EPISODE 346

[INTRODUCTION]

[0:00:00.0] JM: Many companies are transitioning from a monolith to a microservices architecture. Tools for cloud computing and containerization and continuous delivery are making this a lot easier, but there are still technological and organizational challenges that a company will encounter while making this transition.

Cassandra Shum is an engineer with ThoughtWorks, she has worked with major financial institutions and other large companies to architect their migrations from monolith to microservices. Also, she regularly puts on workshops to engineers who are seeking to make this transition at the company that they work at.

In this episode, Cassandra describes some of her experiences and recommendations around transitioning from a monolith to microservices.

[SPONSOR MESSAGE]

[0:00:59.7] JM: For years, when I started building a new app, I would use MongoDB. Now, I use MongoDB Atlas. MongoDB Atlas is the easiest way to use MongoDB in the cloud. It's never been easier to hit the ground running. MongoDB Atlas is the only database as a service from the engineers who built MongoDB. The dashboard is simple and intuitive, but it provides all the functionality that you need. The customer service is staffed by people who can respond to your technical questions about Mongo.

With continuous back-up, VPC peering, monitoring, and security features, MongoDB Atlas gives you everything you need from MongoDB in an easy-to-use service. You could forget about needing to patch your Mongo instances and keep it up-to-date, because Atlas automatically updates its version. Check you mongodb.com/sedaily to get started with MongoDB Atlas and get $10 credit for free. Even if you're already running MongoDB in the cloud, Atlas makes migrating your deployment from another cloud service provider trivial with its live import feature.

Get started with a free three-node replica set, no credit card is required. As an inclusive offer for Software Engineering Daily listeners, use code sedaily for $10 credit when you're ready to scale up. Go to mongodb.com/sedaily to check it out. Thanks to MongoDB for being a repeat sponsor of Software Engineering Daily. It means a whole lot to us.

[INTERVIEW]

[0:02:59.9] JM: Cassie Shum is a lead developer at ThoughtWorks. Cassie, welcomes to Software Engineering Daily.

[0:03:04.1] CS: Thank you. Thank you for having me.

[0:03:06.3] JM: We've done all these shows about microservices and any listener who is curious can search for those other episodes. In case a listener is tuning in for the first time in they're totally unfamiliar with this idea of monolithic software architecture versus microservices architecture and they're not familiar with the idea transitioning between the two, can you give an overview of what's the different monolithic architecture and a microservices architecture and what is involved in a transition?

[0:03:33.9] CS: Yeah, that's a great questions. Essentially, a monolithic application is pretty much a massive application where normally there is about one repository where a lot of people are pushing in their code and in a big enterprise system these monolith systems can be very very big because they've been evolving over time. Usually these monolithic systems have probably one type of database and one tech stack and the main issue with the monolithic application is that they have to be deployed all at once. When you actually take your enterprise system and you're making a small change in a particular feature you still have to deploy the entire monolithic application in production in order for the users to get an upgrade to that particular services.

Microservices on the other hand which is why people are talking about this a lot these days is these independent individual deployable units and services. Essentially, when I was talking about the monolithic application and making that small change in a microservices application or architecture, if you're making a small change to a particular service that's really nothing to do

with any of the other services in your microservices architecture then that particular service can be deployed independently without affecting everything else.

Microservices are also quite good to have around especially when you have to pick different technologies, different databases, and each of the services have their own certain autonomy. That's sort of why a lot of people are trying to transition over to microservices architecture. That transition itself can be actually quite difficult. However, that's actually what a lot of companies are trying to do these days, like in a Netflix started it all.

Essentially, what the first step to that transition would be is to figure out what is the most critical services layer that you have that you want to be able to deploy independently from the rest of your monolith and start strangling those things out into their own microservices. That would be the first step.

[0:05:38.9] JM: Yeah. Netflix, I remember, the first thing that they strangled out or that they moved out of their monolith, I think if I recall correctly, was the job so job board, job postings on Netflix. Because they broke that out and because there is low risk involved there. If the job board goes down, it's not a huge deal because, okay, people can't apply for a job at Netflix, but that's not the end of the world. That doesn't stop people from streaming movies on Netflix.

