EPISODE 1403

[INTRODUCTION]

[00:00:00] KP: Serious software projects require several environments. Your production environment is obviously mission critical. A staging environment is also necessary to perform validation and regression testing before taking the risk of pushing an update to production. Best practices and approaches for managing these and other environments vary from organization to organization. In some sense, different software system should be expected to have unique needs. Yet certain commonalities and wisdom can be gained from observing high scale success stories. Towards that end, in this episode, I interview Senthil Padmanabhan, about how eBay turned around an impeding staging environment into its biggest asset for developer productivity.

[INTERVIEW]

[00:00:43] KP: Well, Senthil welcome to Software Engineering Daily.

[00:00:47] SP: Thank you, Kyle. It's great to be here.

[00:00:49] KP: Where does your journey into software engineering begin?

[00:00:53] SP: Oh, it began right out of my undergrad. After college, I started working in software service industry. Actually, it began much, much earlier right when I was in sixth grade, I started basic programming language when my mom put me in a class. From that time, I started getting hooked up into this. In fact, even in my undergrad projects, I used to build websites and that was the one thing that I really knew to do well, and I used to think if I don't have a job in software engineering, probably I'll do that as a hobby. I feel very fortunate that I'm doing something that I that I love to do, and I'm getting paid to do that.

[00:01:28] KP: Well, you've been around software engineering and code, then for a long time and doing it at least at the hobby level, probably long before you knew that there were these things called environments. When did that concept first become a part of your process?

[00:01:40] SP: I think when I joined an organization, like a big organization like eBay, or before that Yahoo, this concept of environments became known to me, but I was a junior developer at that time, but everything was already there for us to play around. But I also understood one thing is that like any implement in that environment has this multiplier effect, because that's the software that takes your software to your customers. It's like a mini operating system, which is for your application software to go and reach your customer. Any improvement you do there, it has this ripple effect of having a huge impact on developer productivity. While I was working in the early days, I was not working on environment, I was working on application space. Then I used to take it for granted when things all worked well. But when things don't go well, is when I found out how much of a pain it cost for developers, and it's been a big frustration point.

I started appreciating the value that environments provide. As I went through in my career, and I'm at a point where I **[inaudible 00:02:38]** developer, staging environment for eBay, the eBay marketplace. That was one of my primary goals, is to make sure that I unblock developers. To me being blocked is a worse outcome than being wrong. Because being blocked means you're not doing anything, you're just waiting on thing and that's just unacceptable. That's one of my primary principles on how I look at environments.

[00:02:38] KP: I think most people are going to agree on the definition of the production environment. Is there a somewhat universally accepted definition of what the staging environment is?

[00:03:10] SP: I think so, yes. I think in the software world, just like how staging applies in outside of the software world, like in an act or a play, or in real estate, right? It is the platform where you put something where you deploy something before it goes to production. I think that's a sort of a high-level definition of what a staging environment would mean. But having said that, it becomes subject to when we go into the details, because the environment can be many things starting from your app server, from your application services, all the way to your databases, along with all the other auxiliary systems that are required for a system to be up from your ETL jobs like Kafka, your pipelines and things.

Various companies have different definitions, where they just host some of the subsystems in a different environment and just leverage the production environment for the auxiliary systems. It depends, like you ask every organization, they may come up with a definition. But the overall idea is that, it is something that your end customers do not see. It is something that is for the developers to confidently deploy software before it reaches your actual users.

[00:04:17] KP: And do organizations need a single gatekeeper who manages that or is it something that gets distributed across teams?

[00:04:24] SP: The answer is, it depends. But what I have found out in my experience, is having multiple environments to manage eventually becomes a nightmare, both for the management perspective, also from an application developer perspective. Having less environments is actually better. The more automated they are, it's even more better. My preference would be to have one environment called the staging, which is basically a replica of production, but you just have your next version of your software, the Nplus1 version of your software constantly deployed, and you tested with all the other things that are there in your system. So that when it goes to production, you know it's working. My preference would be to have one environment called staging for the whole organization where you can confidently deploy.

