EPISODE 534

[INTRODUCTION]

[0:00:00.3] JM: Smart contracts are programs that run on the Ethereum blockchain. A smart contract developer pays ether to deploy the contract, and when a contract is deployed, every full node on the theory and blockchain has a copy of that contract code in their address space. Every full node needs to hold a copy of every smart contract and this allows every full node to process every call to any smart contract. If you want to call a smart contract, that contract will execute on every full node. When you call the smart contract, you are initiating a transaction. Like Bitcoin transactions, these Ethereum transactions get batched into blocks. Ethereum full nodes compete to solve the cryptographic puzzle associated with a block, but instead of mere financial transactions, these are computational transactions that are associated with a smart contract.

Raine Revere is a smart contract engineer and the cofounder of Maiden, and he joins the show to describe the smart contract creation and deployment. This is a great introduction to some Ethereum and fundamentals. But if you want to find our older episodes about cryptocurrencies, we've got plenty of shows. You can check out our apps in the iOS or android App Store. They have all 700+ of our episodes with recommendations and related links and discussions, and more. It's all open-source as well, and if you're looking for an open source project to contribute to, please come check us out at github.com/softwareengineeringdaily. We welcome all kinds of contributors, new developers and experts, engineers and non-technical folks, and I don't really like the term non-technical, but if you're in sales and marketing or design or whatever, we would love to have your help with the project. We've got a big vision and would love to have your help.

[SPONSOR MESSAGE]

[0:02:03.0] JM: Software Engineering Daily is brought to you by Consensys. Do you think blockchain technology is only used for cryptocurrency? Think again. Consensys develops tools and infrastructure to enable a decentralized future built on Ethereum; the most advanced blockchain development platform.

Consensys has hundreds of Web3 developers that are building decentralized applications focusing on world changing ideas, like creating a system for self-sovereign identity, managing supply chains, developing a more efficient electricity provider, and much more. So, listeners, why continue to build the internet of today when you can build the internet of the future on the blockchain. Consensys is actively hiring talented software developers to help build the decentralized web. Learn more about Consensys projects and open-source jobs at consensys.net/sedaily. That's C-O-N-S-E-N-S-Y-S.net/sedaily. Consensys.net/sedaily.

Thanks again, Consensys.

[INTERVIEW]

[0:03:18.9] JM: Raine Revere is a smart contract engineer and a cofounder at Maiden. You also formally worked at ShapeShift where you worked on the Prism Project. Raine, welcome to Software Engineering Daily.

[0:03:29.9] RR: Thanks so much. Glad to be here.

[0:03:31.7] JM: We've heard about many exciting ideas that smart contracts could eventually be used for. What are the smart contracts that are in production today?

[0:03:44.8] RR: Let's see. It's a great question. I mean, the project that I worked on is one of the examples I'm most familiar with. I helped to build the Prism Smart Contract System, which is portfolios of cryptocurrencies all stored on the Ethereum block chain, and they're in a private beta right now, about to open it up to an open beta, and it's running on the main net. The Ethereum main net has access across the globe and people can interact with the contract just by having any kind of wallet. They don't even need a full Ethereum node. Just having the wallet is enough to initiate these contracts or initiate portfolios with the smart contract systems.

So the Prism System is in production right now, just limited to a close beta until they scale. Other systems right now, Ethereum is in a space where it's really new. I mean, Bitcoin is only nine years old and Ethereum is just a couple years old. So I would say most things are really in

an alpha-beta kind of place, which is exciting. It's hard to tell what this technology is capable of and what kind of security concerns or optimization concerns are happening.

A fun example of a project that was in production and which had a big impact on the Ethereum network is a distributed app called CryptoKitties, and it was this game. It's this game where people can buy and sell cats that have specific genetic properties and then they can breed them and there's this whole collectible cat phenomenon. It's in production and people are using it and people are paying real money for these things.

In fact, for a time there, the activity with this collectible game was providing a substantial amount of traffic on the Ethereum network, and so things like that are ways that we're kind of seeing what the Ethereum network is capable of in production, even if it's just a fun game. That's honestly even a better place to see what the network can handle and what it can't handle when you can all laugh about what the application is. But we learn a lot even from those kinds of projects when people are designing and building larger, more production or commercial grade applications.