Netflix and I think Amazon were some of the earliest people who talked about this publicly. Google may have done this before Amazon and Netflix. There were a few decades where people were talking about service oriented architecture and I think that was somewhat similar to microservices but we eventually made to today were people talk about microservices. Do you have any idea why we changed the words that we used to describe this architecture?

[0:06:40.4] CS: Yeah, I think what happened was, and this is just my opinion, is that from the SOA days we were able to call these particular services via their endpoints and all of those things and they were different components. However, I think what's happened is is that microservices is actually more thinking around like a domain-driven design type of mentality which is what SOA did not do. When I talk about domain-driven design, it's actually really thinking about the business use cases of what your domains actually are and then being able to

architect your microservices around these particular domains because that's actually what's going to determine what you can deploy or not deploy independently.

What we find out with some of the SOA architectures is that they became, again, too highly coupled and then we got to our big ball of mud again. When we're talking about microservices we really want to talk about that mentality of domain-driven design and really having these particular services that are actually independent and can operate on their own.

[0:07:47.0] JM: Some of that evolution towards smaller composable units and more narrow design I think was a technological evolution because when you had VMs where the deployable unit was a VM, that's a bigger address space and so people had bigger units of computation sitting on those VMs. Then as we move towards Docker containers it became easier to break them into even smaller units and so it really fit the domain-driven design better. At least the resource mappings became more aligned with the desired design mappings.

[0:08:35.2] CS: Yup, absolutely. I think with the rise of Docker and all of these different technologies, that's actually why we've started to move to microservices as well because things like the virtualization on demand and Docker and how that's actually become a lot easier. One of the other things that we've matured quite a bit it is the infrastructure automation as well. One of the big things around microservices is that there is a lot of overhead because now you actually have to deploy all these different microservices as opposed to the one.

For a while, that infrastructure automation was not as mature for the last few years but it has actually come up and about. Being able to automate having to push everything to production has gotten a lot easier.

[0:09:24.0] JM: Yeah, also I think — I guess cloud, you're saying something similar to cloud, the infrastructure automation. It seems like teams were spending more time dealing with outages before cloud was popular and dealing with outages and dealing with deployment issues and having to troubleshoot those yourself because — Troubleshoot them not only at a software level but at a hardware level and those outage difficulties created a lot more organizational friction than you have today where you're just kind of outsourcing the hardware difficulties to the cloud provider. If you're outsourcing that stuff you can just focus completely on the software design

and since there's more focus on the software design you can create more intelligent domain-driven designs.

[0:10:18.5] CS: Agreed, and taking that a step further I think we're going towards this area of DevOps. Being able to combine the operations piece in the developers piece together really changes the mindset of the developers and being able to put their own software into production. I think that in itself is why we actually really like going to microservices well because we have — In a monolithic world, there's a lot of complexity. There's a lot of high cognitive load.

For a particular developer or a team of developers who can focus on one particular microservices or a couple and really focus from front and on putting — Building it, testing it, and putting into production also as quickly as possible using continuous delivery. That actually really does increase the quality of the particular software that they're putting into production.

[0:11:15.1] JM: You've given a number of workshops for people who are transitioning to microservices. What's the mindset of the typical engineer who is in that audience who is thinking about working on some sort of migration to microservices?

[0:11:32.0] CS: It's a good question. Interestingly enough, a lot of the people who I've given these workshops to were people whose managers told them that they had to go to microservices so. They had to figure out what that meant. What I really really enjoyed about the workshops themselves is as being able to take these engineers and actually really show that it is not just the job of engineer to switch the entire architecture to microservices, but it really does take the whole team including thee product focus individuals, the QA, the testing pits, the DevOps. It takes the entire team to really think about and change their mindset to the microservices architecture.

Most of the people who leave the workshop will actually come back with a lot of questions with their entire team, and actually that's the better way to think about it as opposed to it should just be the architect of think about how to move things over, it actually needs to be the mindset of the entire team.

