# EPISODE 481

[INTRODUCTION]

**[0:00:00.6] JM:** Duolingo is a language learning platform with over 200 million users. I've personally used it to learn some Spanish. On a daily basis, millions of users receive customized language lessons targeted specifically for that user, these lessons are generated by a system called the session generator. Andre Kenji Hori is a senior engineer at Duolingo and he wrote about the process of rewriting the session generator moving from Python to Scala and changing the architecture at the same time.

In today's episode, guest host Adam Bell talks to Andre about the reasons for the rewrite and what drove them to move to Scala as well as the experience of moving from one technology stack to another.

[SPONSOR MESSAGE]

**[0:00:00.6] JM:** Cloudflare runs 10% of the internet, boosting the performance and security of millions of websites. Many of you probably already use Cloudflare on your sites but we're not talking about using Cloudflare today. We're talking about building on top of it. If you're a developer, you can build apps which can be installed by the millions of sites which rely on Cloudflare. You can even sell your apps.

They can make you money every month. Your users can log in or register to your service inside your app, they can get a real time preview of your tool live on their site and they can start paying you monthly all from within Cloudflare apps. They can go from never having ever heard of you or your service to having it installed on their site and paying you in seconds.

Visit cloudflare.com/sedaily to watch how you can build and deploy an app in less than three minutes. That's cloudflare.com/sedaily, thank you to Cloudflare for being a new sponsor of Software Engineering Daily.

[INTERVIEW]

**[0:02:07.8] AB:** Andre Kenji Jorge is a senior software engineer at Duolingo. Andre, welcome to software engineering daily.

**[0:02:16.7]AKJ:** Hello, welcome for inviting me.

**[0:02:19.2] AB:** Yeah. You have a great sort of case study about the rewrite works that's been done at Duolingo but before we get into that, I was wondering if you could explain what Duolingo is?

**[0:02:30.6] AKJ:** Yes, Duolingo is a language learning app, we have around 200 million users as of today and in the app, well people can learn how to learn any language by doing exercises and translating sentences from their language to their native language they want to learn and then we have other exercises that practice listening skills and some reading skills, some grammar skills.

**[0:03:02.1] AB:** It tries to make it sort of a game to learn a new language, is that correct?

**[0:03:06.2] AKJ:** Yes, that's correct, we have all of the gamification mechanisms such as hearts, levels, experience, all of that you make the language learning experience more engaging because when you're learning a language, that's the sort of process that takes people like several years to have become fluent in a language, right?

By gamifying it, we make it easier for our users to keep learning and keep engaged so that it can achieve this very long term goal.

**[0:03:37.4] AB:** How big is the engineering team at Duolingo?

**[0:03:41.0] AKJ:** Engineering team should be around 40 to 50 people maybe?

**[0:03:46.1] AB:** How is that organized?

**[0:03:48.3] AKJ:** We have several teams, we have some product teams which we call product teams for example, we have the learning team which is the team that focuses on improving our learning metrics and improving the learning experience to users, we have for example our growth team which is concerned with expanding our user base and keeping people engaged and we also have some more foundational teams.

For example, Q&A, our operations.

**[0:04:18.8] AB:** Which team are you on?

**[0:04:20.1] AKJ:** I'm in a team called the architecture team. It's basically guaranteeing that we have a healthy code base, a healthy two stem in terms of the application level.

**[0:04:35.7] AB:** Would your tasks be around sort of setting standards around what languages are used or what frameworks or things like that?

**[0:04:42.9] AKJ:** That also. Language is used, also, maintaining like common frameworks for libraries for the rest of the company to use. Making sure that the architecture as a whole is healthy and each micro service is doing correct things and not abusing it, the structure.

**[0:05:04.0] AB:** You were involved in a rewrite of the session generator, could you explain what the session generator is?

**[0:05:09.6] AKJ:** Yes, absolutely. Well, when a users is learning Duolingo, we give users exercises and we have a pool of exercises which is pretty large. Actually, let me take two steps back, we have a product called the incubator and in that product, we have volunteers making the courses that you see, it's like a crowd source approach and you know, the volunteers, we will input all of the data and we'll make the core structure and write all of the course content.