[0:06:18.2] JM: Indeed. Well, the CryptoKitties example is a good one, even though it is something of a toy example. Here you have a system where scarcity and uniqueness was enforced by the smart contract. You compare that to something like Pokémon cards. With Pokémon cards, you always have the risk that somebody was trading you a fake, like a fake Charizard or a fake rare.

[0:06:48.4] RR: That's right.

[0:06:49.1] JM: And with CryptoKitties you actually cannot do that, because the way that the smart contract is designed is that you actually have enforced scarcity of these digital objects that are created. So it's actually not a bad example of what smart contracts are capable of.

[0:07:04.8] RR: Yeah, that's right, and from CryptoKitties came a new standard for non-fungible digital assets. So cryptocurrencies are fungible. You can divide them down into small parts that are equivalent to each other in terms of value, and collectibles like CryptoKitties are nonfungible. One cat, one kitty is unique and has unique properties, and that same thing can be

applied to assets in the real-world. It's like every diamond is unique, or every piece of property is unique. So the ERC-721 standard has emerged to be able to describe — Or rather it might be 271. I might be mixing those up. But the standard emerged from a toy project that got a lot of attention and was actually utilizing this non-fungible digital asset that, as you said, has enforced scarcity, and that could be applied to many other scenarios.

[0:08:05.0] JM: So someday we will have insurance companies built on the Ethereum blockchain, and maybe the decentralized Uber or a decentralized Airbnb. Today we have CryptoKitties. We have things like ShapeShift Prism where you have bundles of securities that are mediated by smart contracts. So we're in very nascent stages, but someday we will get to the maturity level. What are the things that need to happen between now and the days of a decentralized Uber, for example? What do you think are going to be the points on the timeline where we are able to mature the ecosystem?

[0:08:46.7] RR: Yeah. So you're asking the hard questions. Sort of asking me to predict the future, but I'll bite. I'll give it a shot. Speculate a bit.

[0:08:54.1] JM: We'll get to the easy questions afterwards.

[0:08:56.6] RR: Great. Start with the hard ones. We've been working on Prism for almost 2 years and even though I'm not actively on that team anymore on an engineering side I still speak about Prism a lot, and there's a lot that I learned from my experience there and a lot — I engaged with the community and a lot of other engineers. What's happening right now is building out the infrastructure for one. There aren't the same programming environments and tools that exist in other environments. Even at amateur hackathons, there are new tools being created or new libraries being created that are pushing the engineering community forward and pushing the tools forward to make it possible to build production level application.

For example, I was on the judging team at EatDenver this weekend, and one of the teams that made it on stage because of their work created a gas profiler. Now, gas is the fuel of the Ethereum network. So this profiler not only looked at your program and pointed out possible areas where the computation was expensive, and thus making the gas costs and the actual monetary cost expensive for running that contract. They, in fact, ran a tracer so you could

analyze past transactions on a given contract and actually find the exact cost, like dollar cost per line of code when you run this contract. So you could look and find the line of code where your contract would cost the most.

These kinds of tools are really essential to building out the infrastructure, because blockchain is not just an information system, but an economic system. Then your application has certain costs and that's going to incentivize certain behaviors. So the costs of computation and the cost of transactions all plays into whether the system your building and designing is going to be usable by the broader audience.

For example, with this ShapeShift Prism, the digital asset portfolio platform, there is a cost to creating that portfolio, and due to the mechanics of the smart contract construction, right now it's rather expensive. You wouldn't be able to create a 50 or even $100 portfolio and get away and be able to maximize your profits if you're paying a $20 fee just to be able to do that.

There're optimizations that are really relevant to the accessibility of distributed applications. So costs have to be taken into account. So that's at the infrastructure level. At the onboarding level, there's a lot of misunderstanding out there about blockchain and Bitcoin. The mainstream media has been pretty wildly speculative and sharing extreme examples of how this technology has either been used for harm or is extremely volatile. So the public doesn't know what to think.

Non-technical people don't know what to think. Is this just a trend? Is this something that we can rely on? Is this secure? Non-technical folks have a hard time distinguishing between a hack that's due to bad security practices and a hack that's due to weak security within the application. Cryptocurrency and blockchain technology is actually founded on increased security, and we have a distributed database technology and a distributed digital currency that uses cryptography to ensure the validity of transactions instead of central parties, like banks or governments. Blockchain is fundamentally more secure, but for a non-technical person it can be hard to differentiate those two.

