Capgemini

CR017

Modernising software systems with John Kodumal, LaunchDarkly

CR017

Modernising software systems with John Kodumal, LaunchDarkly

Disclaimer: Please be aware that this transcript from the Cloud Realities podcast has been automatically generated, so errors may occur.

[00:00:00] Yes, I hope it is good because otherwise I recommended a book that I haven't read yet.

Welcome to Cloud Realities, a conversation show exploring the practical and exciting alternate realities that can be unleashed through cloud driven transformation. I'm David Chapman. I'm Sjoukje Zaal, and I'm Rob Kernahan, and this week we're going to be talking about the evolution of software systems.

Everybody talks about modernization, but we know it's risky and expensive. So how do you balance running authoring new software with evolving a running system? Joining us this week is John Kodumal, CTO and co founder of LaunchDarkly. Welcome John, great to see you. You introduce yourself and tell us a little bit about LaunchDarkly.[00:01:00]

Yes, thank you for having me. My name is John Kodumal. I'm the CTO and co founder of LaunchDarkly. If you're not familiar with LaunchDarkly, we are an eight year old startup. We're based in the Bay Area. Our mission is to help teams leverage feature management to deliver change with less risk and maximize the impact of the business value of that change.

Okay, so Let's start by understanding a little bit about your journey to launch Darkly. So tell us initially about what the core idea was that you thought an organization with a purpose could get behind helping fix. Yeah, so I started the company with my co founder Edith about a Years ago and I was working at Atlassian at the time, and Edith, my co founder, was working at Concur, which had been a tribute prior to that, which had just gotten acquired by Concur, and one of the things that we observed was this flaw in the way that people released [00:02:00] software namely that, When most organizations released a change out into the world, so they invested all this time to build something, and then they wanted to release that change.

The process by which they did that was very risky. So essentially, you would just, build your artifact, you go through the entire software development life cycle, you get ready to go, you ship the thing, you'd flip a load balancer, and then all of a sudden, all of your customers are experiencing that change.

Now, if that change is bad, it's your fault. Something's wrong, maybe that you introduced a defect, a performance regression or something else. Everybody is seeing that negative impact, and that's not optimal. Obviously, a better way to do that is to progressively roll out that change. And the interesting thing is back in 2014, 2015, when he and I started this company, people were actually getting around that by hacking things like A B testing platforms to do this kind of to release this kind of change.

So they were taking experimentation tools and [00:03:00] repurposing them to the degree that they could, even though those tools weren't purpose built for that use case. So I remember examples of us There was a scenario actually where at Atlassian we had to roll back a change and we couldn't figure out how to roll it back quickly enough.

It was like a sub zero type of incident. And we ended up temporarily resolving the issue by rolling out an optimizely experiment to 100 percent of traffic that kind of disabled the piece of functionality that was causing the incident. So we realized from that point that there was a market need for this.

We also coupled that with an observation that a lot of teams had built internal tools to solve this need and we realized that there was an opportunity to build something commercially that could solve the problem. So perhaps for listeners who are not aware of software release cycle, maybe try and explain what LaunchDarkly does in a way that, we can get our heads

around it.

Yeah for, a non technical audience or a semi technical audience. Yeah, or just not a dev audience. Yeah,[00:04:00] if I go from a completely non technical perspective it's, the way I would describe it is when companies release something out into the world, even in a non software context a fast food restaurant, a takeaway restaurant is testing a new sandwich or something like this, right?

I'd like to get involved in that business. If there's a business in testing sandwiches I'm in. There is they do a massive amount of testing, but that's the important thing, right? If an organization a mcdonald's is releasing a new sandwich, they don't release it to all their franchises Without doing substantial testing and not even just internal testing, right?

These companies do have labs, right? So it's not just in a lab setting or in front of a test audience. They actually released that new sandwich to a small set of test franchises, and then they evaluate whether. That is working in those franchises and they roll it out more broadly and that's a logistics right it's extraordinarily expensive at these organizations that are doing food service at scale to introduce the changes required to serve a new.[00:05:00]

Sandwich to their customers and so it better be worth it so i want to do you risk that change as much as possible and so you apply that into the software setting what's the software. analog of that. And that is what LaunchDarkly is, in a sense. It's the ability to roll out change, to vet that change in a smaller setting, before releasing it more broadly to maximize the impact of that change.