Then, the session generator is responsible for pulling all of the data and picking the exercises that will be the most interesting for the user at that point and give in a session that is bite sized, small for the user to do whatever they are.

**[0:06:18.8] AB:** A session is like a unit of learning and the session generator creates that unit?

**[0:06:25.2] AKJ:** Yes, that's correct, the session is, well, simply speaking is a collection of exercises.

**[0:06:31.7] AB:** Okay, is this session generated on the fly or is it created ahead of time?

**[0:06:37.5] AKJ:** Part of it is on the fly, we have a lot of preprocessing going on because there's so much data involved in each session that if we do everything, goes on the fly then that's a very time consuming.

**[0:06:51.8] AB:** Why couldn't all the Spanish sessions be created ahead of time, is there –

**[0:06:59.8] AKJ:** Yes, we also want the sessions to be adaptive for each user, right? For example, let's a user knows 10 words that and some user knows 50 words. The sessions for these two users will be different and also we take into consideration, how likely they are to remember a word and other things like that.

It's very important for these sessions to have this online component.

**[0:07:31.1] AB:** How do you figure out when somebody's going to remember a word?

**[0:07:35.7] AKJ:** We have a model for that, there is this thing called the forgetting curve for a word and it's basically the probability that a user will forget or will remember a word after a period of time, after they've learned, let's say that one day later, they have ability of remembering that X, two days later, they probably will decrease.

We modeled that as in a curve and then we do some sort of regression to estimate how well duals know each of the words.

**[0:08:10.2] AB:** If I learn 'garçon' is man today then you figure out based on certain probability, I probably know it tomorrow but not next week and then you reintroduce it?

**[0:08:24.4] AKJ:** Yes, that's correct.

**[0:08:27.5] AB:** Why did you decide that a rewrite was required. You have the session generator, it takes into account these kind of forgetting curves and builds a lesson on the fly I guess? Why did you decide you needed to rewrite it?

**[0:08:42.6] AKJ:** Yes, it all starts with the monolith I guess and all of the nightmarish stuff started a monolith. We have the monolith and it's written, Python, and the thing is that in the first years of a startup, we are all thinking of let's move fast, let's ship quickly. After some years, we end up adding a lot of dependencies like data stores and other services and what not.

In the end, the entire system becomes very – the performance is not so great anymore and also, the site's not as stable and that's – one of the reasons is to re-architect the whole system so that the system's more robust and the second reason is that we have the coding Python and, well Python is a great language for writing things quickly and having like prototypes ready very fast.

But for systems that are very large, they have like very complex data structures and complex algorithms, then Python is not so great because you know, let's give it an example. Something as simple as dynamic typing becomes a nightmare because then you don't have all of the niceties that you have in a strongly typed language because the compiler won't do as many checks for example.

Then the developers lose confidence in writing code and then they have to spend a lot of time testing and testing to see if all of the possible corner cases are being caught and nothing's going to break my production.

**[0:10:27.6] AB:** Let's see if I understand it. We have this giant Python monolith so as you add new features, it keeps on growing. You're saying, the problem with Python is when you want to add something new like your level of confidence is low because of dynamic typing, do you have an example maybe to explain that?

**[0:10:47.3] AKJ:** An example, sure, okay. Okay, I'll give a very simple example which is let's say we want to rename a function and I mean, you could grab your entire code base and start changing stuff, right? But it's also possible that in some part of the code, you pass that function as a [inaudible] and it has a different name somewhere and then your code breaks into production.

Or for example if you want to add another argument to a function then you have to guarantee that also nothing is going to break but then it's also very hard to find all of the occurrences or that for example, even simple things as you don't know what data type you're getting into function.

You can assume you're getting something but in the end you're getting something else. These are all things that a simple compiler check could figure out and say that you're doing it wrong but since Python is – does not have that, then it's very easy to break stuff.

