EPISODE 973

[INTRODUCTION]

[00:00:00] JM: Roblox is a gaming platform with a large ecosystem of players, creators, game designers and entrepreneurs. The world of Roblox is a three-dimensional environment where characters and objects interact through a physics engine. Roblox is multiplayer and users can interact with each other over the Internet. Roblox is not one single game. It's a system where anyone can design and monetize their own games within Roblox.

Over the last 14 years, Roblox has grown to be massively popular. As the product is grown, the software has evolved to meet changes in consumer demands and engineering constraints. Client devices include mobile phones, desktop computers and virtual reality. All of these clients must have a consistent experience in graphics and functionality.

The backend platform has to support a high-volume of concurrent players who are accessing a high-volume of content. The networking needs to support multiple players operating in an environment that demands high-bandwidth.

Claus Moberg is a VP of engineering at Roblox and he joins the show to discuss the engineering of Roblox and the future of gaming.

We are hiring a writer and an operations lead. The writer is somebody who would be writing about software engineering and computer science and a part-time capacity, and the operations lead will be helping us run our business more effectively, and that's also a part-time role. If you're interested in either of these roles, send me an email, jeff@softwareengineeringdaily.com. We don't exactly know who we're looking for. We don't exactly know what skillset we're looking for. So don't be shy. I'd love to hear from you.

[SPONSOR MESSAGE]

[00:01:51] JM: Cox Automotive is the technology company behind Kelly Blue Book, autotrader.com and many other car sales and information platforms. Cox Automotive transforms the way that the world buys, sells and owns cars. They have the data and the user base to understand the future of car purchasing and ownership.

Cox automotive is looking for software engineers, data engineers, Scrum masters and a variety of other positions to help push the technology forward. If you want to innovate in the world of car buying, selling and ownership, check out cox autotech.com. That's C-O-X-A-U-T-O-T-E-C-H.com to find out more about career opportunities and what it's like working at Cox Automotive.

Cox Automotive isn't a car company. They’re a technology company that’s transforming the automotive industry.

Thanks to Cox Automotive, and if you want to support the show and check out the job opportunities at Cox Automotive, go to coxautotech.com.

[INTERVIEW]

[00:03:09] JM: Claus Moberg, welcome to Software Engineering Daily.

[00:03:12] CM: Thanks. Super excited to be here.

[00:03:14] JM: You work at Roblox. Roblox has existed for 14 years. It’s a large gaming ecosystem. Describe the Roblox platform in its current form.

[00:03:27] CM: Absolutely. So perhaps the easiest 30,000-foot description is that what YouTube is to video content, Roblox is to 3D multiplayer video game content. That is that we produce a set of tools and a platform that lets anybody in the world create a 3D multiplayer online experience or game, and then we also build the platform that lets random people from the Internet come and play that game in a superfun social context.

So we have about 100 million players that come and experience of these games that they themselves have created every month, and it's just an amazing destination where you can go play super original, creative gaming experiences with your friends online. If you so want, you can actually make them yours self.

[00:04:29] JM: Give a few examples of the kinds of games that people build and play.

[00:04:34] CM: Yeah. I mean, it really looks like a microcosm of the greater gaming industry at large right now. There's everything from social online hangout games, to first-person shooters, to mobaz, to – You name it, whatever your favorite game genre is. There's probably a version of that that’s live on Roblox today.

Traditionally, our audience has skewed sort of towards the 9 to 13-year-old age demographic. So a lot of the games sort of skewed towards that demographic as well. Though especially as of late, that’s really starting to change. We see some more competitive content that is skewing towards older age demographics as well becoming very popular on the platform.

[00:05:27] JM: Describe the process that someone goes through to make a game for Roblox.

[00:05:35] CM: So Roblox provides a game development environment the we call Roblox Studio. It is a full-featured game development environment. If you're familiar with the Unity IDE, or the Unreal IDE, it'll look pretty familiar to you with some specific differences.

But within that, you have control over basically everything in your game. You can script gameplay in our scripting sandbox. The language that we provide to users is Lua, which is a very popular game scripting language in the industry. Top Roblox experiences will include about upwards of 50,000 lines of code, but they also have full custom 3D meshes that are uploaded to Roblox’s asset services. They include full maps and 3D environments that you build. We have a terrain editor that lets you build beautiful three-dimensional multi-material terrain.

Basically it looks relatively similar to what building a game in any other major popular 3D game IDE looks like today. The one big difference is that sort of central to the Roblox ethos is trying to make that game development process dramatically more efficient for the developer. o most of the other sort of AAA development environments that are out there, you start with a complete blank slate. All your defaults are empty when you start a project, and you have to build everything from the ground up.

Roblox really starts with the idea that it we are going to provide our developer with a starting point with rational defaults for everything, from the strength of gravity, to a base plate that's the sort of basis for the 3D world that they're building, to a set of player scripts that describe how our 3D avatar would interact with its environment in the game.