Just to go back to your original question, communication and education is a really key thing for adoption not only to help people understand what this technology is really capable of, but to ensure that it's adopted by a broad audience of diverse individuals and is not just a tool for rich

people to get richer or just a tool for tech folks like us to play around within our spare time. This is technology that is really poised to change the world and change economics. So there's a lot of education that needs to go in into ensuring inclusive adoption.

[SPONSOR MESSAGE]

[0:13:42.7] JM: QCon.ai is a software conference for full-stack developers looking to uncover the real- world patterns, practices, and use cases for applying artificial intelligence and machine learning in engineering. Come to QCon.ai in San Francisco from April 9th to 11th, 2018 and see talks from companies like Instacart, Uber, Coinbase and Stripe.

These companies have built and deployed state-of-the-art machine learning models and they've come take QCon to share their developments. The keynote of QCon AI is Matt Ranney, a senior staff engineer at Uber ATG, which is the Autonomous Driving Unit at Uber, and he's an amazing speaker. He was on SE Daily in the past. If you want a preview for what he is like, then you can check out that episode that I did in conversation with him.

I've been to QCon three times myself and it's a fantastic conference. What I love about QCon is the high bar for quality, quality in terms of speakers, content and peer sharing, as well as the food and the general atmosphere.

QCon is one of my favorite conferences, and if you haven't been to a QCon before, make QCon.ai your first. Register at qcon.ai and use promo code SEDAILY for $100 off your ticket. That's qcon.ai and you can use promo code SPDAILY for a hundred dollars off.

Thanks to QCon for being a sponsor of SE Daily, and check out qcon.ai to see a fantastic cutting-edge conference.

[INTERVIEW CONTINUED]

[0:15:28.6] JM: Let's talk about some of the basics of developing smart contracts. So I'd like you to define and outline the terms gas, gas price and gas cost and explain how those relate to smart contracts.

[0:15:45.0] RR: Yeah, great question. So gas and gas costs — Like I said before, gas, you can think of it as kind of the fuel of the Ethereum network. Let me get a little more detailed. Solidity is one of the languages that compiles down into a byte code that gets executed on the Ethereum virtual machine, or the EVM, and this byte code is a fully Turing complete programming language. Unlike Bitcoin, which has a more limited functionality or less expressive, less powerful language. It does have a language that you can perform certain operations, instead Ethereum has an entire Turing complete virtual machine and the language that you can build on that platform, and the one caveat is that because you're executing code on a distributed system, you're executing code on other people's computers. There needs to be a preventative mechanism for infinite loops and other spamming DDoS kinds of attacks, because you're executing this code on other people's machines.

So the general solution in Ethereum is that there is this intrinsic variable fuel called gas, which is a per operation cost. So you actually pay money for every operation that's being executed within the virtual machine. What this ensures — And that cost can be driven down to a market level where it's just a fraction of a cent, and that cost ensures that an infinite loop at a certain point, the cost will exceed the payer of that transaction, that if there is infinite computation, you could never provide infinite money to pay for that, and at a certain point the nodes in the network would give up before it caused any kind of DDoS attack.

So gas is translated into ether, and so the gas is a fixed cost per operation. So if you have, let's say, a storage operation and then a read operation, low, low level. I mean, we're talking the assembly level. Each operation is going to have a specific gas cost, and that's just in a number that was assigned and built into Ethereum based on the relative computational intensity of it.

Storing a value is a lot more expensive than reading a value, and doing a [inaudible 0:18:20.6] is a lot more expensive than that. So there's different costs assigned, and these gas costs mirror the computational costs. You could Google gas cost, gas with opcodes in Ethereum, and you could see a table of how much gas is mapped to the operation.

So a smart contract that's executing every function call consists of a series of operations executed in the virtual machine, and so thus the functions are going to cost a certain amount of

gas. Now that gas gets translated to an actual monetary value, and that's what gas price is. Let me explain this. Gas was that designation that reflects the computational effort for nodes on the network to execute your code, and the gas price is how much you're willing to pay, how much ether you're willing to pay per gas.