**[0:11:55.1] AB:** Did unit tests help with this process at all or were there tests?

**[0:12:00.8] AKJ:** We do, for part of the – so unit tests definitely help but in the starting years of a startup, I guess people are more concerned about shifting things than writing unit tests. It gets just data, even parts of our code base are very difficult to mock with a unit test.

Part of the monolith because after while we, we learned from our mistakes in real life that we can architect things in a different way that makes tests easier but in the monolith there is to many places where it's very hard to write unit tests for how to mock things.

**[0:12:41.0] AB:** Especially if you didn't have unit testing in mind, sometimes you make decisions that make it hard to add them after the fact?

**[0:12:47.2] AKJ:** Yes, that's very true.

**[0:12:50.1] AB:** When you decided to do this rewrite, what made you choose Scala?

**[0:12:54.0] AKJ:** There are some reasons, the first is that our infrastructure is built on top of AWS so yeah, we needed to think of the languages that are natively supported by [inaudible] as of right now. Well Python, java script, NOJS, JCM based languages such as Java, Closure and Scala and I think GO with also one of the supported languages.

Well, Java script has the same problem, as far as weekly typed, this is something that we wanted to avoid for the particular case of the session generator and then Java is well known and it's a bit slow and verbose, it's low to develop and it's very verbose.

We wanted a more modern programming language which is why we thought that Scala might be a good choice and also because Scala is also very mature in the back end and it's used by many applications in the big data domain.

Big data is kind of similar, well, we're dealing with the session generator, right? Because it's like, there's also like a lot of data, complex data structures, complex algorithms and it seemed to be a very good fit.

**[0:14:17.0] AB:** Was there any concerns about the learning curve of a language like Scala? I mean, it has a reputation for being somewhat complex.

**[0:14:25.5] AKJ:** Yeah, we had concerns, right now, since our entire engineering team is small and the number of people dealing with Scala is also small, my population of people who have learned Scala here in the company is also very small so I don't know how we can talk about like how easy it is.

But up until now, I personally thought it would be harder for me and for other engineers to learn Scala and to get going but it was a lot easier than we anticipated.

**[0:15:00.4] AB:** let's discuss some of the features of Scala that you found useful in your rewrite. Could you describe a referential transparency?

**[0:15:13.4] AKJ:** When we have referential transparency, we have a method that the only thing it does is calculate the output and doesn't change any state anywhere so it's very easy to unit test and to also to just logically debug what's going on because you know that once you have that – once you have some input, the output will always be that one, always be what you're expecting.

It makes things very easy to test, to reason about and that's one thing that's very good once you have like very complex data structures.

[SPONSOR MESSAGE]

**[0:00:00.6] JM:** The octopus, a sea creature known for its intelligence and flexibility. Octopus Deploy, a friendly deployment automation tool for deploying applications like.net apps, java apps and more.

Ask any developer and they'll tell you that it's never fun pushing code at five PM on a Friday and then crossing your fingers, hoping for the best. We've all been there, we've all done that.

That's where Octopus Deploy comes into the picture. Octopus Deploy is a friendly deployment automation tool. Taking over where your build or CI server ends. Use Octopus to promote releases on prim or to the cloud.

Octopus integrates with your existing build pipeline. TFS and VSDS. Bamboo, Team City and Jenkins. It integrates with AWS, Azure and on Pram Environments. You can reliably and repeatedly deploy your .net and java apps and more.

If you can package it, octopus can deploy it. It's quick and easy to install and you can just go to octopus.com to trial octopus free for 45 days. That's octopus.com.

[INTERVIEW CONTINUED]

**[0:17:32.1] AB:** An example, it's helpful for me and probably the listeners if I can tie it, tie it back to an example. Referential transparency means you have a function with some amount of inputs so let's say, like the words that I know in Spanish and then the output is – do you have an example of where that could be used?

**[0:17:55.3] AKJ:** Yeah, sure, let me think. Okay, let's say some part of the algorithm, we are thinking if we should choose an exercise A or an exercise B and we put both into the function as I mentioned the function. What happens before referential transparency is that first start, they're not going to be any side effects so we want write things to the database for example because that will be adding a side effect.