[00:05:08] KP: What do we need beyond that, especially in a scenario where maybe two disparate teams are collaborating on the interface between their two mutual software systems.

[00:05:17] SP: I think in that case is when the things, you got to more towards the left or right. If you think about a software release pipeline, it starts from your source code. That's where your code repository, like GitHub, GitLab, they are there and then – so once you do your coding, you check in it, you can then do a lot of unit testing around it, where you don't depend on anybody else. It's just that you're testing a very small unit of your software. And then comes a little bit more than that, it's where the functional testing comes in where you can test your interface as you pointed out, and that's possible if you do a contract-based testing, where you clearly define your schemas, and you test your endpoints and make sure your schemas are all working well and good.

Then you come to staging where you just do integration testing. You need all these things before staging itself, where you have a good unit testing environment, where you build the test pyramid starting with unit testing, you write a lot of unit tests, and then you do this functional or competent level testing, and that you leverage techniques like contract-based testing. And then you come to an environment like staging where you do your full integration testing. Having said that, we are also looking into ideas where if you have a really good solid staging environment, you can also leverage that one environment by making you do stuff which is isolated to your dev environment, meaning, instead of having an environment isolation itself, you can have a request isolation where every request has a special headers associated with it, which instead of taking you to a different environment, it just takes you to a local dev box or your local testing area and you can validate them.

[00:06:50] KP: You've shared the wisdom that less is better and I tend to agree with that. With that minimalism in mind, can we be sure we actually need staging?

[00:06:59] SP: That's again a great question. In fact, a blog that I published recently Staging Dichotomy, I tried to answer that question, because – just to give some history, eBay has been having a staging environment starting the early days. But in the recent few years, it has reached a state that it was not be – it was not usable, it became a very big frustration point for developers. And when we did a developer survey that was highlighted as one of the biggest pain points. It was because like in 2019, the environment was very broken, it was very unstable and nobody maintained that functionality. We had two options, either to go and fix staging, or we needed to forget about staging and go to production directly to test it, because we do continuous deployments, just only tiny delta that goes to production every day. And why not, we just leverage this one well-built production environment.

But again, context matters here. We figured out that it's not – it's easier said than done. When production testing is okay for readerly use cases. But when you have right use cases, and when you are an e-commerce company like eBay, where payments is involved, where a lot of transactions are involved with real buyers, and sellers and real items, that gets tricky. We do not want to have a scenario where we – there is no margin of error allowed in that sort of a scenario where you can play around with payments. Even in read-only use cases, it becomes problem when we run like thousands of test cases, because you start polluting the metrics behind it. You can add a dimension call, it's a test metric. But even then, that has to proliferate through the whole pipeline and that becomes another problem. At least in the context of eBay, and I think many other companies would need an environment where engineers can safely do mutations, and deploy the **[inaudible 00:08:37]**. They should be confident to write whatever integration test they want, rather than having any limitation. The answer is, it may work for some companies, but not every organization.

[00:08:46] KP: Well, I'm curious what happened to the staging system as it was then. Do you sunset it with gratitude for the service well done to that point and start fresh or is it something you overhaul?

[00:08:57] SP: What we did was, when we started looking into it, right, we decided that, okay, the question of whether we need a staging or not. We decided we need a staging. The next step was, okay, what do we do with it? Like, either, as you said, do you scrap the whole environment and start something fresh or should we just go and fix the things that are broken there? That's when we did an exercise, which was just not a technology exercise. It was more about an exercise on our philosophy on how we do big initiatives. That was to be very specific as possible, because people like to make generic statements because they are easy to make. One such statement that was floating around in eBay was, "Oh! Staging is broken." That's true, because it's reflected everywhere. But that doesn't let us know what to work on, because staging is broken, it's a pretty useless statement, because it doesn't tell us what to do.