So if a function, say, costs a thousand gas, all of the operations add up to a thousand gas, then when you as the owner of that contract or the initiator of the transaction say, “Yeah, I want to execute this transaction on the Ethereum virtual machine on the main net so that the data is updated for everyone. I want this to be a live transaction, and I am willing to pay 1/100th of an ether for a certain number of gas.” So maybe my thousand gas function would cost .0001 ether, whatever it might be, and you say, “I'm willing to pay that much ether,” and what that does is it incentivizes the rest of the network to actually execute that code for you. It incentivizes the network to run the code and update the data across this distributed platforms, that everybody has the same shared data, and it's the gas price that is that bid for execution. It's the initiator of the transaction saying, “I have this function that cost this much gas and I'm willing to pay this much ether per gas in order for the network to execute this.” Okay? So that's gas and gas price.

I think you asked about gas limit, and the gas —

[0:20:47.4] JM: Gas cost.

[0:20:48.4] RR: Gas cost.

[0:20:50.5] JM: Or gas limit. Sure.

[0:20:51.6] RR: Yeah. I think in terms of the terms that are out there, we’ve got gas, gas price and then gas limit, and gas limit is that's more simple. It’s just saying that blocks in the Ethereum blockchain are of a fixed size. You can only fit so many transactions in there. So there's a gas limit. It depends, it varies. The network can raise and lower the limit through implicit voting mechanisms. In other words, if the network gets more congested, the limit will slowly grow, and if there's — If it's smaller, then the limit will slowly go down. That limit just determines the cap on how much gas total could be included in a single transaction or a single

block so that the network — So that there's the bandwidth constraints that allows the network to continue to confirm blocks every 10 to 15 seconds or whatever that is.

[0:21:52.3] JM: All these operations in the EVM, these are like opcodes, basically. Like traditional operating systems. You've got these low-level opcodes that operate on some number of variables, and this is the step right above machine code.

[0:22:10.9] RR: Yes, exactly. If it helps you think about this EVM is a virtual machine that was created for the purpose of Ethereum, but there is already work being done actively to translate the EVM into web assembly, for example. So these are conventional opcodes that are used in most Turing complete languages that we work with on a day-to-day basis.

[0:22:33.5] JM: Indeed. So every machine in the Ethereum blockchain network is going to execute the same ordered operations across the EVM so that every machine in that consensus network arrives at the same state so that these different machines, these different nodes, these different full nodes are in consensus with each other. The expense to the network is defined in the — I guess it would basically be the gas price per instruction, because if you would look at each instruction, there would be a price associated with that low level instruction. Then your high-level code gets compiled down into the EVM code, which has those different instructions that have prices associated with them, and then the cost of executing that smart contract or deploying the smart contract, depending on what stage we're talking about, but if we're talking about executing it, the cost of executing it is going to be the sum of the cost of each of those opcodes. Is that right?

[0:23:51.8] RR: That's right. The gas mechanism is a way to decouple computational cost from the market cost. So that's why you can have the same computation or the same instruction, somebody could bid and say, "I'm willing to pay .001 ether to have the network execute this operation," but somebody else could say, "Well, I'm willing to pay .003 ether to execute the same operation."

So the gas price allows you to decouple that. The gas prices could even change over time if — It's happened a couple times when we find that certain operations are a little more expensive than are expected. But for the most part, there is a mapping from computational cost, to relative

gas, to a monetary price in ether that people are willing to pay to execute that code on the network. Everything you said was correct.

[0:24:53.6] JM: Ethereum is of volatile currency, and one function of the gas mechanism is to stabilize the effective price of processing transactions. Can you explain how is it that gas has that stabilizing effect?

[0:25:11.6] RR: Yeah. Thanks for bringing that up. So because gas is decoupled from the market price, then it means there's this — Sort of like there is its own market for executing transactions. So if the cost — Let's say, the price of ether doubles. The same amount of gas paid with the same gas price would be double the amount in the end sort of market value. So for the same transactions, people would be paying double if that was fixed to a certain amount of ether. If the price of ether goes up, people can start bidding less ether. If the price of ether doubles, someone can submit a transaction and bid half as much ether as they would before and the benefit to the network would be about the same as it was before.