Or, we won't be changing something inside one of the objects because that will also – that doesn't contribute to the output, right? The idea is that we will just do some calculation and that calculation would be used only for the output.

Let's say, if we wanted to – between two exercises, well, the output is simple, it's one of them but we can guarantee that nothing else is going on.

**[0:18:59.3] AB:** Which is what makes it easy to reason about.

**[0:19:01.9] AKJ:** Yes, correct.

**[0:19:03.4] AB:** You also mentioned, I think it ties in to that example about immutability. How does immutability tie into this?

**[0:19:10.4] AKJ:** Well, immutability, if you think that your data – when data is immutable, you have some nice properties like you don't need to think about – you don't need to think if something is changing your data from somewhere else. I think many people have seen your caller function and then you pass an object to a function, that function calls and functional calls another function and somewhere along the way, someone changes one of the properties and then you don't know who changed it.

What it changed to and you just need to add print statements all over the place to figure out what is going on because something changed part of your data and you don't know why. When you have immutability, you're guaranteed that, well, things won't change so all of your – in each step of your algorithm, the state will be very clear to you.

What this means in the context of Scala for example is that you in each step of your algorithm, what happens is only data changes. Let's say you have a list and you change the elements of a list by adding one to just some property and then you know that in your next step that your new list will have all of the elements with plus one and you know that you're guaranteed that that's going to be – If you use that variable for something else, that value will always stay the same.

**[0:20:49.6] AB:** You did say the list were adding an element to it. How can something be immutable or when we are adding to it?

**[0:20:56.7] AKJ:** Yes, that's a good question. What we do is we actually copy the list, we have the original list, we have the new list with plus one in all the elements. Then we keep just transforming data as we go and that's a way that functional programming happens is that you will always generate new stuff but you won't be changing what you had because if you change what you have then it becomes hard to understand what's going on once you have a giant system.

**[0:21:30.5] AB:** This is like Scala's immutable collections and you're saying the immutable control structures means, when you write your session generator, nothing's changed, if a number of lessons come in to the input and we need to select one then that one comes out the others side?

**[0:21:52.4] AKJ:** Yes, another example would be say, if you pass for example a – you have a set of exercises and in one part of the algorithm, you want your filter and consider let's say half of these exercises because they are better in that context. You filter your pull to half and then you do whatever you have to process.

Then, in your next step, you wanted to use the actual full set of exercises and the data is there because you have not touched it, you created a new smaller set in the previous step which if you want, you can use it or not but the idea is that your variables always stay the same, if you want that entire pull for the next step, you're guaranteed that nobody changed it, nobody removed things without you realizing.

**[0:22:48.1] AB:** To make this rewrite work, it's quite a different model of operations, did you have to change the architecture of the session generator so that it can work more as a transforming inputs rather than mutating them?

**[0:23:02.8] AKJ:** Yeah, we did change, so, many of the algorithms we had to just rewrite them in an immutable way, sometimes when the algorithm was way too hard to rewrite and it was like re-skit or introduce errors those cases, Scala has this nice thing that it's –

I don't know if it's nice or not, people might disagree but – you can write – it's almost – you can port Pava code through Scala and just run java libraries in Scala because all of them – both of them run on top of JVM. What this means is that Scala also supports things from the java world. Like mutable types and some four loops, Y loops that are not very functional, this is functional programming.

If you are rewriting something and then you realize this will be very hard to rewrite without adding complexity or making risky changes then you can write a more java like idiom of Scala.

**[0:24:18.9] AB:** What percentage would you say is more of a Java written in Scala and what percentages did you kind of go with this functional transformation style?

**[0:24:28.4] AKJ:** In our code base, I'd say that more than 99% is functional because we try to do things in a more – when you're using immutable collections and we're using reflection, things are much easier to debug, much easier to maintain.

We try to make things immutable and referential transparent and you know, functional in general in most of our code base. Most of our code bases maybe. One or two operates and we thought it will be better to just use the nonfunctional version.