First step in our journey is we went to this statement and we tried to make it as specific as possible. We partnered with one of the domains and we started analyzing what two engineers mean when they say staging is broken. That was – basically, it was three things that we found out. One was, the staging data was not usable. It was because it became very processed over a period of time when people were just abusing staging and it doesn't return you any solid information. It has like millions and millions of items, but none of them had images, or all of them had the same title, which was pretty useless. We had to fix the data. The second thing was staging environment was not stable. But more than not stable, it was not predictable. I think that was the biggest annoyance, because sometimes it used to work and sometimes it doesn't work. Sometimes the builds take five minutes, sometimes it takes less than 30 seconds and so that predictable performance was lacking.

The last was more about a cultural mindset problem where, since engineers knew that their environment was not stable, they were not maintaining functionality. Since the functionality was not met, the infra engineers were not monitoring, staging and giving the love it needs. If you fix the first two, probably the third will be fixed. We found out that, okay, we don't have to take away everything, we have a system in place. Let's go and attack each one of them individually, and we attack them and that became the new staging environment as you see today and it has had a huge impact on developer productivity in the past **[inaudible 00:11:14].**

[00:11:15] KP: When it comes to data, I can understand where you'd end up in a place where staging environment has just cruft the result of different integration tests and things like that. If you want to provide a more production- like experience, what sort of options do you have?

[00:11:31] SP: We looked at various options here, the first option, we look to us, like, "Why don't we go and create the data? I mean, that seems to be the most obvious thing, right? Okay. You need this quality data, why don't you go and create it." But again, that is very tricky, in a sense that it is okay to create a lot of data, which looks the same. But if you want to create items, in our case, in any base case, its products and items, which has these 20 different skews with each skew having a valid image of, like let's say, you have a shirt with many different colors, each having a different valid image. This combination of each having a resale price, and then with the different shipping options and a return policy. This combination was just practically not possible for our engineers to create because there are hundreds and thousands of test cases.

I mean, we need proper backing in order to do that, and there were no proper APIs exposed, because the APIs that we had, were more business driven APIs and not test case driven APIs. The solution we came up for that, in hindsight, looks pretty obvious is like, take a very tiny subset of our production data, which has high quality because they are real listings from real sellers and buyers, and move to staging environment in a very privacy preserving way. So that you get the high quality, but you still preserve privacy so you get the best of both worlds. Unfortunately, for us, listings are public information, because products are public. The only thing we had to anonymize was the buyer seller and some other aspects of an item, which we implemented in a pipeline and that's what is effective today at eBay.

[00:13:01] KP: That addresses the data in some very nice fashions. You'd also mentioned that the instability and unpredictability of the environment was a challenge. How do you improve upon that?

[00:13:11] SP: What we did there was, again, we went next level deep into being specific on what it means that we had an unstable environment or an infrastructure. It was basically on the hardware part of things, that's what most of the things were done. And also, on the monitoring and remediation part. The first thing is, the staging hardware was not getting updated in the way production hardware was getting updated. For example, our load balancers were a little outdated, right? And one load balancer was new, but the other one was outdated, something like. So combination where everything was not in the state it has to be. We went in each layer, starting with a load balancer and fix the old ones and the new ones. Then, we also did an exercise of cleaning up, freeing up spaces over there. Every load balancer works in the same way it's supposed to work.

Then we also went and created a new availability zone for staging, where we had new hire clusters deployed. So that staging also has the same failover like in production, whether it's a disaster recovery, traffic goes from one zone to another zone. And then wherever the user is close to, then it goes to that particular zone, so we did that. The other thing we did was the database, the database hardware again, was too much outdated. We did a full overhaul of the database where we moved into new hardware with the new instances of our Oracle DB, which is our predominant database along with all our NoSQL offerings. Once we did everything, we saw immediate impact in this predictability. Every request took the same reasonable time.

Then we did another thing was, the monitoring and the support system was not in part of production. We had a lot of state-of-the-art monitoring tools for production. Unfortunately, due to a lot of firewalls and things, those tools were not able to connect to our staging environment and monitor the systems like CPU, how your database is doing and things. We fix that, we fix those monitoring tools. Now, the same interface that we use for production can also be used to monitor staging with just flipping a switch, saying, "Hey! I want to monitor staging and production." And then we also have dedicated videos like technical duty officers, just like in production for staging. I mean, in a smaller scale, who continuously monitor them and act upon them in a way that these things are all always up and running. That's how we sort of address the environment problem, and every step, these are improvements. The data and the environment was ready, so the only thing we had to tackle was the last one, which is solving the commitment and the cultural mindset problem.