And that can be applied both to legacy system as well as like Greenfield or Newbuild? Exactly. And in fact, I think one of the things that I realized about software development is that the legacy case is actually far more important and far more challenging than a lot of the new build cases as a software developer.

I'm going to make up a statistic here, but like something like 80 90 percent of the changes that we're making are evolutions of an existing system and not. New code, new a new system. I don't create a new repo every day and, build a new microservice from scratch with no repercussions to an existing system.

In fact most of the time [00:06:00] I'm editing or modifying existing system in response to changes in business requirements, changes in scale needs, discovery of defects, etcetera. And in fact, the value of LaunchDarkly is actually much more aligned towards helping companies Evolve existing systems and navigate that change and the risk involved in involving those systems Versus new authorship.

It does both. I just think that the value is much greater. In one set of situations It's rare, isn't it? You get the opportunity to do something completely from scratch and just be free to use all the new technologies There's always some millstone holding you back from the legacy that you need because part of the transactional architecture runs in it So it's that just Awareness that you always have to deal with the legacy at some point, don't you?

It has to kick in. Not everyone gets the joy of Greenfield. I think it's surprising to me how far through your career you have to get to realize that as a software engineer, right? When you go through university, You get to most of the [00:07:00] authorship that you're doing is like a brand new project and so i think a lot of people are unprepared with the reality of you know moving into industry and realizing there's this enormous code base that you are building on top of and that is such a point that you come out of the education system you've cut your teeth in learning how it supposed to work and then the reality of.

Compromise strikes you very hard, isn't it? You go, I learned all this pure it and then you come out and you go, it's a bit messier than we all thought in the first game, didn't it? And this is this dawning realization occurs over your career where you go, Oh, it's not quite as perfect as I hope it would be.

The second time you touch it, it is already legacy, right? So yes, the second commit is legacy, right? But it's interesting because this completely changes that realization that Oh, I'm actually modifying an existing system that is running in production with real customers using it. In the context of a team that is maintaining it, that really changes the way that you approach a lot of things as a software developer, for example when we write [00:08:00] code coming out of university, I think we're optimizing a lot of times for authorship.

Writing code. We focus on programming languages, even programming languages designers. I have a background in programming languages. They focus on like the clarity of authorship, but really they should be focusing on the clarity of readership. How? Because that's what you spend most of your time doing.

Code comprehension, reading existing systems. And I think if we approach it from that perspective, there's a lot to be gained. So it seems like you've approached it from a feature management perspective, rather than say, like a universal underlying modernization of code. So it's like, what are we trying to do from a business perspective, create Darkly helps you evolve your software system as a result of taking sort of a business specific view of it.

Yeah, I would describe LaunchDarkly today as using feature flags as a mechanism for helping companies with that problem. But that fundamental problem of evolving software systems and [00:09:00] getting that change out and maximizing the value of that change. That is LaunchDarkly's mission. Right now it is primarily realized through helping teams leverage feature flags and feature management.

And that is a massive opportunity. There is so much there. It's... Continually surprising how many use cases there are to leverage feature flags in this setting to realize this value. The problem we get a lot we obviously we talk a lot about digital transformation and cloud transformation and obviously in those journeys, it's enormously expensive for organizations to consider You know, bulk modernization of software as part of that migration journey.

I have a little specific point of view that says lift and shift is so radically undervalued from a point of view of just getting everything there. George darkly is as a tool set and feature management as an approach. Is that a route into System modernization post migration modernization yeah absolutely i believe it is i think it's foundational to [00:10:00] that and i can illustrate that if you permit with a story of how we've done something like that internally with a dog story which is always great we actually.

We're still a relatively small company in the overall scale things. We have about 5 56 people on the team today. But we're also eight, nine years old as a company. And so some of the choices that we made early in the history of the company we're going through modernization efforts as well at this point.

To Shaq's point earlier, it's every organization's got it, isn't it? It's just about how new your legacy is. Yeah, absolutely. That second commit. We've gotten to that. We've gotten to that second commit. Exactly. And so one of the, one of the things we went through is this massive database migration.

We took our core database that we use for our application, and we wanted to shift it to a new database. And we chose that new database because we had shifting business

requirements. We wanted to move into A multi region architecture with an active set up. It was a business need. It also enabled things like data localization.