[0:12:34.5] JM: When they come to the workshop and they perhaps don't really know why their manager told them to go to this thing, are they able to quickly self-identify as somebody who has an architecture that could use a refactoring?

[0:12:47.0] CS: Yes, I think after a while, because part of the workshop that I give is really showing the drawbacks of a monolith and the pain points that we're trying to get away from. I think when I talk about the difference issues that we have like being able to test your monolith or the complexity changes that happen, again, the high cognitive overload, those types of situations I kind of go through a few examples and most of the people in the room are like, "Oh my gosh! Yes, I have those problems and I see why we want to go to microservices."

It's interesting to see how a lot of people were — It's a buzzword. Microservices is a buzzword, so all the managers and architects say, "Go and learn about them." Really, what I want people to learn about why we want to go to microservices or even go to microservices slowly is to really move away from some of the big complexities that we see with monoliths and some of these drawbacks. If that's actually hurting your business then you should definitely be thinking about it.

One thing is I don't think we should automatically just move to micro service architecture because someone said so. We have to really understand the pains and understand what our business actually needs.

[0:13:59.7] JM: When a company is making that transition, what are the steps that you advise them to take for their first microservice?

[0:14:09.4] CS: For the first microservice. Oh, yeah. I think for the first microservices, you actually alluded to this earlier with Netflix and taking something that is not that critical to the business to really test out strangling this microservice out. The reason I say this is because, A; if it doesn't go well, then you're not endangering your business and you're not hopefully going to lose that much money if it doesn't go well. B; the thing around microservices is it's always an increase in overhead of infrastructure of deployments of running your different pipeline. With that first microservices strangulation, it's going to take a lot more time and a lot more change of your organization. Taking something that's a little bit separated out from the rest of your

business in a little bit low impact will probably be the first safe thing to do. You can iron out all the knots around the operational cost of separating out that one microservice.

[0:15:09.7] JM: As you go from the first to breaking out something that is a little more mission-critical, how does the process change?

[0:15:19.0] CS: I think what I've seen before and like what we've been working with people on pulling out the second and the third microservices is consumer driven contracts. That's a one thing that I've seen more when you're driving out the different microservices because now you have more than one microservices that are actually potentially talking to each other and that overhead is a bit increased. You want to make sure that your test between those microservices are actually strong.

When you're pulling up the second and the third one, we want to make sure that your automated testing between the services, those contract tests, are strong and good and actually people are aligning with those contracts between the different microservices. That's actually the biggest thing that people don't think about when they start separating out multiple microservices.

[SPONSOR MESSAGE]

[0:16:19.7] JM: At Software Engineering Daily, we need to keep our metrics reliable. If a botnet started listening to all of our episodes and we had nothing to stop it, our statistics would be corrupted. We would have no way to know whether a listen came from a bot, or from a real user. That's why we use Encapsula to stop attackers and improve performance.

When a listener makes a request to play an episode of Software Engineering Daily, Encapsula checks that request before it reaches our servers and filters bot traffic preventing it from ever reaching us. Botnets and DDoS are not just a threat to podcasts. They can impact your application too. Encapsula can protect your API servers and your microservices from responding to unwanted requests.

To try Encapsula for yourself, got to encapsula.com/sedaily and get a month of Encapsula for free. Encapsula's API gives you control over the security and performance of your application. Whether you have a complex microservices architecture, or a WordPress site, like Software Engineering Daily.

Encapsula has a global network of over 30 data centers that optimize routing and cacher content. The same network of data centers that is filtering your content for attackers is operating as a CDN and speeding up your application.

To try Encapsula today, go to encapsula.com/sedaily and check it out. Thanks again Encapsula.

[INTERVIEW CONTINUED]

[0:18:04.6] JM: Does it get significantly easier or does it get harder as people are breaking off more of their functionality into microservice? Because I could see it easier because you get more experience, but it gets harder because the easy stuff is out of the way.

[0:18:21.6] CS: Agreed, yeah. I think it's actually where you think the difficulties are. I think from an operational cost, at this point, we have figured out, especially if you're using infrastructure as code and you're scripting out all your deployments and your pipelines and those types of things, that part is easier to really operationalize all the different microservices and follow that template.