The developers always able to modify all of these defaults and build their own player scripts or their own gameplay elements, or obviously radically change the base plate and all that kind of stuff. But we think that by providing a reasonable set of defaults, they can actually move dramatically faster, and I think that's what we've seen with our developer and with our content thus far, is that because we provide this really strong set of default to sort of mimic even like real-world behavior of stuff, real physics, real character, actions and animations that mimic sort of anthropomorphic expectations and stuff like that, that they are able to iterate much more quickly and spend a lot less time building and experience on our platform than it would take them to build that same experience on most of the other game IDEs that are available in the industry.

[00:08:42] JM: There are many facets of the engineering that we'll get into, because Roblox is 14-years-old, and over 14 years he there're lots of ornate architectural decisions that I'm sure have been made and lots of interesting pieces of innovation that we can touch on. I want to continue with a bit of a top-down exploration for what this product is, because I know that there are a lot of people that are – I was unfamiliar with it when I started digging into it. But I understand now that this is one of those ecosystems that may be under the radar for a lot of people, but has a gigantic following.

I mean, the Internet enables these kinds of amazing gigantic ecosystems. Can you give me a brief history for how the product has developed over time?

[00:09:37] CM: Yeah, absolutely. So first off, if any of your listeners haven't heard of Roblox, I would encourage them to either go talk to the closest 12-year-old that they have in their life or go talk to any parents they know of kids in that like 9 to 13-year-old age demographic and ask them about Roblox. You will get an outpouring of emotion from anybody who fits either of those demographics. If you're in that 9 to 12-year-old age demographic, the odds are in the United

States today, it's better odds that you do play Roblox than that you don’t. We have over 50% market penetration that age demographic, and it is just a huge phenomenon.

[00:10:19] JM: What?

[00:10:19] CM: Yeah. So more than half –

[00:10:21] JM: More than 50% of the 12-year-old population plays Roblox.

[00:10:26] CM: In the United States and other English-speaking countries today. Yes.

[00:10:31] JM: Okay. Nice stat.

[00:10:33] CM: So going back to the history, yeah. So our CEO, founder and CEO, Dave Baszucki, started the company with one of his close friends, Eric Cassel, like you said, 14 years ago. So, 2006. They first released our development environment, Roblox Studio, two years later. It was sort of almost like a demo at that time. But the concept from day one has always been really consistent that the idea is that we are building ways for people to do online 3D co-experience with their friends.

The initial idea to found the company really came out of Dave's previous company, which was a 2D physics simulation tool for the educational space. So basically allowing like kids to do their physics experiments not in a real-life lab, but on the computer. This is like back in like the Macintosh 1 kind of days.

So they built this platform that led kids sort of attach blocks that had masses and then like smash them together to do these physics experiments to see sort of what would happen from a Newtonian perspective if a big block and a little block hitting each other. What happens with momentum and all that kind of stuff? They created all these curriculum and the company got acquired for a nice little exit. But what they noticed while building that is that once the curriculum was over, once the kids were done with the official experiment in class, they would just sit there and like build cars and then crash them together, or they make these little like short little obstacle courses that you had to navigate your car on. They're basically using this physics simulation software as a gaming environment and they thought, "Huh! That's really interesting."

They decided after exiting that company to start a company that was explicitly focused on building a user-generated content gaming platform. It's like I said, they found the company in 2004. 2006, they released Roblox Studio, which was the first time people could really build 3D multiplayer experiences that were online and hosted for them.

The company from that point for the next like 8 to 9 years grew, but grew relatively slowly. Until a point in sort of late 2015 when something changed, and we have a lot of conversations internally about what that thing was, and the answer is it wasn't just a single thing, but it was a confluence of different factors, but all of a sudden the companies went from going relatively slowly 25% year-over-year or something like that to growing at over 100% year-over-year and just completely blew up.

We really continued that growth trajectory ever since late 2015 or early 2016 to the point now where we have over 100 million monthly active users. It's really been a wonderful sort of Silicon Valley growth story where that hockey stick actually happened and seems to be continuing to happen even to this day.

[00:13:47] JM: Okay. So I know that a product like this, basically, if you look at any angle of the software, there's going to be some interesting stuff we could discuss. I've done a few shows on gaming. One thing that seems to be characteristically interesting in the shows about gaming is networking. What I mean by that is when you've got a massively multiplayer online game like a Roblox, you've got these people that are independently exploring a 3D environment as a character that's running around.

So you've generally got like a camera that's positioned kind of behind the character, and the character is like a 3D model running through space. So that's your perception. In reality, the perception is like, okay, you've got the whole world on your phone or on your PC. But, of course, this world is like so gigantic that there's no way it makes sense to load the entire world on to your computer and continuously sync it with the server.