[00:15:34] KP: What were the biggest quick wins or low hanging fruit achieved initially?

[00:15:39] SP: For the data, biggest thing was that developers have very delighted that they don't have to worry about a new project that's coming up when we're testing a particular scenario before it goes to production. Because when the write integration test case is, there was a lot of hesitancy in the earlier days, because they have to first go and create this data and that was a nightmare for them. They were writing less integration tests, and they were trying to rely more on production to do the testing, which is not the right way to do because in production, you are again, your hands on tied on doing mutations.

As soon as we move the data, engineers started writing integration test cases, in a way that they're supposed to, right? They were very confident in writing integration test cases. That was an immediate quick when we saw that the count of integration test started going up. The second immediate thing we saw was also the pass rates, because the flaky tests had a big problem in our whole industry, right? Because the false positives over a period of time dilutes the value of a system. We want to avoid that as much as possible. So with the stable data there and the stable infrastructure in place, the false positive started reducing. I think these were the two quick wins that I would highlight, which really helped us in terms of developer productivity in like **[inaudible 00:16:51]** when we started doing the project itself.

[00:16:54] KP: eBay's production system must have some interesting dynamics that you don't see it every place, given its scale, and what the site may do. For bidding on things that can go down to the wire, there's probably problems that companies like eBay, and maybe only Ticketmaster and a handful of other people have to face. Do those unique challenges impact the way you have to think about testing or set up environments?

[00:17:16] SP: That's true. I think it's a very valid question. I think the auction mechanism in eBay is a very unique system in the world, where we have to tackle all these efficiencies. When we did the staging implementation on the infrastructure side, and on the application side, this was one of our criteria for measurement on how much parity we have with production. We had to run these test cases, numerous times to make sure that we have reached that level of parity and only when we reached and we'll get that confidence as we sign off and saying, "Okay! We are done with this particular area or with this particular domain." And again, that's the past rate, which I told you also reflects on the scenarios, when test like these in the past used to fail, and when developers go and see. It used to be a false positive, it was like – it's actually a scenario, which works fine. But due to some stability and data issues, it was failing. Or due to this firewall issues, were these – even a millisecond, difference matters. We can really – but once we started addressing these things, the pass rate improved dramatically and we are able to tackle those unique challenges, which are very particular to a company like eBay. We put some extra focus, I would say on those scenarios in the early days, which has helped us also make great progress when we were able – went towards the end of the initiative.

[00:18:27] KP: When I began my career in software, there was a lot of ceremony around releases. We might even take a downtime. There's been much more of a trend towards continuous integration, continuous deployment and these sorts of approaches. You can move really fast, maybe some companies move a little too fast. Is there a general philosophy or approach you have to the release cycle?

[00:18:47] SP: The way I look at release cycle is that as an engineer, the primary responsibility of an engineer should be to check in code, and then one, and then submit a pull request. Meaning, to verify the code and make sure it follows the standards. Then another human, another engineer reviews it, one or two engineers, and then they approve it, right? Once approved, it has to go – the system should take care of deployment. My philosophy here is that deployment should be behind the scenes. An engineer should not be worrying about a deployment when it happens, or whether even if it happens or not. They should be more focusing on developing applications, and checking in code. Once that PR is upload, it's the system which takes care of taking it from – there should be no human involved from the time your PR is accepted to the way it deploys. It can happen one time, it can happen 100 times in a day, it should not matter. It should be something like breathing, I say to people like, you don't think about it, but it's the most essential thing that you have to do.