So these are — It allows it to stay independent. Naturally there're some delays, and so it's not perfectly liquid, but it does allow the market to respond. If the price of ether keeps going up, it's not going to be more and more and more expensive to run transactions. If the network responds appropriately, then the machines on the network, the nodes on the network would be willing to execute transactions for lower gas, because the value of ether is higher.

[0:26:36.9] JM: This gas is ultimately paid to the miners. These are people who are running full nodes. At any given time, there are lots of transactions that are queued for processing by the Ethereum network. How do the miners choose which of those transactions go into a block?

[0:26:59.6] RR: The miners are the ones that are getting the financial reward including transactions in the block that they mine, and of course they're all competing and trying to get the winning blocks so that they get the rewards. So the gas costs are paid by the initiator of the transaction. Whoever is submitting that transaction on to the network is bidding on that operation to be executed, and then ultimately the miner with the winning block for that block gets paid.

So the miner gets to choose what transactions they want to include in the block and what ones they don't, but this whole thing is an economic system, and so the minors are incentivized to include as many transactions as possible in a block all the way up to the gas limit. So any financially rational miner is going to — Once they find that winning block, they're going to stuff it full of as many transactions that have been submitted. There's something called the pending transaction pool, and miners are connected through the dev P2P network, which is the distributed network of the Ethereum protocol, are connected in this peer-to-peer network and they are tapped in to the pending transaction pool and they're going to try to stuff as many transactions into a block as possible, and naturally they're going to prioritize the transactions where people had bid higher, higher gas price on those transactions. In a situation where the network is flooded with transactions, then it allows market factors and market prices to effectively sort out which transactions get included. The ones that are willing to pay more are going to be included in the blocks, because the miners are incentivized to include those transactions. Does that make sense?

[0:28:50.7] JM: It does. Let's get to programming smart contracts. So Solidity smart contract code, it looks a lot like JavaScript. Solidity is the most popular language right now as of 2018 February or March. It's very approachable. You have contracts, and contracts are sort of like the classes of solidity. They're the high level objects, and you have struct for defining lower-level objects. You have functions. You have in-memory variables, and these in-memory variables only exist while the contract is executing. Then you have storage contract variables, and storage is more durable. These are variables that remain in durable storage on every single Ethereum full node.

So that's kind of the basics of the language. When you started writing Solidity, what were the biggest points of confusion when you were trying to learn this language and get acquainted with it?

[0:29:54.1] RR: Like you said, Solidity looks a lot like JavaScript. Beyond that though, it's pretty different, and I think it's kind of an unfortunate comparison. Maybe it helps new programmers feel less intimidated by it, but they're going to find out pretty quick that there's more going on under the hood that they have to be aware of in order to write smart contract.

So when I explained Solidity to people, I just tell them that it's lower-level. It's closer to the “hardware”, closer to the virtual machine, and you’re going to be doing things that are maybe more similar to C++ than to JavaScript a lot of the time. Now you don't have to do — There is no pointer manipulation, but there are a few different types of storage, which you started to mention. So that was one of the tough, kind of new things for me.

My background is largely web development, although I've worked in a variety of languages in my programming career. So some of the hardware is sort of more systems level aspects of solidity can be new for developers who don't have experience with that. Like you said, some variables can be stored in memory, which the EVM has a stack-based memory. So under the surface it's when you allocate — When you declare a variable, it's going to be allocating space on the stack for that variable.

There is also heap memory, and you can allocate memory on the heap, and that is a little more flexible. That allows you to have things like arrays with no fixed length, and you can keep pushing things on to the array because you can continue using space. Under the hood, it's going to use more space from the heap. So you have the stack, you have the heap, and then you have this thing called storage, which is what persists. You described it as durable. It persists across contract execution, and this is what makes the EVM what it is. It that makes Ethereum what it is. This is shared storage. So you basically have shared storage across the entire network, where write access is limited to either the creator of the contract or the logic that the contract will allow write operations. Read access is public. So this is something that's important to be aware of [inaudible 0:32:15.2] smart contract development on Ethereum, is that all of the bite code is visible. You can decompile contracts to various levels of readability, but ultimately the bite code is visible and any data that you store is also going to be visible. So you would have to encrypt data and put that on the block chain if you wanted the data itself to be private. Otherwise anybody can read the data from the blockchain.