In reality, many of these MMOs, these massively multiplayer online games, you've got a gigantic world that is hosted on a server and it's periodically syncing with each client and giving the client enough world to explore on their client device, and this creates a very interesting networking problem because you have to ask, "What are we passing in those packets between the backend and the frontend? What are we giving to the client? What does the client – What is the minimum amount of information that the client needs to render on their device, and is there path to graceful degradation?" If the client is on a mobile device on a spotty cellular network, can we give them a good experience? Can we pass them enough data from the backend?

I guess I'd like to just explore the networking challenges of building an MMO that is available across all the platforms, whether you're on an Xbox, or an iPhone, on a T-1 connection, or a cellular connection. Tell me about the networking.

[00:16:22] CM: Yeah. First off, that's a great tee up to the problem, though I would take it one step further and say that most MMO's benefit from being built by a sort of vertically integrated studio where the same company that's building the network infrastructure to support the game is also building the actual contents of the game.

So you can have the level designer sitting down with the networking engineer and sort of making decisions on tradeoffs between whether the networking infrastructure can support the crazy number of triangles that he wants in the volcano or whatever, right?

Roblox is a user-generated content platform. We have zero control over the content that our community is building. So we have to actually do the much harder thing, which is to solve the general case and build a system that basically opens as much capability for the game developer as is possible across all of the devices that we support.

So the first question is what frontend client devices do we support? As of today, we support iOS devices all the way down to the iPhone 4S, which is at least my modern standards ancient, and its CPU, GPU capabilities, etc. Obviously, we support the Xbox 1. We support PC and Android down too. I don't remember the minimum supported version of Android right now, but it's essentially the equivalent of 4S on the iOS side.

So the big differentiator for Roblox when it comes to this type of networking is our game engine. We have a fully proprietary custom in-house game engine that we have built from the ground up explicitly to do multiplayer online “streamed 3D experiences”, where you have some geometry and rendering done locally on the device, but you have the exact same version of the game engine running on the game server and supporting all of the simultaneously connected clients.

So there is a source of truth on the server running our C++ game engine. Then we have obviously a robust networking layer that arbitrates what aspects of the game world are going to be passed off to the client in terms of ownership for physics simulation. Obviously, the vast majority of rendering happens on the client, and that's all sort of the optimization that our game engine was built with from the ground up from day one, and it's really from that perspective the only 3D game engine on the market that was built with this as its primary or sole use case.

Even when you compare it to Unreal or Unity, both of those were even in their current conception built to have the vast majority of the triangles that they’re rendering downloaded in a large DLC or whatever when you first download the game. Roblox is very different, and that all of our geometry is streamed from the game engine from the moment that you actually join a game server. None of that is local to your device prior to joining any of our individual experiences. As a result, the game engine is actually optimized for not just passing up and down ownership, but also passing down the specific geometry of the world that the developer has created.

[SPONSOR MESSAGE]

[00:20:09] JM: Being on-call is hard, but having the right tools for the job can make it easier. When you wake up in the middle of the night to troubleshoot the database, you should be able to have the database monitoring information right in front of you. When you're out to dinner and your phone buzzes because your entire application is down, you should be able to easily find out who pushed code most recently so that you can contact them and find out how to troubleshoot the issue.

VictorOps is a collaborative incident response tool. VictorOps brings your monitoring data and your collaboration tools into one place so that you can fix issues more quickly and reduce the pain of on-call. Go to victorops.com/sedaily and get a free t-shirt when you try out VictorOps. It's not just any t-shirt. It's an on-call shirt. When you're on-call, your tool should make the experience as good as possible, and these tools include a comfortable t-shirt. If you visit victorops.com/sedaily and try out VictorOps, you can get that comfortable t-shirt.

VictorOps integrates with all of your services; Slack, Splunk, CloudWatch, DataDog, New Relic, and overtime, VictorOps improves and delivers more value to you through machine learning. If you want to hear about VictorOps works, you can listen to our episode with Chris Riley. VictorOps is a collaborative incident response tool, and you could learn more about it as well as get a free t-shirt when you check it out at victorops.com/sedaily.

Thanks for listening and thanks to VictorOps for being a sponsor.

[INTERVIEW CONTINUED]

[00:21:58] JM: A game like Roblox has something called a physics engine, and a physics engine essentially outlines the rules of the physics of the game. If my character jumps, how does gravity interact with that character? Is it like the real world where it's 9.8 m/s^2 or whatever, or is it like you jump and then like – And you can double job or something?

[00:22:29] CM: Exactly.

[00:22:31] JM: Is the physics engine something that is simple enough that you can keep entirely on a client device, or is that something where it's like you have to take make a round-trip to make physics decisions?