**[0:25:05.7] AB:** While 99% is quite – it's very functional.

**[0:25:09.4] AKJ:** Yes. When you're writing, equal base from scratch, you can also do unit test that you couldn't do in your monolith. Things are much easier to just test that your algorithms are working as expected.

**[0:25:26.1] AB:** How was unit testing in Scala?

**[0:25:29.4] AKJ:** In Scala or should I say, like in our whole framework, we use Sinatra as our ATP server which is Sinatra is the HTP server written by Twitter. They use Juice which is the Google library and so unit testing for – in this context, we have like all dependence injection on the box, we have mocking out of the box and it's very easy to do everything.

I'd say that it was just easy to write unit that saying, we end up writing a lot of them and I think our coverage right now is somewhere along 70% maybe?

**[0:26:10.6] AB:** Nice. Back to the architecture, is the session generator, you mentioned micro services, is it something that calls out to a bunch of services and combines them together or how does it interact with the rest of Duolingo's infrastructure?

**[0:26:30.1] AKJ:** We do have a lot of – we have to pull data from a lot of places and we have a bit of pipeline for that and it's – a pipeline is, right now it's just a task that runs offline so that we don't make real online traffic depend on our data stores, right? We have this task that runs daily and hits all of the services and what not, we need to hit in our fetched data.

Then this task through processes everything and then realizes all of the data into all of the descript process data into S3 which is a file server by Amazon and the best. Then, when we are in, online, we're serving actual request for users, what we do is we just fetch from the file server and cache it in memory and serve it and when we do that, we get – we have a system that's very robust to failures because the only real dependency is your file server and network of course.

Also, it's also very fast because everything is cached in memory.

**[0:27:49.1] AB:** The only real external dependency you have is S3 and then even that is kind of insulated by your cache?

**[0:27:54.8] AKJ:** Yes. With all this functional idiomatic code you're writing, does that mean the session generator is sort of like, it takes as an input like a user and then all of the possible lessons ever and then it spits out like what they should learn next?

**[0:28:11.6] AB:** A little bit.

**[0:28:13.7] AKJ:** It actually takes as input, the online part of the session generator takes as input, the less on the user ones to learn and some, what our user setting and outputs the session to the user so the collection exercises.

**[0:28:28.3] AB:** Okay, there's an idea that statically typed languages are more verbose and dynamic languages are more succinct. I actually found in my experience that Scala is a very succinct language, maybe even more so than Python in some cases, what did you find in terms of verbosity moving from one language to the other?

**[0:28:50.5] AKJ:** Yeah. I think robustly depends a lot of the language itself and so if you're familiar with Python as your – Python or java script as your go to dynamically typed language and Java is your aesthetically typed language then I would agree that yes because Java is super verbose but the we have Scala is a language that is concerned about a lot of typing. Scala tries to infer types whenever you can. Sometimes it can't and make some errors here and there but usually it is able to infer your types, sometimes infer other things that you de-compiler can infer.

It makes like Python you can't do a list comprehension and write things, write one liners instead of write like in Java you need to write three up to five lines of code just to do a far loop. So yes, Scala is very succinct and there is not as much typing as a language like Java and compared to Python I would say I don't see yet. In some cases Scala is less verbose for example when you are defining a class you don't actually need to write. Basically if you don't need a body you don't need to write a body for Scala.

**[0:30:16.9] AB:** Are you using implicits within your code base?

**[0:30:21.4] AKJ:** We are using – so in Scala there are two kinds of implicits, the implicit parameters and the implicit conversions and so the implicit parameters is when you're basically define what some of the parameters of your method as implicit and then as long as you have – we think the scope of the color a variable of that type, that is also implicit variable. It is declared as implicit then you are able to not write the culture pass that value into the function. So it's basically to avoid typing.

So for those we do use implicits. We don't use implicit conversions because we usually find that a little bit scary because you won't see where things are being converted. So generally no implicit conversions but we do use implicit parameters.

**[0:31:20.0] AB:** And now what's an implicit conversion?

**[0:31:23.2] AKJ:** Yeah, so an implicit conversion is when you have an object of type A and then let's say that you want to convert it to an object of type B. There is a mechanism in Scala that you can define your conversion from type A to type B and if you put that conversion in your scope, you are able to convert it from A to B without writing code to convert things.