[SPONSOR MESSAGE]

[0:32:50.4] JM: If you are on call and you get paged at 2 a.m., are you sure you have all the data you need at your fingertips? Are you worried that you're going to be surprised by things that you missed, errors or even security vulnerabilities because you don't have the right visibility

into your application? You shouldn't be worried. You have worked hard to build an amazing modern application for your customers. You've been worrying over the details and dotting every I and crossing every T. You deserve an analytics tool that was built to those same standards, an analytics tool that will be there for you when you needed the most.

Sumo Logic is a cloud native machine data analytics service that helps you run and secure your modern application. If you are feeling the pain of managing your own log, event and performance metrics data, check out sumologic.com/sedaily.

Even if you have your tools already, it's worth checking out Sumo Logic and seeing if you can leverage your data even more effectively with real-time dashboards and monitoring and improved observability. To improve the uptime of your application and keep your day-to-day run time more secure, check out sumologic.com/sedaily for a free 30-day trial of Sumo Logic.

Find out how Sumo Logic can improve your productivity and your application observability whenever you run your applications. That's sumologic.com/sedaily.

Thank you to Sumo Logic for being a sponsor of software Engineering Daily.

[INTERVIEW CONTINUED]

[0:34:34.9] JM: So these contracts themselves, when you write them, you deploy them, and these contracts themselves can also store their own balance of ether. Why would a contract want to store a balance of ether?

[0:34:51.2] RR: Great question. In Ethereum, there's actually no functional difference between a regular user address that is like an account that's storing ether and a contract address. So every contract is deployed to the network and it lives at a given address, and then ether can be sent to that address. There's something called a fallback function, which controls whether ether can be sent or if there's any operations. Think of it like an event handler for people sending money to a contract. Certainly, there might be cases where you wouldn't want people to send money to a contract if it's not built for that. So the fallback function allows you to handle the event of people

sending money to the contracts in ether or simply disallowing that. Having contracts be able to store ether, it really what makes smart contracts so powerful as an economic tool.

Contracts can hold ether and that allows you to build financial tools. You can build simple things like escrow, where two parties — A party would put in money and it might be released to a second, a counterparty at a separate time, and there is a third party that functions as an arbiter, and the smart contract logic controls who has access to the money when. So instead of relying on, first, things like this, instead of relying on protocols in the real-world, in the material world and legal designations, instead you can actually control the flow of money and control the flow of execution of operations on data in this shared virtual computer with smart contract logic.

So escrows are one example. Another example would be the Prism system that I helped build. You have a portfolio of different cryptocurrencies. What's actually happening? Well, there is a smart contract that is storing ether that is equivalent to the value of your portfolio. So if you have a little bit of Bitcoin, a little bit Z Cash, a little bit of Litecoin, the market price of those coins will go up and down, but the smart contract stores given amount of ether as collateral, and that will stay equivalent to the value of that portfolio, or rather you're entitled to a given amount of that. That wouldn't be possible if the contract couldn't store the ether itself. Otherwise you'd have to trust a third party, and the whole point of the blockchain and these contracts is to dis-intermediate peer-to-peer transactions and use math, use cryptography, use programming to enforce trust at key points. Then the parties that are involved, instead of having to rely on that third party to be trustworthy or to be secure, you can audit the code and you can know the code is going to execute and control the flow of money exactly as designed.

[0:37:49.9] JM: Let's say I write a contract in Solidity. I finish writing it. I compile it to byte code so that it can run on the EVM. When someone wants to call my contract, they call it via the ABI, which is the application binary interface. Explain what an ABI is.

[0:38:10.7] RR: Sure. So if you've deployed a contract, then it's associated with a given address. That's where the contract lives, and the byte code for that contract has been copied to every single node on the network. So every node on that network knows how to execute that code. So how does someone actually call a function on that contract? Like you said, there's an ABI, an application binary interface, which basically shares the access points, the functions that

can be called, the public functions on this contract, which consists of a function signature that's been hashed and they take a certain number of bytes from the beginning of that function signature. So then you have unique hashes for each function. What you do is you wouldn't be doing this manually, but if you use your wallet or you use any kind of application that has the ability to communicate with smart contracts to call functions on them, what's happening under the surface is that a normal Ethereum transaction is sent to the address of the contract and it includes a little chunk of data that is that hashed function signature. It's built-in to the EVM to recognize.