So if we have [00:11:00] customers in the UK, we could service them differently and around their privacy requirements. So we selected a new database and we had this process of having to migrate. Everything, all of our core data into this new database and we ran into the problem that most organizations feel when they go through one of these modernization efforts, this enormous tangled ball of wax and it's so risky that the idea of, okay, we're gonna do everything we can to de risk this.

We're gonna run this in staging. We're gonna, we're gonna, we're gonna spend months So testing this before we have the big day where we do this migration in production. We thought about that and we realized that it's not The modern way to do this that is not how we are going to achieve a successful outcome here and so we leverage feature management to break this down break the problem down let every team do their own migration for their collections or their scope of ownership and isolation within their own road map.

Use feature flags to keep that off and then gradually release each one[00:12:00] until the point where collection by collection, month by month, we were migrating pieces of the system over to the new infrastructure and at the end of the day. Customers never realized along the way that this was happening really.

There was no big launch day. In fact, there was just a day where the system had been running in production on the new database on the old database was sitting there for a while. We were actually running them in peril. And that was one of the benefits that flags would allow us to do something goes wrong.

We could roll back. We hit the roll forward point where we realized we no longer needed the old database and the launch day. It was basically just a ribbon cutting ceremony. It was a decommissioning ceremony for the old database where we were. We're all just like really excited to no longer be paying the tax of running the system and retiring the infrastructure felt really good, but there was zero risk and zero impact from a production standpoint because, three weeks prior, we've been at 95 percent on the new system.

We knew nothing was wrong. We gradually rolled out [00:13:00] the last bits. And so was there a point during that migration? What the application was running concurrently across two databases in fact for months we were operating in that state and so there's a bit of a downside there from a cost perspective but that is massively outweighed by the risk mitigation.

I'm so so effectively we can run in modes where like one system the old system was the source of truth and we are comparing answers between the two systems and then eventually we have confidence to move to the new system and use that as a source of truth. I mean if you think about the value to the business that.

The reduced risk because with a failure in production comes massive potential reputational damage and your brand can be not destroyed, has a massive knock and so yes, you have to spend on the I. T. But actually the benefit to the experience the end users again. Is much, much better and much more valuable.

And I think many organizations look at base I. T. Costs is a line item and think, Oh, we can't afford that. But actually the but if [00:14:00] you don't, the consequences of your actions could be far more dramatic and hit somewhere completely different in your business model. I think this changes the way people need to think about risk with the way their I.

T. Delivers the outcome that they're after. It's quite important change in mindset shift that you just don't look at The base costs, you got to think about the what ifs and avoid the potential pitfalls. Yeah, absolutely. There has to be a risk calculus involved. In fact, that's one of the things that we talk to our customers about in terms of articulating the value of LaunchDarkly.

How can we show you or demonstrate to you how much impact we had on your teams and how much business impact we had by, mitigating that risk? for you. Sometimes it's a little challenging to do right in this database migration story. Everything went great. And can't A B test life. I can't necessarily quantify how bad the impact would have been if we done that the old fashioned way.

I think there are plenty of I. T. Professions out there that remember that cut over. That once goes down has burned indelibly into their [00:15:00] mind where they go, I'm going to do that differently next time. No, it's never good. Never good for usual Friday night or a Saturday night, like flying by the seat of your pants with everything crossed.

Yeah. I think one of the ways to think about this from a risk calculation perspective is imagine the launch day. Okay. How is that going to like operationally? What is that going to look like for us? What are the stress points? And what can we do as a team to eliminate all those stress points to the point where now the launch day that we envision is just that ribbon cutting ceremony, like where we're all just sitting around and I don't know, having a glass of bubbly and enjoying ourselves and then, okay, how can we create that state of reality?

And, feature flags might be 11 opportunity, but just it's a good way of thinking about de risking. Projects and that viewpoint changes the who's interested in this type of capability. You think it would be the software engineers, but actually, no, it's going to be the business owners and the service owners and all that sort of stuff that sort of say, actually, you need to use this type of [00:16:00] approach because the old way of doing things is just too risky.