You should not have that thought in your mind. When is this code going to go to production? I mean, that's like the North Star goal that we are there. Some teams in eBay are doing it, not everybody. But that's my principle on how deployment should work. Because that becomes just an artifact of your development which doesn't need any humans involved. I remember the days when deployment used to happen, like we used to hold on things, then we used to roll out, keep monitoring, every developer used to – see the metrics and things like that. I don't think it's scalable in this new world without hyper competition where features get rolled out so quickly in a very good quality fashion. I would say deployment should be behind the scenes, that developer should not be thinking about it.

[00:20:25] KP: Well, having a staging environment with a lot of parody towards the comparison to production has some obvious benefits for developers, like we've mentioned, probably also QA engineers will be happy. Are there any other beneficiaries, could perhaps a data scientist or machine learning engineer see some benefits from this process?

[00:20:45] SP: Yes, absolutely. I think the data scientists, to some extent, can use a lot of the data that we have migrated to do more mutations on it and try to do some training on these environments. But we found another side effect because of this is like, since we have this data available for us, we also gave it to our design team to start building these new mock ups and UI using this data that we migrated. Because usually, what happens in a traditional system is that, designers when they develop screenshots, we're sharing with executers or when they come up with these novel ideas. They use stock images and photos, but they are not a representation of the real world. But now, we have these anonymized privacy preserving data, which is a representation of real world, but without knowing who the users are. And now, you can come up with a sort of realistic prototypes. You really know how a product looks like with real data, because sometimes in the past, we have been in situations where the markups look beautiful, and the prototypes look beautiful. But when the product launches, it may not be exactly what you were thinking when you originally saw the mock up.

That was something that we started doing with the staging data, where you start building all these new interfaces with this data, which is more realistic representation of how the product will eventually look like. That helped us a lot in tweaking mechanisms much earlier in a product lifecycle.

[00:22:07] KP: Do you have any interest in synthetic data?

[00:22:10] SP: Yes, I think synthetic data is still required, because not every use case would be covered by this data migration use case of taking data from production to staging. We still need to create, especially in user scenarios, a lot of sign in related use cases or registration use cases that the basic scenario itself is to create the data. In those scenarios, we have another utility that we revamped as part of this process, which is to have, again, very efficient APIs to go and create data in a way that developers would have to create, not in a way for a business use case or like an entity service. It's more like, Okay, I need to create an item with 40 skews with 20 images. This is not how our real business use case works.

We have created that platform also, because that will only cover the whole spectrum of our test case needs.

[00:23:00] KP: We've outlined a number of ways in which development and productivity could be advanced. What about triaging, having a closer environment does that maybe give you less production incidents in some measurable way or a faster time to find things. What sort of benefits you get from that perspective?

[00:23:17] SP: There are couple of things I can highlight here, is that, whenever an incident happens in production now, it has become very easy for an engineer to say which item that is associated with, and immediately use the pipeline that we're able to get that item to staging. In the past, it was again a very difficult problem to deal with. Whenever an issue happens, we can immediately get the item in an anonymous way to staging and then developers can quickly use that to play around, deploy the new code and it goes to production. It has been a big-time saver for us. A lot of developers have been extremely happy on how this utility, I mean – this was – we never intended this as one of the use cases when we started. But this became a very good mechanism for us to triage and get things out. Another thing that we are also seeing in this whole system is that, since we are writing more and more integration test cases on staging in a confident way, we asked – when the code goes to production, it is –most of the time, it is thoroughly tested end to end. Because of this micro service architecture that a lot of the modern companies have, you are dependent on many, many services. Sometimes when you don't write confident integration test cases, you're just praying and deploying your code, hoping that everything would work fine. And if it doesn't work, it becomes a bigger cycle of fixing it.

Now, we have started emphasizing teams to code and write more and more test cases. I mean, focus on unit testing and then build the test pyramid. But don't hesitate to write integration test cases that will make the system fail. If it fails at staging, it's much easier to fix and that has also resulted in our production bills being more stable down the line.

[00:24:54] KP: Well, there's a path, somewhat unfortunate path. I see a lot of companies walk where they start out with a few basic tests, things are going well, they have a nice integration system. And over time, those tests are taking longer and longer until developers start skipping them more and more often. And then suddenly, they never pass anymore. In some sense, I could think you could say some of the efforts you put in were a level setting, getting things back to a usable state. Do you have any concerns that the garden will grow out of control on its own if you don't come and maintain it the right ways?