As you said before, yes, once you now get to, in the end, like your ball of mud, the critical paths and where you want to actually strangle out, I think that takes more work on the design point of view from the code being able to use that strangler pattern and understand putting together some sort of dispatcher or something in order to pull out the different components in which you want to pull out that microservice gets very difficult. My one advice around that is don't leave it in the middle when you're actually trying to strangle it out. Make the decision on what you're actually trying to strangle it out and focus on getting that out into a microservice.

One of the things I've seen in enterprises that we've worked with was they would start a strangulation of a particular domain or a microservice and then stop in the middle because it got

too hard. That can be even more troublesome than just having your monolith, right? I rather you keep your monolith instead of doing this in between fashion.

[0:19:48.2] JM: At what point do you introduce continuous delivery?

[0:19:50.7] CS: From the beginning.

[0:19:54.4] JM: The very first microservice.

[0:19:55.5] CS: Absolutely, from the very first microservices. We kind of talked about you have to be this tall in order microservices. I always say you have to have understanding and the thought process of continuous delivery before you have microservices.

The one major reason for me at least why I really like microservices is, again, the independent deployment of your services. For me, continuous delivery is a precursor to being able to deploy these things in a safe manner. Without continuous delivery I could be deploying all of these microservices that could be breaking everything and then you're in a worse shape than you are before.

Being able to safely deploy your applications into production is actually going sto be the major precursor before microservices because you want to make sure you have your pipelines in order, your testing, your automated tests in order, making sure that everything works between the microservices as well. Being able to do that in a consistent basis will make it also easier for you to deploy many microservices on a consistent basis.

[0:21:04.2] JM: Does every company have a different delivery pipeline where they have different structures for staging and QA, or do you tend to standardize? To you tell people a standard, “Hey, here are the different stages in the pipeline that you should have.”

[0:21:21.4] CS: I think as a company or as a consultant, we do have our standard ways that we like to see our pipelines, like having a development, QA, staging, and production are pretty standard environments. However, we do go into the organizations that we work with and it depends on how they’re set up. It depends on where their bottlenecks are. It depends on where

their pain points are and that's normally where we want to focus. If that means that this company only has QA and production, then we have to work around that we have to figure out what does it actually mean as opposed to abiding by a standard template.

I think the angle is to be able to, with confidence, deploy your application of production, and that's the end goal and however way we do that, if it's use of test, if it's deploying five lines of code every five minutes, being able to roll that back. Those types of things we need to think about. At the end, if you can deploy to production with confidence, then you're already a step ahead.

[0:22:25.9] JM: As a consultant, you help companies make these migrations that take a considerable amount of work. When you are working with a company to help them move to microservices, what's the scope of that job? When you have a giant financial institution that you're helping to migrate, what are you doing and how long do you stay on? How do you make sure you stay on long enough to make sure they're there going to get a satisfactory migration?

[0:22:58.4] CS: Sure. That varies for sure. I think one of the major things that we work on as consultants is not the fact that we may stay on until the very end but that the whole company or the whole team and company and organization has an understanding of what the different steps looks like to that migration Some of these migrations can take years and will take years depending on how big these enterprise systems are.

One of the things that we really stress on is what is your business critical outputs? What are the things that are going to differentiate your company from the rest? Where should we start pulling out these microservices, if that is in fact you want to go that route? Also, how do we actually get the entire organization on board with that?

I think more of my issue as a consultant is getting everyone on board. Normally, when we get called in for these migrations, there's a few people that said, “Hey, we heard about microservices. Let's do that.” A good chunk of work in the beginning is really transforming the organization to really thinking on this level and why they should be thinking about, because in the end, if your team is not writing the tests and they're not practicing continues delivery but

they’re pulling all these microservices, that could end up a lot worse for your organization than before.