**[0:31:55.6] AB:** So it could be very handy and save on typing but also maybe you don't know what is changing to what?

**[0:32:01.0] AKJ:** Yes.

**[0:32:01.7] AB:** In my experience, Scala code can be less prone to runtime bugs. I think you mentioned you had some issues with runtime bugs getting to production in Python. So how did this changed now that you have rewritten?

**[0:32:16.9] AKJ:** Yeah, so in Scala we do have a lot less runtime bugs. Part of it is because your compiler will just get most of the errors. So as long as compilation passes, you pretty much have – well you don't have like programming errors. You have application logic bugs here and there but that is another problem right? So the compiler does a lot of stuff for you and also the unit testing framework is also very user friendly so we can write a lot of test that make sure that your code doesn't have the most common application logic errors that you'll know about. So in the end it's very hard to have these runtime bugs going on Scala.

**[0:33:07.7] AB:** What do you find to be the pain points of moving to Scala as a language?

**[0:33:12.7] AKJ:** Pain points that is a good question. So I was actually more fascinated that Scala as a modern language has so many nice things that we don't have in Python and Java. The kind of uplinked pain points, there are some pain points but I would say that they are not actually like the language itself. There are some small things here and there in the language but those are – it is not a big deal. Most of our pain points were, let's say we have a library in Scala that is under documented.

Or we have a function that is available in Python by default or anyone of the common libraries. It is not in Scala. So there are some small things but I guess that is true whenever you change a code base from one language to the other.

**[0:34:04.8] AB:** Nice, you mentioned there is some things about the language that fascinated you, such as?

**[0:34:10.3] AKJ:** Okay, let me see what are these things, so I think one of the things is malfunctioning function of programming in general and how easy it makes to – how readable it makes your code because and also all of the – since the languages are not too verbal, the end result is that your code is very explicit on your application logic instead of having like a boiler plate of just controlling your loops or things like that. So I find that coding in Scala very readable and that's nice.

It is also very easy to just look at the code and see if there is a problem because you know it is hardly readable and it's just easy to debug even without running any code and also I originally started my life as a programmer in Java and then you know, when you're in Java and you move to Python the first thing you think to yourself is that, "Oh this is a lot lesser balls" it is much faster to write stuff and then there is also the difference like Scala it is very fast to write code.

So in the end, I spend my time not only writing code but all of the extra time that I would either spend just typing in Java or testing things in Python, I write. I use that time to write a unit test in Scala and in the end, it is that confidence thing. You can write code and be confident that things will work the way you expect.

[SPONSOR MESSAGE]

**[0:36:08.4] JM:** You are building a data intensive application. Maybe it involves data visualization, a recommendation engine or multiple data sources. These applications often require data warehousing, glue code, lots of iteration and lots of frustration. The Exaptive Studio is a rapid application development studio optimize for data projects. It minimizes the code required to build data rich web applications and maximizes your time spent on your expertise.

Go to exaptive.com/sedaily to get a free account today. That's exaptive.com/sedaily. The Wxaptive Studio provides a visual environment for using backend algorithmic and front end components. Use the open source technologies you already use but without having to modify the code unless you want to of course. Access a Cayman's clustering algorithm without knowing R or use a complex visualization even if you don't know D3.

Spend you energy on the part that you know well and less time on the other stuff. Build faster and create better. Go to Exaptive.com/sedaily for a free account. Thanks to Exaptive for being a new of Software Engineering Daily. It is a pleasure to have you onboard as a new sponsor.

[INTERVIEW CONTINUED]

**[0:37:39.5] AB:** So you're back on the JVM. So you got tired of the JVM and you left and now you're back but the language is more fun.

**[0:37:46.5] AKJ:** Yes, exactly.

**[0:37:49.6] AB:** I think you mentioned this but do you find that it is faster to write Scala is slower but worth the tradeoffs or compared to Python, compared to Java?