[00:25:25] SP: Yes, I had the concern from day one when we started this effort. In fact, I say this to my teams. I think fighting off redirection to the mean is one of the toughest things to do. It is very hard to build a good system, but it is even more harder to keep that system great. Because as you just mentioned, Kyle, it's very easy to start regressing, right? Because we have put a good system in place, but things can be – I mean, just laugh entropy, things just keep adding more and more randomness or more and more things get accumulated and you start regressing. Any good system, any successful system. If you see any successful projects, they have put this system in place where it fights regression, even tiny regressions. That's something that we took very seriously when we did this effort, because staging was good 10 years back, or whenever it was at eBay and everybody had good intentions.

But for a period of time, nobody in particular but collectively ended up in a bad state. The same thing can happen three years down the line, you're now in a good state. But as you said, things accumulate and it can go to a bad state. So we put a couple of things into place, in order to make sure that things – that regression does not happen. The one thing that we did was, we built a system called smoke, which generates traffic to staging 24 by seven, by executing integration test, P1 and P2 integration test cases every 15 minutes. Those test cases have to run within five minutes. If it doesn't, it immediately sends a pager alert to the domain. We keep the domains in check on doing it. Initially, it was very frustrating, but then domains got used to this. The question of how I write big integration test case. I'd write like very lengthy or like time consuming integration test cases that are not possible in the system now. Because if you do that, you will not be allowed to deploy code because your smoke test would start failing, and you have to address it right away. Humans will not be the right mechanism to keep guard of these things, so we have to put the system in place. So that's one thing we did.

The second thing is also, we assigned an availability goal for all domains to be at 99% in staging. In production, it can be 99.9598, things like that. In staging, we wanted to keep 99%, which means they have every application has only seven hours of downtime every month. We put the whole system there for them to – with serious technical UD officers. Whenever that test fail, they get pager alerts, but they also have a mechanism and help mechanism where they can go and trash the problem and deploy good code. With all this in place where we have goal, availability goal, as well as a system which generates traffic 24 by seven, just like in production, we are confident that this regression will not happen and the system will only become better going forward. We are thinking that it creates a virtuous cycle of three blocks. One is this availability goal. Since you have an availability goal, it puts pressure on the infra team to make sure that they have a stable infra and in order for that to make you a good data. And if you have good data, your availability further increases, so I think it creates a good cycle for us that we can fight off this regression.

[00:28:28] KP: Well, it's often not easy to get resources for a project like this to go in and evaluate a system that's experiencing some challenging, and growing pains, and level set and come up with a new approach. Can you talk about the organizational steps to get resources allocated and make the decisions?

[00:28:45] SP: Yes, that's a very important point. In fact, a lot of times the problem is not with the technology itself, but it's with alignment. An initiative at the scale, it needs alignment, both the top down as well as bottom up, because we need proper funding. It's a very large initiative. So you need leadership support. Also, we need bottoms-up approach, because engineers should organically feel that this would make a difference to their workflow. That was what we spent the most of our initial basis with – the developer survey that we did really helped get the attention of our senior executives from our CTO and our CPO, that they understood that this has to be addressed. Once we got that support, that made our life easier.

I think this is a general suggestion that I would give to anybody who wants to do big initiatives in a company is to make a good case so that you get buy-in from senior leadership, but then also understand what happens at the ground level so that you have this top down as well as bottoms up alignment. That is very critical when you start an initiative. So in my case, as I told you in this initiative, the survey really helped us to set a very good story. We are able to build a very good story with a senior executives saying that this is the biggest frustration and then they were aligned. Once that alignment happened, the next job was to go and work with the engineering crowd and make sure that what we are doing is the right thing to do.

We in fact started the initiative itself only after we got this alignment on both sides, and **[inaudible 00:30:07]** our CEO was very particular about one thing called velocity. He was very pedantic on this point that release, like velocity is very important for a company. When you have the support from the CEO itself, who is all about like faster deployments, good developer environment and developer productivity, it makes your life even more easier. Fortunately, for this initiative, we had the support from the top level, which helped us navigate a lot of organizational challenges and put the team in place.