That’s the major amount of work in the beginning. Then, of course, as consultants we have teams that come in to really help teach the basics of what continues delivery actually means. We have DevOps people who come in and really teach how to do infrastructure as code and how to build out those pipelines and automate a fashion. We have a lot of people who come in who figure out how to write these automated tests with our clients and teach them how to do those things as well. There's a lot of groundwork that needs to be done before you really even think about going to microservices, in my opinion.

[0:24:57.1] JM: I was reading your LinkedIn, you have some projects listed, that's why I said financial institutions. You've worked on some financial institutions as projects and I spent about five months my first job out of school working at a financial company, and a financial institution sounds like basically the hardest type of company to update infrastructure-wise, because — Or at least an older financial institution because they have so many layers of different legacy products and integrations. Just look at the financial system and the payment system and how everything works, just layers and layers. It’s like an archaeological dig of these different stratus of a legacy integrations.

[0:25:41.8] CS: Oh boy, do I remember those? Yes.

[0:25:45.8] JM: How do you approach a job like that?

[0:25:49.3] CS: With a lot of support from your teams. It's been great going to these big infrastructure companies because we look at them and it is very overwhelming, but we go in as a team as well. I’m not the only person that goes in there and says, “Hey, this is how we’re supposed to do all of these things.”

One of the big things that we did at least for one of the big financial companies that we worked for is we went in there for a few months to do a lot of research into looking at all the dependencies, all of the different layers that you're talking about and some of them very legacy

layers, like back in the COBOL days, and in really understanding where can you draw those lines on those dependencies and actually really even thinking about what those tests look like.

ThoughtWorks is a very test-driven culture, and really the first step that we normally do when we go into some places like this is try to drive out a lot of tests in some of these massive third-party dependencies or just dependencies in general. We know when we're starting to strangle these things out, we're not breaking those basic things. I know that sounds easier said than done, it was. I know it sounds much easier, but it takes time. It takes time to really understand these big legacy systems. I think that's part of the fun as well.

[0:27:07.4] JM: The testing question — I've done several shows with people from the DevOps community, like Gene Kim, and this is a question that I always ask these people; you take a big company like an insurance company that's been around for years and years and years and they don't have a lot of automated tests, and they want to move to DevOps. In order to get to a place of rapid experimentation, you often have to be able to have a lot of tests around your software so that if you make an update you can quickly test it against a battery of automated tests.

One of the challenges of the systems is when you get to a certain scope where there's just so much code that's untested it becomes like a massive undertaking to say, "We're going to add significant automated tests to this sort of thing." Then like an insurance company is pretty determinist. If you take a financial services company, like a trading company, the trading systems have a lot of behavior that is essentially nondeterministic where it's like, "Okay. It's really hard to think about all the different situations for how a trade could occur."

I've heard a lot of places that just basically do a lot of manual testing for these trading systems or other systems that have so much complexity that you can't test everything. When you're looking at building tests around a system as the first inroad into getting a complex ball of mud into something that's more maneuverable, something closer to a microservices architecture, how do you approach these insurmountable architectures where building in automated test is looking like a three or six month challenge if you want to do so comprehensively?

[0:29:01.3] CS: Yeah, that's a great question, and that's usually the question I get when I go into these these big companies. I like the fact that you said big ball of mud, right? We essentially will

start off with doing some sort of high-level analytics on the codebase and actually look at where the big ball of muds are. Where are those god classes are? Where are those places that are, A; the most critical from a business function, and, B; the most complex in terms of the codebase.

When we are talking about with a business, what do we strangle out first, or how do we actually start this process? My first device is not to say, “Test everything.” Absolutely not, you’re right. That’s going to take a very long time to automate that process.

What I would say is actually have some very high level functional tests to run against what the critical business functions are. Making sure those things are not broken. Looking at what we’re going to strangle out first and take those high business functional areas and, yeah, run those high level functional tests. And then strangle that out as we are writing those tests. It's not something that we’re just going to cover everything in the world and then we'll start. It's definitely — In the end of it, it’s very — For me, it's a very simple question, is, “Okay. What don't you want to break?” So we can keep continuing on this company. When you ask those questions and they give you a list of answers, those answers are what you’re going to write your tests on, and that will be the first pass.