[00:22:43] CM: So the physics engine itself is definitely something that could be kept on the device, but you actually run into the networking and ownership problems that we were just talking about, right? If you imagine a game like Grand Theft Auto where you can be driving a car and I can be driving a car and then we crashed them together, right?

If you assume that there will be some round-trip time from my clients to the server and from your client to the server, from my perspective rendering in real-time on my device, if you make a last-minute adjustment right before our two cars crash, I'm going to not see that, because that last-minute adjustment didn't make it to my device by the time my last-minute input got sent up to the server and a new sink of the real-time geometry locations came back down from the server to my device. So you might see a different car crash than I would see on my client.

So, really, one of the core competencies of a game engine like what we've built at Roblox is how you arbitrate ownership of physics and, frankly, how you do some predictive modeling of even things down to user input to try to come up with the best guess of how these multiplayer physical interactions are actually going to work in the game space. Do you want to make it something where all physics is owned by the game server and so you know you have a completely level playing field?

But a lot of the input and controls then are going to feel super laggy, because in order to see the results of an input, you have to take the input on the device, send it up the to the game server, which is obviously some hundreds of miles away from you and almost the best case scenario and then send that? Have that determine what happened from a physics standpoint and send the results back down for rendering locally on your device, or do you want to have the no lag option where you actually have direct control in rendering of your car or wherever it is locally on your device, but where the reality in your game client can, as a result, diverge from the reality on the game server and consequently from the reality on another player's local client device because all of it is having to be arbitrated and synced back out of real-time from what you're seeing being rendered on your screen?

In Roblox, we have a really complex set of code that basically says, "Well, if there's no other players near you, obviously we can have physics owned locally on your client." Then as other players and other dynamic interactions come closer and closer to you, this essentially physics ownership codebase kicks in and tries to determine the optimal placement of stuff either in the cloud or on your local client and which version of the world takes precedence.

[00:25:41] JM: That's crazy. I mean, this is the kind of thing – This is why I like talking to people about the gaming business. You have these kinds of crazy problems. The 14-year trajectory, what's interesting about a 14-year trajectory is that takes you back to pre-cloud days. There was a show we did with Intuit fairly recently, and Intuit made this crossing of the chasm on-prem to the cloud, and this is a pretty interesting area of debate or architectural decision-making, because there are many companies that have investments in colo's or their own on-prem infrastructure.

They get to a point where like, "Well, we've got this infrastructure. It's useful, but there're these cool cloud services, and maybe it'd be cheaper if we just didn't own any infrastructure." So they have a set of very interesting decisions about should they get rid of their on-prem infrastructure? Should they move to the cloud?

With Roblox, I assume it's a little bit simpler, because the real traction with the product didn't start until like the post-cloud era. But I am nonetheless curious. Are there uses of on-prem infrastructure or is everything entirely cloud?

[00:27:00] CM: No. There's actually two aspects to that. One is you're absolutely right, that when they started Roblox, the word cloud hadn't really even been coined yet. In fact, if you ask the engineers who are at the company at the time, we actually have our game server infrastructure internally. It was referred to as RCC Roblox compute cloud, and they swear up and down that the industry had not used the word cloud before they decided to call it that internally. I don't know if I believe that if I wasn't here at the time. But if you believe them, they coined the name cloud, which is hilarious.

In terms of our current infrastructure, there's actually another aspect of it as well, which is that operating a company at our scale, we are now having to make decisions about not just whether or not it's effective to move stuff out of our legacy on-prem data center in Chicago into the cloud, but also at this point our scale justifies building out our own global essentially private cloud infrastructure.

So one of the big investments that we've made at the company over the last two years is literally acquiring our own global fiber network, standing up a series, I think we're at 16 or 17 edge termination pops around the world and running all of data between our own fully owned data centers in Chicago and Ashburn, Virginia over our own private fiber infrastructure to those edge termination noes.

So from our perspective, the way we think about this is if you're a sort of full stack application engineer building a service, you shouldn't actually care what infrastructure that service is being hosted on. There should be an abstraction layer that says, "Hey, I just need to host this service that needs to be able to serve X-amount of volume and to be able to scale at Y rate up to future volumes."

Then beneath that abstraction layer, there's a microservice platform team that doesn't actually care explicitly what your service does. They don't care if it's serving that is social graph or if it's serving the in-app purchase functionality or whatever. They just see a service that needs hosting with a specific geographic distribution of consumption and specific growth rate with certain likes signals around volume and stuff like that. Their job is to place that service on the best possible infrastructure, whether that's public cloud infrastructure, our own bare metal for cost reasons, or our legacy data center because it's one of our sort of legacy monoliths that we haven't been upgraded into the latest stack yet.

From our perspective, basically, we try to have a team whose sole mission is making optimal decisions around what infrastructure is best to host individual services and then to abstract that away from the work that an actual application engineer would need to do. So optimize the result in user experience. Does that make sense?