[00:30:35] KP: Are there any metrics or KPIs you can point to that demonstrate the success of the efforts?

[00:30:43] SP: Yes, I think I have. There are a couple of things I can say for example, today, 90% of our integration testing happens in staging. In the past, it used to be very random, sporadic, like some domains used to have 40%, some use to have 80%, 60%. We didn't even know the numbers for some domains. Today, across the board, we have about 90% of integration tests run in our staging environment. Our pass rate is at 95%. It was 70% last year when somebody does our testing and staging. Every person here in past state has a multiplier effect, because of the time you save of developers going and looking into this flaky test. That is a big win. Also, I would say a lot of the sanity testing, developers used to offload it to production and just wait and see what happens. Now, very, very small amounts of sanity testing happens in production, and mostly read-only use cases.

Another bigger change I would say is our native apps. iOS and Android apps used to be a three-week release cycle in the past. Now it is a weekly cycle. I mean, there are many things that contributed to it, but having a proper staging environment was one of the key reasons we are able to achieve this weekly releases now. Our customers will be able to get features more quicker, bug fix can go to our customers more quicker, especially in the native app world where we don't have this control of deploying in the cloud and users immediately see. I think these are some of the things that I can point out.

[00:32:05] KP: I don't know if this effort now goes into maintenance mode or if you have some vision for other milestones to hit. What's the future look like?

[00:32:11] SP: No, no, it will not come in. As I told you, right, our goal is to fight off regressions. If you are to fight off, that needs to be a dedicated team that always works on this. Just like how in production, right? We have a full dedicated team that works in staging. Our first goal is to allow zero regression and our operational efficiency. Our availability number currently – I mean, I didn't cover that in your last question. For example, our availability on staging was like 55% in early 2020 and now it's close to 96%. We wanted to go to 99%, so we still have to put efforts. We are still working on some parity gaps in staging, because new things keep coming, new features come. We have a very exhaustive roadmap that we have to cover on features, and we need the data for those to be created. Our roadmap is pretty full for staging and it will be going indefinitely, because as I told you, like if you lose track of it, then you start regressing. We want to make sure that we put the building block in place, we also put a system to fight regression. Now is this, the engineering people to keep that system up and running and that's what we're going to do.

[00:33:12] KP: I'm wondering if you have any advice for any companies that are taking a hard look at their technical debt and assessing if they should engage on something like the project you did, and really formalize their environments in better ways. What are some key indicators to help me make a decision like that?

[00:33:29] SP: One thing I would say is that, a lot of organization look at more of the peripheral layers, on outer layers of application space and don't go deep into the platform, which is behind it. I mean, even if you are in a cloud-based environment, you still have an abstraction layer that you need to build upon, in which your other software gets built. I would say that software, which is your abstraction layer, or your platform, or whatever is it should get more attention or should get more allocation in terms of developers, in terms of engineering allocation in order to make it work very well. As I told you, that has the multiplier effect that the other stacks doesn't have. Because application layer is what your users see, it's what probably your senior executives see. But that is just the outer story, but it's all built upon this stack behind it. I would suggest organization to take a harder look at that stack and see whether it is doing the right thing or not always. It's very fragile. And they suggest the applications are covering it up. Because in the long run, you will not be able to sustain a very unstable infrastructure. Your whole application will start crumbling.

You may not see the benefits now, but once you start investing in that layer, your feature development becomes quicker and quicker and it becomes much, much more easier. You will start spending much time and less time in feature development when you actually go and fix the layer below it. That would be my suggestion. Take a hard look at your platform, which is actually powering your software that's close to your customers and invest more in making sure that that is in the state it has to be and that's more stable and robust.

[00:35:01] KP: Good advice. Well, Senthil, thank you so much for coming on Software Engineering Daily.

[00:35:07] SP: Oh! Thank you, Kyle. The pleasure is mine.

[END]