**[0:38:00.5] AKJ:** I'd say that far it is a bit slower than Python but there are caveats on it right? So it is slower to write a piece of code but then the mono effort you have to maintain that code and to test that code is a lot lower than Scala for me, in the long run Scala is just a faster language than Python.

**[0:38:22.8] AB:** So not faster to initially develop but faster like the all in times?

**[0:38:28.6] AKJ:** Yes.

**[0:38:29.7] AB:** So how about maintenance? So maybe you don't know, maybe this is so new that you haven't had to do much maintenance of it but –

**[0:38:38.8] AKJ:** Yeah, so we haven't done that much maintenance because the system is very new. Whenever we have to refactor something, it's very easy too because your IDE does it for you basically so you don't even need to think about it. We just click two buttons and that's it.

**[0:38:58.3] AB:** The compiler gives you confidence I would imagine too around these refactoring's because you get some sort of error checking?

**[0:39:06.4] AKJ:** Yes, exactly. So I remember the first time I had to refactor some stuff in Scala, I was just surprised that in Python it would take me like, I don't know an hour or maybe not an hour but a lot of time and in Scala, I would finish it in less than one minute and I was just so surprised. I mean it was that sort of thing that after working with Python a lot of times you just become so used to that that whenever you have something nice you're like, "Oh that's nice".

**[0:39:38.6] AB:** Scala being on the JVM should in a lot of cases be much faster than Python. Do you have any sort of numbers about that?

**[0:39:49.3] AKJ:** I wouldn't say – so we haven't actually done any benchmarks that you would be able to trust like that comparison, the exact same code in one environment or in the other. The one thing that we have going on is that well we rigged it in the whole system, the whole session generation in Scala. We have also react that things and some of the performance gains that we saw was from we are protecting any use in D in memory cache and using a steak, we decrease latency from –

I don't remember that, it was maybe 700 or 800 milliseconds through things should be off tens of milliseconds. So it was more than 10 times that was very good. Also the number of servers that we need to use to serve the same amount of traffic decrease by, how much was it? It was like maybe 10 times or so.

**[0:40:51.3] AB:** Wow, so that is a big savings cost of bottom line I guess.

**[0:40:54.8] AKJ:** Yes and also just the fact that Scala and what the JVM in general does better in multi-threaded environment and multiple assessing and it is able to just run a lot of stuff at the same time, it's nice. Yeah so one thing that Scala has that Python does for example is called futures. So what a future does is basically is a unit of a synchronous computation. So whenever you do a request, you get the response of the future but the value is not that.

The value is you are waiting for the value in another thread. So what happens is you don't block your original thread and because of that, you are able to do a better job than in concurrency for example and that's one thing that we have seen in Scala is that our servers are able to handle a lot more concurrent requests than Python because in Python, you're have SGI and whatnot and then whenever you have a request and then you are waiting for IO that thread is completely blocked and if you have, I don't know, let's say 20 threads in your server then you have one last reserved traffic.

**[0:42:10.4] AB:** Is there a Python way to deal with this or there is just not?

**[0:42:14.5] AKJ:** Not that I know of, well not out of the box. Maybe there are some libraries that do for example the actor model which is something. It's the thing that languages like Elixir and Airlang do out of the box too. So it just handles concurrency better. I think there is something for Python as well. There is ACA, it is a library for Scala and I think if you use that, you might be better off but not out of the box not if you are using Flask Corp Pyramid.

**[0:42:48.8] AB:** Are you using the actor model in your session generator?

**[0:42:52.6] AKJ:** Not in the session generator, no because we wanted to – the interactions with IO are very simple in our session generator and there is also the overhead to get it set up first because nobody in the company had that kind of know how. So we chose to first to do the more common approach but it is something that after you start reading it you become interested in.

**[0:43:19.0] AB:** Are you using actors at all? I am just curious because you mentioned it so.

**[0:43:25.0] AKJ:** I was thinking of implementing that for the offline part of session generation because we have a lot of data and in that situation, it wouldn't make more sense to have a data pipeline that uses actors. For now, we still haven't had the opportunity to do it because we are still working on some other things and that's not the highest priority yet.