[00:30:18] JM: Yes. So speaking of services, I assume that for many years, Roblox had a monolithic CodeBase because I think over the course of 14 years there has been this increased promotion of the breaking up of a monolith into domain specifi c services, or service-oriented architecture, or microservices architecture or whatever the term du jour is.

So that probably happened, that kind of breaking up of the monolith or addendums to the monolith. That probably started happening before you joined the company three years ago. Can you just tell me the rough picture of the architecture today? Is there some central monolith and then some other services, or is it like a lot of just a flat services architecture?

[00:31:11] CM: So in an interesting way, this kind of goes back to a core Roblox engineering principle, which is sort or set your Northstar in terms of the way you want to solve a certain class of engineering problems. Then sort of make sure that any near-term engineering technology choice that you make moves your stack towards that long-term Northstar.

So it essentially is another way of saying, “Make sure you doing the engineering investment necessary to avoid any sort of unnecessary technical debt.” This is a core Roblox engineering principles. Something we think about every day as it relates to architecture vis-à-vis like monoliths and microservices. We still do have a core of monolithic codebase that powers some percentage of the Roblox www website and associated services.

Basically, the reason that still exists is that it has proven capable more or less of serving the 50 to 100X traffic that increase that is experienced over the last 3 to 4 years as we've gone through this crazy exponential growth curve. There is an internal mandate today and has been for years now where if you are doing significant modification to existing services, you should simultaneously do the work that is necessary to break them out of the monolith and stand them up as containerized microservice independently scalable, manage dependencies, etc.

So we have like internal tracking sheets that show the migration of core services out of the monolith and into these independently hosted microservices that now are a significant majority of our backend infrastructure. But there is not an abstract internal mandate that says, “Hey, stop all product work until the monolith is dead.” It basically draws back the idea that when that monolithic codebase was created, yes, it was a monolith, but it was also created with a ton of attention paid to scalability and sort of future magnitude of demand concerns. It's actually, even to this day, operating relatively well from a scalability standpoint.

The only part of it that's actually a really major negative drain on internal resources that add our current scale of headcount in the engineering department, having everybody iterating on the same monolithic codebase can be kind of painful and the release schedule becomes a little bit harder to manage and stuff like that. So we think moving to microservices actually unblocks individual teams and allows them to sorted control their own destiny in terms of shipping their own services and not have to wait for the daily monolith release to get their code out into user’s hands.

[00:34:15] JM: Give me some general perspective for where you're at in terms of assessing technical debt and resolving technical debt versus building out new features.

[00:34:29] CM: Yeah. What I often say to engineers that I'm interviewing is that for a 14-year-old company, the amount of technical debt we have is astonishingly low. I also say that with a grain of salt, because I've never seen the codebase of any other 14-year-old company in my career. So it's easy for me to say that.

But, again, the Roblox operating principle from an engineering perspective is, “Look, we’re a platform. We’re very unique in the game industry and that nothing we are building today has to ship tomorrow.” If you're at a traditional vertically-integrated game studio, you have your content deadline, and your game has to be done by October first. So I can be cut on to optical disks so it can be on store shelves by November first so you can make the holiday season or whatever.

So the entire development exercise is basically like nine months of trying to figure out what you can actually get into this release. Killing yourself through crunch time for 80-hour weeks or whatever for those nine months to get every last line of code you possibly can and by that deadline, and then you're done. People go and take a vacation or if the game is not successful, they get laid off or whatever, until they sign up again two or three months later to do it all for the next year's release.

Roblox is totally different. Everyone on my team could go on vacation for six months, and some 17-year-old in Bubuque, Iowa would still release a new title tomorrow that is marginally better at retaining users are marginally better at monetizing users. As a company, we would still hit every single one of our user acquisition, short-term user acquisition and monetization goals even if the big thing that my group is working on doesn't ship this quarter even, right?

So it's dramatically more important for us to build the thing right than it is for us to build it on any arbitrary timeline. Internally, it's frankly celebrated. If somebody's working on a project and they come up with an implementation that’s twice as good but takes twice as long. That's always something we want to see, is people taking the time to build things the right way in a way that avoids unnecessary technical debt and gets the product out.

We still want to do that in an iterative culture and we don’t want to decide to make a big bet. Spend three years heads down, no product feedback. Just building the thing and just like flip a switch and see if it works. That's not the way we work. But it does mean what we normally do is set some major aspirational guiding principle, a Northstar that we want to work to over the next like say 3 to 5 years.

Then every near-term sort of iterative choice that we make, we evaluate whether the near-term implementation or the near-term sprint or the near-term test is moving us generally towards that Northstar guiding principle or away from it. If it’s moving us away from it, we don't do it. We don't ship it and we won't take that debt. But it is moving us toward it, we will do the most incremental solution we can that still validates our long-term assumption. Does that make sense?