[0:30:38.8] JM: Yeah, you can imagine — You described these higher-level functionality tests and this is in contrast to lower-level unit tests where you could imagine scripting a selenium test that goes through the online registration workflow, just like from a web browser, where you’re scripting a web browser doing that functionality. You can always very easily run that test and make sure that it works as opposed to testing the lower level code and validating at the database level or at the end memory level. If just validate the browser level, it does give you a high degree of security around things. It’s not perfect, but it’s a start.

[0:31:24.9] CS: Yeah, and as I said before, it's a start. It's not everything that if I just have those tests and I'm good to go, absolutely not. I think it's just a start to really focus everyone on, “Okay. What can we not break in the system for us to continue?”

When you're strangling out a microservice, what you’re really doing is actually figuring out those high level functional tests that you need to maintain and make sure that they're still working. As you're strangling out to your microservice, as you're strangling it, you are writing unit tests as

you're rewriting that code into that microservice. That's like the two-way street that's coming. You want to make sure that as you're strangling it out, as you're pulling out different code, you're going to probably be unit testing those as you're going along as you're strangling them. At the same time, understanding that you're not breaking anything from a high level functional point of view because those tests will constantly be running in your pipeline and you just want to make sure that that's still working as it should.

[0:32:25.9] JM: When you work with these companies, do they halt feature development completely to do this migration to microservices?

[0:32:32.6] CS: No. Sometimes I would like them to, but no. Feature development is very important especially for, in the end, making money. That's actually why when we're strangling out microservices, the first one as we sort of discussed before, is, "Let's go ahead and strangle out that low impact microservice so we can concentrate on sort of the operational concerns as everybody else is probably still continuing on with feature development."

Now, the one thing I will say is that if we're coming in to strangle out this big ball of mud we, want to ensure that the new features that people are putting into their monoliths are, A; somewhat componentized from the monolith itself, and, B; tested as they're actually building out those new features. Those can be easier easily strangled out when the time comes for those as well.

[0:33:24.6] JM: How does a transition to microservices differ if a company wants to remain on prim? Have you had anybody like that, or do you typically — The people you work with, they want to go to the cloud if they're already on the cloud.

[0:33:38.3] CS: Yeah, from my experience, they've typically been on the cloud or want to go to the cloud anyways, and we've already had some DevOps infrastructure people, or from our company have worked with them already.

I sort of come in from more of the microservices and also the front end portion of the architecture because that's sort of where I like. I pretty much look at what do the actual organizational teams look like as well, how to actually take the organization and prep them for

microservices. Had to actually look at the business point of view and figure out what we need to drive out from a microservices point of view, and those types of things, so yeah.

[SPONSOR MESSAGE]

[0:34:25.7] JM: Are you ready to build a stunning new website? With Wix.com, you can easily create a professional online presence for you and your clients. It's easy. Choose from hundreds of beautiful designer-made templates. Use the drag and drop editor to customize anything and everything. Add your text, images, videos and more.

Wix makes it easy to get your stunning website looking exactly the way that you want. Plus, your site is mobile optimized so you'll look amazing on any device. Whatever you need a website for, Wix has you covered. The possibilities are endless, so showcase your talents. Start that dev blog detailing your latest projects. Grow your business and network with Wix apps that are designed to work seamlessly with your site, or simply explore and share new ideas. You decide.

Over 100 million people choose Wix to create their website. What are you waiting for? Make yours happen today. It's easy and free. Just go to Wix.com, that's wix.com and create your stunning website today.

[INTERVIEW CONTINUED]

[0:35:47.8] JM: Companies that are moving towards a microservices architecture or a DevOps mentality, oftentimes they have a lot of manual processes that are maybe written down, maybe they're not written down, they might be the tribal knowledge within the company. The newer mentality is to encourage people to go towards automation and really focus on automating those things that have become manual processes. Should a company standardize on an automation tool or a workflow of some sort for building in the automation workflows?

[0:36:21.8] CS: That's a great question. I think it's easier if a company standardizes on, say, the tool for the pipelines and how you automate your pipeline. If you have Jenkins and GoCD and Snap CI and all of these different places to actually run your pipelines, I think that's bad. I think

we should standardize for the teams on the different platforms in which you're actually running these pipelines and those types of things.

