engineers? How happy are your customers? How quickly can you get stuff into customers' hands?

[01:02:38] JM: Martin Woodward, it's been great talking to you. Thanks for coming on Software Engineering Daily.

[01:02:41] MW: Thank you very much. Good to speak to you.

[END OF INTERVIEW]

[01:02:45] JM: We are running an experiment to find out if Software Engineering Dailydaily listeners are above average engineers. At triplebyte.com/sedaily, you can take a quiz to help us gather data. I took the quiz and it covered a wide range of topics; general programming ability, a little security, a little system design. It was a nice short test to measure how my practical engineering skills have changed since I started this podcast. I will admit that though I've gotten better at talking about software engineering, I have definitely gotten worse at actually writing code and doing software engineering myself.

But if you want to check out that quiz yourself, you can help us gather data and take that quiz at triplebyte.com/sedaily. We have been running this experiment for a few weeks and I'm happy to report that Software Engineering Daily listeners are absolutely crushing it so far. Triplebyte has told me that everyone who has taken the test on average is three times more likely to be in their top bracket of quiz course.

If you're looking for a job, Triplebyte is a great place to start your search. It fast tracks you at hundreds of top tech companies. Triplebyte takes engineers seriously and does not waste their time, which is what I try to do with Software Engineering Daily myself, and I recommend checking out triplebyte.com/sedaily. That's T-R-I-P-L-E-B-Y-T-E.com/sedaily. Triplebyte, byte as in 8 bits.

Thanks to Triplebyte for being a sponsor of Software Engineering Daily. We appreciate it.

[END]