[00:37:46] JM: It does.

[00:37:46] CM: The result is actually – Again, you could consider our existing monolith as technical deb. I frankly don't. I say it's actually a marvel that a significant amount of codebase that was written 5 to 10 years ago is serving 100 to 200X the traffic that we had at the time that code was written. I don't think that’s technical debt. It doesn't mean it can't be significantly improved, and we’re doing that every day. But it’s an example of the amount of forethought and discipline that was used when that codebase was originally created and it’s something that we try to carry through to everything that we’re doing today.

[00:38:26] JM: What the programming languages that you use at Roblox, and do you have rules around what programming languages developers can use?

[00:38:36] CM: That’s a great question, especially for my group. So at a really high-level, the proprietary game engine that I mention is predominantly C++. I don't think there’s any surprise there. Our web stack, the monolith is .NET, sort of Microsoft stack. Though we now have a microservice architecture that gives developers a pretty broad mandate to choose the best language that they think is applicable to their individual problem that they're trying to solve. So we have code hosted in Go, or Python, or a whole bunch of other sort of languages de jure.

On the application side, our tech stack is extremely interesting and nontraditional. So for most mobile applications, especially cross-platform applications, you have this sort of devil's choice to make. Do you want to build your application logic and UI in the native language for each platform, which generally means you get best in class results at the cost of having to re-implement each feature at least once for every platform that you support?

So for us, that’s at least iOS, android, PC, Mac, and Xbox, and VR. If not, different implementations for both iPhones and iPads, right? Or do you want to use some sort of cross-platform codebase, historically embedded web views? More recently, React Native or something like that? Where you get the benefit of only having to write each feature a single time and having it ship everywhere, but the result kind of sucks.

Roblox chose a third door that's really unique to us. It's struck us that the very first thing we do when we port Roblox to a new platform is integrate our proprietary in-house C++ game engine at the lowest level graphics APIs that are available to us. So metal on iOS, Vulcan on android, OpenGL wherever possible, etc.

Our game engine itself has 2D UI rendering capabilities, but also our application infrastructure, the stuff that users navigate to play the games before they actually join a game server. The stuff where they choose a game to play or curate their social network or chat with friends or buy Roblox, our digital currency, all of that is both 2D, but with a huge smattering of 3D elements, your actual avatar itself, the way you decorate your avatar within the avatar editor and equip and un-equip 3D items and all that kind of stuff.

So we came up with this idea of saying, “Well, what if we actually used our game engine to render our player-facing app? The thing that a normal game studio would call the app shell. We've taken the last 2-1/2 years to basically migrate from our previous mobile infrastructure, which is a mix of native and embedded web views, to now when you download Roblox and open it from your home screen on your iPhone, everything you interact with after the splash screen is actually a Roblox game, scripted in Lua, which is the coding – The sandbox that we surface to Roblox developers, and rendered by our proprietary game engine.

The benefits of this have been huge, because any time in doing this that we found the performance of our game engine wasn't up to our standards for making a first-class social mobile experience. We were able to go into the underlying C++ game engine and fix its 2D rendering capabilities. So we made our scrolling frames dramatically more performant. We fixed the whole bunch of stuff in terms of how we manage our data models and all that kind of stuff. The dog fooding aspect of this was huge.

But at the same time, we get to build a feature a single time and have it shipped to literally every single platform that we support, and the result in product is, the vast majority of the time, indistinguishable from something that was built using that device’s native UI libraries. It’s been pretty phenomenally successful for us and I think it’s really unique in the industry.

[00:42:58] JM: I think when we did an interview with Google Earth, they had done something kind of similar to that.

[00:43:04] CM: Yeah. So that's exactly right. We’ve heard of a couple of other companies that have been in a similar space where they had some sort of proprietary in-house client rendering capabilities and decided to sort of dog food it for their own purposes. Again, for us, the results have been fantastic. It's just you have to go through the hard work of building those client’s time rendering capabilities, which at Roblox is like a five-year effort to create our proprietary in-house game engine.

[SPONSOR MESSAGE]z

[00:43:42] JM: As businesses become more integrated with their software than ever before, it has become possible to understand the business more clearly through monitoring, logging and advanced data visibility. Sumo Logic is a continuous intelligence platform that builds tools for operations, security and cloud native infrastructure. The company has studied thousands of businesses to get an understanding of modern continuous intelligence and then compile that information into the continuous intelligence report, which is available at softwareengineeringdaily.com/sumologic.

The Sumo Logic continuous intelligence report contains statistics about the modern world of infrastructure. Here are some statistics I found particularly useful; 64% of the businesses in the survey were entirely on Amazon Web Services, which was vastly more than any other cloud provider, or multi-cloud, or on-prem deployment. That's a lot of infrastructure on AWS.