When a transaction comes through and it includes data, you use that data to look up which function to call and then execute that function. That data could also include parameters. So you have inputs provided. So it's interesting that in Ethereum, sending transactions, the same transaction could be used to just send money, to send ether from one address to another, and it's really the same format. It's just a protocol that says, “If you send a transaction with a little bit of data attached to a contract address, then the EVM is going to respond by looking up the function to execute,” and then executing it in that shared environment.

[0:40:08.2] JM: If we take the example of CryptoKitties. Is CryptoKitties a set of contracts or is it one specific contract?

[0:40:16.0] RR: I’m not super familiar with architecture of CryptoKitties. Let’s say, theoretically, it’s a single contract. It’s possible to have contracts that spun off other contracts, but that can be quite expensive. So the way that I architect smart contract systems these days is I try to contain as much as possible in a single deployable contract, and even if there's multiple parties involved or multiple objects or entities being created, you have arrays, you have structs, you have objects within the language. So you can contain all that in a single smart contract.

So CryptoKitties, I would have to look at the code, but theoretically it's a single smart contract that has different functions that are available to transfer a kitty from one owner to another, to breed kitties, to do whatever the different operations are within that game.

[0:41:07.1] JM: All right. Well, let's assume we've written CryptoKitties. We've got the contract. We want to deploy it. What is involved in deploying it?

[0:41:15.2] RR: So deploying the contract, like I said before, the concept is because this is a distributed network, you want every node in that network to have a copy of the byte code so that anybody can call functions on the smart contract, or you can limit who calls functions on it, but at least the bite code is out there so that anybody can initiate the function call and then the contract itself will authorize and either accept or reject that. But the byte code needs to be put out there on the network, and that's called deployment, and the way that works, similarly, it's also a transaction. However, it's a transaction sent to a specific address.

I might be wrong with some of the details here, so excuse the fuzziness, but the general idea is that there is either a special address or a special code that the EVM understands to know that, "Hey, this transaction is not just sending ether. This transaction is not just calling a function. This transaction is deploying a contract," and the byte code for that contract is included in the data field, just like the inputs, the input parameters or the hashed function signature is in the data field when you're calling a function. The entire contract byte code is included in the data when you're first deploying that function. So, consequently, deploying functions can be quite expensive, but hopefully you only have to do that once. You pay the cost to have everybody in the network. You incentivize the network to take this byte code that you've sent out on your transaction and assign it a unique address and then it's living on the blockchain where anybody can invoke functions.

[0:42:57.2] JM: Okay. So there's a set of tools that we're going to want to use in smart contract development. So a couple tools that come to mind are Web3 and Mist. So let's talk about Web3 GIS. Web3 GIS is the Ethereum JavaScript API. If I — Developing an Ethereum smart contract, why would somebody want to interact with that contract through JavaScript?

[0:43:24.0] RR: So there's a distinction here between the code that's running in the contract. People refer to that as on-chain, and the code that's running on a normal web server on a machine, online or off-line, but there's code that's interacting with a smart contract, calling functions on a smart contract, and that code is off-chain. So the off-chain code can initiate transactions, can read from the blockchain, then the on-chain code is that EVM byte code that's being executed.

So off-chain, you can use any language you want. The interface is JSON interface that when you submit transactions, it encodes data in a specific format. So it's protocol. You can implement that protocol in any language. So Web3.js is the JavaScript implementation of that JSON RPC protocol that allows you to interact with a local Ethereum node. It allows you to read data, submit transactions, all of that.

JavaScript is a pretty ubiquitous language, and so naturally a lot of people will be comfortable using JavaScript. Even if they're not writing smart contracts themselves, they can install Web3 with node and NPM. and write a normal JavaScript script or application that uses Web3 to communicate with an Ethereum node. There're similar libraries in other languages like Elm, and Java, and Python and who knows how many, but that is the off-chain code. The interface with the Ethereum node can happen in any language, and Web3 is the one for web developers and JavaScript developers.

[0:45:12.6] JM: Awesome. Mist is a browser that we can use to browse and use smart contracts. How would I interact with Mist as a user?