In terms of like automated tests, I actually think there should be some autonomy especially if we're going to the microservices route, and this is not to say pick any language you want just because you feel like it, but there is a certain autonomy that we want from these microservices. Say, example, we have a microservices that wants to use closure as supposed to the standard Java infrastructure that the monolith has. Your automated testing tools will be a bit different in that particular microservice.

I think to have that autonomy of choosing your testing tools I think needs to be there a little bit but there needs to be some standardization on how these tests are run. I think that's more the differentiation I want to make between them.

[0:37:36.1] JM: It almost doesn't matter if these scripts are batch scripts or puppet scripts or selenium scripts that are running as you're doing a deployment. As long as you can look into the deployment tool in your deployment pipeline and understand what is happening every time a deployment occurs.

[0:38:00.8] CS: Yeah, absolutely, because I think — And I've alluded to this before, but your team — And if they have that certain autonomy and being able to follow their particular code that they develop and then they've tested and using the choices of their automated testing tool and then are able to deploy it on a standardized deployment platform, but being able to see their service go into production. That in itself actually reinforces that team to write better code, to write better tests and to have some investment in that in that service going to production.

I think that's actually why we like this way of working better because in the olden days, olden days, teams are separated by throwing things over the wall. Your software developers are the ones who write some code. They throw it over the wall and then the testers say, "Okay. Is this going to work, or this is not going to work?" If it does, they throw that over to the walls to the operations people and then the operations people are like, "Oh gosh, now we have to push out to production."

By the time that piece of code has gotten to production, the developers have already forgotten about it, and so that investment piece is not quite there. That's why having these small microservices and being able to own the entire pipeline and have that autonomy within that team I think is very very important for good quality software.

[0:39:29.0] JM: When a company builds a lot of different services, oftentimes you'll have a scenario where if serves A and then services B and C are both consuming A and the service owner of service B might say to service owner A, "Hey, I actually need this additional piece of data from your service." Can you just make a change a real and make it so that your API hands me the data from service A? That's fine, except that it also impacts the owner of service C, because they're also consuming service A.

One way to handle this issue where you have multiple downstream services that are consuming the same upstream service is versioning. Service owner B who needs a new piece of data from service owner A might say, "Hey, can you just make a new version and deploy that separately and I'll start consuming that," and then you won't impact service owner C. Of ourse, this leads to further complexity. How should a company handle the versioning of services?

[0:40:42.6] CS: Yeah, this is a great question that I get all the time, and so it's essentially, people are used to the versioning that they have from monoliths. You have your massive API, your big service, and then people are calling the different endpoints and then your versioning those.

A few things that I would talk through is, again, from earlier, is consumer-driven contracts is probably the one thing I definitely push especially now that you have a microservices architecture. There's so many more dependencies and so many more changes between your providers and your consumers. A provider could be a service and your consumers could be a service as well but they're all now intertwined.

As you said before, some consumer may need a change that a provider needs to provide. That's where when if you think about consumer-driven contracts, that's where this could actually really help the organization because if I am a provider of an API and I'm actually providing an endpoint. Say, for example, I'm providing an endpoint that gives some user information. For

consumer A, that consumer only wants like the name and the phone number. Consumer B wants the ID and the phone number and then consumer C wants all of the above.

If consumer A says, “You know what? Let’s go ahead and split out the name to first name and last name and that's how I want my provider to give it.” Instead of versioning that, what we could do is make sure that each consumer; consumer A, B and C, have consumer contracts that they are saying, “I expect this in this format.” The best part about that is the providers are actually running those consumer-driven contracts in their own pipelines.

Anytime they change their API, they must run all of their consumer contracts to make sure they're not breaking anybody else. Really, when consumer A says, “Change this,” and the provider has changed it, what going to happen is that they change it and then all of the other consumers are broken in their own pipeline before they hit production.