Another factoid I found was that a typical enterprise uses 15 AWS services, and one in three enterprises uses AWS Lambda. It appears serverless is catching on. There are lots of other fascinating statistics in the continuous intelligence report, including information on database adaption, Kubernetes and web server popularity.

Go to softwareengineeringdaily.com/sumologic and download the continuous intelligence report today. Thank you to Sumo Logic for being a sponsor of Software Engineering Daily.

[INTERVIEW CONTINUED]

[00:45:31] JM: You're VP of engineering, right?

[00:45:33] CM: Correct.

[00:45:34] JM: So how are the engineering teams within Roblox arranged and how do they interact with each other?

[00:45:42] CM: So Roblox is a very flat organization. Just from a size perspective, we have about just less than 600 total employees. About 80% of that is products and engineering. So we're incredibly heavy on engineering especially. Essentially, we are arranged as collection of over 30 relatively autonomous full stack teams that have some full stack product ownership over a certain feature set within the Roblox offering.

So I currently oversee about five of these teams. My team's purview is on the Roblox application side, the stuff we were just talking about. Basically, you divide all of Roblox software into the bucket of stuff that helps 17-year-old's build awesome 3D multiplayer online experiences. My teams don't do any of that.

The other half is stuff that lets random people from the internet come and experience those awesome 3D multiplayer online games with their friends. My team does a lot of that and basically owns the entire user experience from downloading a binary on to their device to actually joining a specific game server. Selecting an experience and actually joining a game server for that game and getting dropped into the UDC environment.

So what that means is that these five teams that report to me, there's a social team, a game discovery team that sort of owns the Roblox equivalent of the Steam game store or whatever. There is what we call a universal lab team that owns the infrastructure and architecture behind this Lua application where we're dog fooding our own game rendering capabilities to build the app itself. Ad all of those teams have at least a director level product leader, director level engineering leader. Their own roadmap of our backlog of features that they own and that they're maintaining and looking to build in the future, and they're basically looked at as a sort of micro startup or a small autonomous company within the larger Roblox construct.

So they'll have their own status reports with our CEO who also is the ultimate sort of product visionary for the company, and they're in complete autonomous control over what they do on a daily basis. How they hire people into that team. What their prioritization is. They have to justify their decisions to their stakeholders. But apart from that, it's sort of their show to run.

The way we think about scaling the company moving into the future is we like these autonomous teams of 10 to 30 engineers and associated product designers, data scientists, etc. So we'd like to see the number of these teams proliferate as they themselves and their size stay about the same. That's sort of the way we've been operating for about the last 18 months and it's working really well for us.

[00:48:51] JM: I'd like to give the listeners a little bit of a picture for how the Roblox game economy functions. I realize that's not the area that you spend the most time in. But can you just give me an outline of how the Roblox economy works and what sorts of engineering problems that economic engine creates?

[00:49:21] CM: Yeah. It's actually fascinating. So the Roblox as we've been talking is this user-generated content platform. The fundamental idea is that people around the world come to

Roblox, use our tools to build these 3D multiplayer online experiences. They host those experiences on our infrastructure. They distribute them to our user base and we give them all of the tools, hosting and infrastructure they need to do that for free.

What they're able to do is sell digital assets within those experiences, and those digital assets, whether it’s a skin for a character or whether it's access to a VIP area within a game, you can look at sort of the entire ecosystem of sort of freemium monetization mechanisms that exist in the gaming industry at large and you'll see some version of that on Roblox today.

When they sell something on our platform, our players purchase it with our digital currency, which we call Robux Robux is obviously purchased with real currency, hard currency from wherever the player is coming from around the world. So if they’re in United States, they’re buying Robux with U.S. dollars, and developers who sell these digital assets within their game can actually cash out their Robux and earn U.S. dollars back.

So the top game developers on our platform are earning over million dollars a year from the game experiences that they're building on the platform itself, and it's a full-time job. Easily a full-time job. Very lucrative full-time job for them to build and maintain these top-quality experiences.

The exchange rate that they take that Robux and cash it out for, the program is called DevEx. The process of exchanging Robux for U.S. dollars. That exchange rate basically is how we pay our bills and become profitable. So there's a different exchange rate for buying Robux from cashing it out, but inherent in that is essentially their “app store fees”. So instead of paying 30% to Apple for just the privilege of listing your mobile game in their app store and having users download it, they pay a slightly higher percentage but have literally zero other costs other than their time to develop their game for publishing it on Roblox and it’s playable across basically every gaming platform in existence. Again, iOS, Android, PC, Mac, Xbox, etc.

[00:52:01] JM: Given your front row seat to the development of the Roblox economy and how the software is advancing, and I am sure you have a lens into how other aspects of the gaming economy are developing, Twitch, and Minecraft, and whatnot, all of these other verticals. So you just see up into the right growth.