**[0:43:49.3] AB:** Makes sense, so now that you're rewrite is over what were the business benefits of the rewrite? Was it a success?

**[0:43:58.7] AKJ:** Yes, so for now I think it's a success. It is not completely over it because we have some features to port but it is mostly done. It's a success because we were able to have those cost benefits to that. It's just a lot cheaper to run or to serve traffic with the rewritten code and the rewritten lecture than the original one. Also in terms of developer productivity, it is also very good because it's a bit weird of I say it but my feedback and the feedback from the other developers is that it's just less painful and I think painful was the word they actually used.

**[0:44:47.2] AB:** So what makes it weird?

**[0:44:50.4] AKJ:** Well, I started the whole process of moving to Scala so I might be very biased towards the new system.

**[0:44:59.6] AB:** So you don't find it weird but other people do, is that what you're saying or?

**[0:45:03.7] AKJ:** No, it would be weird for me to say it because I am biased but other people, I also got feedback from other people that it's less painful. It's the thing that I talked about like you can write code, there's confidence and I think that is very important.

**[0:45:19.6] AB:** Okay, I understand. Would the rewrite have been successful or as successful if you had made the architectural changes but not the language change?

**[0:45:31.9] AKJ:** I think partially. So we would have with the architectural change, we would see improvements in latency. I think in Python they wouldn't be as large. One reason is because Python is a bit slower than Scala. The other reason is that having a thread safe cache in Python is not as trivial as it should be. So I think that would be one problem but also generally, we would lose all of the benefits of developer confidence of not pushing and breaking changes because it's all dynamically typed and it would be much harder to make larger changes or just to know what they did as structures are.

**[0:46:24.6] AB:** What is it like working at the company with such a focus on learning. I think are you language learning yourself? Does a company have a learning perspective based on what it does?

**[0:46:37.6] AKJ:** Yes, so I think it is very interesting to work here and so I am language learning. So my native language is actually Portuguese. So I was born in Brazil, I am a Japanese-Brazilian so my native language is Portuguese, second language is English. I've learned Japanese and I kind of know Spanish so it's very fun to be surrounded with people who have the same interests and for learning, there are some people who are learning a ton of languages and they know a lot of languages.

And even with our community, sometimes when we meet some members of our community, it's very interesting because they have all of these view of the world that you wouldn't – that you don't usually see in your daily life. People who want to learn new cultures and learn new languages and have very broad horizon I'd say.

**[0:47:36.0] AB:** Yeah, I could see that would be refreshing, talking to people who have a global perspective. Well it's been great to talk to you about this rewrite and I am glad it has been a success. Thank you so much for you time.

**[0:47:50.1] AKJ:** Yeah, thank you for inviting me here. It was great talking to you.

[END OF INTERVIEW]

**[0:47:58.0] JM:** At Software Engineering Daily, we need to keep our metrics reliable. If a botnet started listening to all of our episode and we had nothing to stop it, our statistic could be corrupted. We have no way to know whether a listen came from a bot or a real user and that's why we use Incapsula to stop attackers and improve performance. When a listener makes a request to play an episode of Software Engineering Daily, Incapsula checks that request before it reaches our servers and it filters the bot traffic preventing it from ever reaching us.

Botnets and DDOS attacks are not just a threat to podcasts. They can impact your application too. Incapsula can protect API servers and micro services from responding to unwanted requests. To try Incapsula for yourself, go to Incapsula.com/2017podcasts and get a free enterprise trial of Incapsula. Incapsula's API gives you control over the security and performance of your application and that is true whether you have a complex micro services architecture or a WordPress site like Software Engineering Daily. Incapsula has a global network of over 30 data centers that optimize routing and cache your content.

The same network of data centers are filtering your content for attackers and they're operating as a CDN and they are speeding up your application. They are doing this all for you and you can try it today for free by going to Incapsula.com/2017podcast and you can get that free enterprise trial of Incapsula. That's Incapsula.com/2017podcast, check it out. Thanks again Incapsula.

[END]