What I like about that is that encourages communication between the different teams and that should open out discussion pieces around, “Okay. Is this what your API really wants and is this what we really need to do? Then let's have that conversation.” Being able to catch that earlier on would really really helped in aligning what those API should look like.

[0:43:16.6] JM: What kinds of failures can occur when a company is transition to microservices?

[0:43:21.2] CS: Yeah. I think a big failure here that I've seen before is trying to break up your monolith into many many different microservices at one time. I've seen companies say, “Okay. We need to move everything to microservices,” and then they do but they don't put in the work for the operational overhead.

Now, they're just a lot of different little services that they're trying to maintain and trying to figure out and when one changes it breaks to the other and that's a big failure that can happen. When we say transition to microservices, I personally say do it one at a time. Do it very slowly and making sure that operationally and from a continuous delivery point of view you’re ready before you actually pull out the rest of it.

[0:44:12.8] JM: What are some other anti-patterns that people can avoid during a microservices transition?

[0:44:18.5] CS: Yeah. I think some of the anti-patterns that I've seen during transition is some of the benefits of microservices give you is that autonomy that we talked about earlier. Some anti-patterns I've seen is they go a little bit too overboard with that autonomy and they say, “Oh, this microservices will be in Scala, and this one will be in Closure, and this one will be in Java, and this one will be in Ruby.”

Really, what I wonder is we should be picking the language based on a very very good reason and if the reason it's just because I like the language, I don't think that's good enough. I think when you have that certain economy, there needs to still be some sort of standardization on some level. There needs to be some ability for the communication between microservices to be standardized and also like a really good reason why you want to switch to a different language or to a different database or something like that, because the more you choose the more complex it is for people to understand different microservices.

What we do like in different teams is people being able to move from one team to another to really learn about different services stuff and not get stuck in one. Yeah, those are some of the anti-patterns I have seen, again, all in moderation.

[0:45:36.2] JM: Yeah. Just to close off, can you tell me something unexpected that has happened when you were helping a company move to microservices. What are some of the surprises that you've seen during these large-scale re-architectures?

[0:45:52.3] CS: I think maybe they're not so much surprises anymore, but I think people think they’re mature enough to go to microservices when they're really not. It’s not so much a surprise. It's just more of a, “Okay. Let's talk about that before we jump into this hot water here.” I think one of the things that I would urge companies and enterprises to really think about is the confidence that they have that they're pushing their code to production, and a lot of that has to do around the testing maturity of the organization. I find that, again, with a microservices “buzzword”, people say, “Yup, that's my silver bullet and that's what I want to do.”

Really, it's not a silver bullet. It has a lot of work to do in order to get that. What I do get a little bit surprised about is sometimes people not understanding the benefits of microservices as well versus the amount of work it's going to take to get there. Before people even think about going to microservices, I really urge that company or that business to really think about what are the cost benefit analysis around it. Is it worth taking all these time to figure out operational costs to you being able to manage and deploying many many different services now as opposed to one, and really thinking about that, because if a business doesn't really need to deploy things independently and they're actually okay with their monolith, I say take your time.

Make sure that you're fully tested and ready to go and you can actually componentize your monolith without having to move to microservices yet. You don't have to jump into that boat just yet if you don't need it. My big thing is if you don't need it right away then don't you don't give yourself more work to do. That's maybe the biggest surprise is people kind of jump into it without really thinking about how much is that actually going to benefit them and how much cost is going to take to move to microservices.

[0:47:49.0] JM: All right. Let's hope that the listeners head those warnings.

[0:47:54.3] CS: We'll see.

[0:47:56.2] JM: Thanks for coming on the show Cassie, it's been a great conversation.

[0:47:58.7] CS: Thank you so much.

[END OF INTERVIEW]

[0:48:03.9] JM: Thanks to Symphono for sponsoring Software Engineering Daily. Symphono is a custom engineering shop where senior engineers tackle big tech challenges while learning from each other. Check it out at symphono.com/sedaily. That's symphono.com/sedaily.

Thanks again to Symphono for being a sponsor of Software Engineering Daily for almost a year now. Your continued support allows us to deliver this content to the listeners on a regular basis.

[END]