Then I think I saw some stat about like the proportion of people who watch video games versus the proportion of people who watch sports, and it’s just like something like 10X or 5X. Something absurd that most you wouldn't expect. But gaming is not going to stay the same as it is today and it's definitely not going to shrink. So that certainly leaves only one alternative outcome.

[00:52:53] CM: Yup.

[00:52:54] JM: How is gaming going to evolve over the next decade?

[00:52:58] CM: It’s a great question. First of all, if I knew the exact answer to that, I'd be doing something else for a living other than just building software. But if I look at the sort of trends that exist in the industry right now, there are a few things that seem really clear. If you look at what's happening across the industry, going back to your networking questions, about where the actual computation for the experience is happening. 20 years ago, it was 100% on your computer, or your console, or your device, and the trend has clearly been over the last 15 to 20 years pushing those computations to somebody else's computer up in the cloud, right?

When you see things announced, like Google Stadia or any of the other competing platforms that are being announced by other large companies in the gaming space, it seems clear that some semblance of game streaming that enables players to access AAA quality content regardless of the local processing power of their device is a big part of that future. This is something that’s been part of Roblox’s vision for, again, literally 12 to 14 years at this point. The idea that we need an architecture that basically produces the best possible experience for every player regardless of the actual client-side hardware that's running that experience.

Our approach is pretty significantly different from what the pure streaming entrance, recent entrance into the market do, where Google is basically taking your user input, sending all of that up to the cloud, doing all of the rendering on the cloud and then streaming basically a video back down to your device. It’s nice because it makes all of those really hard things around physics ownership and rendering, all that kind of stuff, really easy. You just do it all on the game server.

It's hard, because it places big constraints on the size of the pipe that you have to send stuff up and send stuff down, and you have inherent lag on your user input because it's not being processed locally at your fingertips. It’s being processed to hundreds, to thousands of miles away on somebody else's computer. So that’s sort of a trend that I think we’ll see continue, but I think the implementations that we see in that space will become significantly more complex than what we're seeing from a bunch of the new entrants in the space today.

When we take a step even further back and look at the trends in the industry, the big question is how far and how soon do we get to full immersion, right? Four years ago everyone was like, “Oh my gosh! It's finally time for VR.” Four years later it seems pretty clear that VR is still – Thankfully AR as well, is a wonderful tech demo that for a bunch of different reasons has not had its mainstream moment yet.

When I was hired at Roblox three years ago, it was to lead a small VR team. We don’t have a VR team today, and largely less because the player base just hasn't materialized yet. There's not enough people with compatible devices to justify us having full-time engineering staff dedicated to the support of that as a platform even though our game engine is capable of doing it and we’re actually actively available on both Oculus and Vive platforms.

So I think when Roblox thinks about the future of immersive co-experience, we’re pretty confident that at some point in the future, and it could be 3 to 20 years away, there will be this idea of fully immersive online digital co-experience, where you are in using a bunch of different digital tools to consume an alternate reality with your friends. It's the sort of ready player one, or multi-verse, or whatever you want to call it. That's definitely what we feel like we are building and it’s really a question of when will consumer adaption of the hardware catch up to the digital platforms that we’re creating.

The great news is unlike a lot of other startups in this space, we don’t have any need for VR to take off to be successful. We’re growing incredibly fast on 2D consumption of devices today. We’re already sort of very, very profitable and doing very well with the current technology infrastructure that distributed around the globe. But we also think we’re fully ready for a future where 3D consumption of these same experiences predominates the user experience.

[00:57:58] JM: Claus, thank you so much for coming on the show. It's been really fun talking.

[00:58:02] CM: Absolutely I really enjoyed it. Thanks for having me.

[END OF INTERVIEW]

[00:58:13] JM: Better.com is a software startup with the goal of reinventing the mortgage industry. Mortgages are a $13 trillion industry that still operates as if the Internet doesn't exist, and better.com is looking for engineers to join the team and build a better mortgage experience.

The engineers at better.com are attacking this industry by bringing a startup approach into an industry filled with legacy incumbents. Better.com automates the very complex process of getting a mortgage by bringing it online and removing the traditional commission structure, which means that consumers can get a mortgage faster, easier and end up paying substantially less.

Better.com has a modern software stack consisting of Node.js, Python, React, Typescript, Kubernetes and AWS. They iterate quickly and ship code to production 50 to 100 times every day. Better.com is one of the fastest growing startups in New York and has just announced a series C that brings the total funding to $254 million. If you're interested in joining a growing team, check out better.com/sedaily.

Better.com is a fast-growing start up in a gigantic industry. They're looking for full stack engineers, frontend engineers, data scientists. They're looking for great engineers to join their quickly growing team. For more information, you can visit better.com/sedaily.

[END]