[0:45:22.6] RR: Yeah. mist is a browser. It renders HTML just like a normal browser, but it has an interface with a built-in wallet so that web applications can use Web3 to have a smart contract functionality integrated into the application. This could be a whole other conversation, but the reason this is important is that it's a completely different paradigm from the server client relationship. So, in a typical web world, the conventional web world, we're really use to client/server architecture and the browser is the client and the server is running on some hardware somewhere on the internet, and we are trusting that server. We're trusting that server with our data and we're trusting that our data gets there securely and we're trusting that the server code is going to be executed as expected. But there's a lot that can go wrong in that process, and the main benefit of Ethereum is that we can write applications that have fully auditable code that is guaranteed to be executed as expected and can't be interfered with in a way that client/server architecture could be.

So Mist is a browser that is developed in order to allow for that, what we could call client-client architecture or peer-to-peer architecture. You're interacting from your client with the smart contract. It's not a server controlled by a company or organization, but it's a smart contracts that

has predefined rules that are executed based on the byte code that was deployed, and anybody can audit. So the Mist is a special browser that allows you to interact with the web application where you can send and receive ether, you could sign transactions, and you basically prove your identity through signing transactions with your private key, and that's built into your wallet, and Mist is fully integrated with that.

Another really popular mechanism is called MetaMask, and it's a browser extension. So instead of downloading an entire new browser like Mist, you could just download the browser extension, like the Chrome extension or Firefox extension, MetaMask, and it has the same kind of capabilities where if you access a web application that's been optimized for Ethereum or rather that has actually been built with those integrations involved, it's commonly called a DAP, or a DIAP, distributed application, then you can securely interact knowing that the web application can't steal your information or can't steal money from your wallet without your approval, and MetaMask or Mist provides that secure intermediate layer where anything that the web application does involving money or involving transactions or signatures, it's going to prompt you and provide hopefully an intuitive and easy to understand interface that warns you sort of like the warnings that browsers have for you today when you're about to reveal your location or when you're about to turn on notifications.

Similarly, MetaMask and Mist will give you a warning and say, "Hey! This website is integrated with Ethereum, with Web3, and it's requesting that you sign this transaction or that you send a little bit of ether in order to interact."

[0:48:50.7] JM: And in both of those cases, my browser is essentially all light client. Is that right?

[0:48:57.0] RR: Yeah. A full node requires downloading the entire, which is massive amount of data, many gigabytes and growing. Mist is a full client. So when you download Mist, it's going to sync with the network, and there are some optimizations that it has so that it doesn't have to. It'll just download all the hashes of the data so that it's cryptographically secure. They're basically checksums of the entire network and all the data. So you don't have to download as much data, but Mist is a full node, and then MetaMask is a light node, which means that it doesn't download all the data. So you don't technically have a full validation of the authenticity of that data, but

you're relying on other people in the network saying, “Yes. If your block starts with this hash, then that proves cryptographically that the data hasn't been tampered with, and you can trust your peers.” So light node are a slightly lower form of security, but as long as the rest of the network is being secured by full nodes, then you’re able to transact safely knowing that the data you're using has been secured through other nodes.

[0:50:13.6] JM: Okay. Raine Revere, thanks for coming on Software Engineering Daily. It's been great to have you.

[0:50:17.4] RR: Thanks so much. It’s been great to be here.

[END OF INTERVIEW]

[0:50:22.1] JM: If you enjoy Software Engineering Daily, consider becoming a paid subscriber. Subscribers get access to premium episodes as well as ad free content. The premium episodes will be released roughly once a month and they’ll be similar to our recent episode, The Gravity of Kubernetes. If you like that format, subscribe to hear more in the future. We’ve also taken all 650+ of our episodes. We’ve copied them and removed the ads, so paid subscribers will not hear advertisements if they listen to the podcast using the Software Engineering Daily app.

To support the show through your subscription, go to softwaredaily.com and click subscribe. Softwaredaily.com is our new platform that the community has been building in the open. We’d love for you to check it out even if you’re not subscribing, but if you are subscribing, you can also listen to premium content and the ad free episodes on the website if you’re a paid subscriber. So you can use softwaredaily.com for that.

Whether you pay to subscribe or not, just by listening to the show, you are supporting us. You could also tweet about us or write about us on Facebook or something. That’s also additional ways to support us. Again, the minimal amount of supporting us by just listening is just fine with us.

Thank you.

[END]