We don't like it anymore. And I don't like to be on conference calls at three o'clock in the morning. Thank you very much. Yeah, I agree with your point. But I would also add that there is so much impact that can help the developer directly from a collaboration perspective. Just one scenario, for example, we all are in the world of having a shared staging server.

Probably right. We have all these teams collaborating, shipping new change. And then how often are you blocked? Because Staging is down or something bad is happening on stage and that's a collaboration process that your customers are your other you know your peer engineers and so we can help in that scenario to throughout the software development life cycle let's improve testing for you by creating worlds where you're not impacting you know the other consumers of the testing infrastructure.

That's a very good point that the systems that need to synchronize a painful what you're going for is massively parallel. Agile approaches where everybody can run at their own pace. So yeah it's a good enabler. Massively [00:17:00] reduces frustration in the system as well, doesn't it? For engineering purposes.

Yeah. Yeah. If you implement it once that you can use it across each and every environment. Yeah, that's right. Yep. I'm interested in the conversations you've had with customers in terms of mapping business benefits to this. Have you got any examples or a framework by which you think about decomposing modernization or software system evolution in a different way that connects it more directly to business benefit rather than it looking a giant expense to just rebuild the same application of see what I mean.

Yeah, I think oftentimes the challenge that we have is that the product itself is very swiss army knife, right? It's not really in and of itself It is so incredibly flexible in terms of what it allows you to do because it exists at the code level Anything is possible in code. Basically one of the ways in which we do this is we map it directly to One of the critical initiatives that an organization is going through and then we were trying to do a better job of saying, okay, here's a paved path that allows you to [00:18:00] use this technology for that solution.

So a solution might be a database migration where, a company will tell us one of the big initiatives that we have this year is this huge migration that we really need to de risk it or a cloud migration we're migrating into. seven new regions in Azure. We're going multi cloud. The business value is there we want.

But now how do we actually practically make that happen? So some of the use cases that you know, the cloud migration database migration mobile applications are another example of this, where we help you avoid the sort of app store release model. And so we help you get mobile applications to market faster by delivering more experiences behind feature flags and avoiding the the app store approval model and then a B testing experimentation as another critical example of companies that want to run those types of programs and and are increasingly using feature flags as a mechanism to run those programs.

So maybe just by we're bringing today's conversation to a [00:19:00] bit of a close, what advice would you give to CIOs or CTOs who are. Faced with a problem of needing to modernize their environment, maybe don't have the budget that they need. Maybe they're part of all the way through like a data center exit.

What are the first or second steps that they should take to move towards being able to feature manage in the way that you're describing? I would say perhaps this is like overly simple advice, but I think it's the right advice thinking incrementally some stepping back from the technology, stepping back from even feature flags or feature management as a mechanism, forget all the mechanisms, just think about the migration and recognize that if there's risk involved in it one of the best ways that you can de risk a project from a From a process perspective, but also from an outcomes perspective is to modularize it take whatever change, whatever digital transformation, whatever migration effort you have and reduce the problem to how do I make this [00:20:00] incremental? How do I modularize it? And then the technology solutions that you need to make that a reality, they'll come to four. They will come they will make themselves apparent.

But yeah. That is a massive enabler for teams from a productivity perspective. And it's a massive way to de risk these types of things.

Sjoukje, what you've been looking at this week. So each week I will do some research on what's trending in tech and this week I want to focus on the following. Can developer productivity be measured? So we all have a picture that comes to mind when we heard a term developer productivity and for me personally I think the best description is The measure of how efficiently a [00:21:00] developer or the software development team can handle software development operations within a given time frame.

And this goes from building to deploying and maintaining the software. So why is it important to measure this? Basically, you can't improve what isn't measured right. So measuring developer productivity gives you an idea of where the bottlenecks are and what needs to be improved. So what should you measure?

First of all, it is essential to measure developer productivity beyond just the many lines of code a developer can write. And this means that you focus more on the quality and the

innovative effort that is involved in the whole software development process. And this can be done through metrics. So I've done quite some development projects in my career and was also measured based on the lines of code I generated and also the number of code check ins I made, commits that I made.

And I think every developer knows less lines of code is often better for [00:22:00] speed, for readability, et cetera, and also from a sustainability perspective. So a question for you, John, do you think that measuring development productivity is important and that it can also affect the business outcomes? I love the question.

It's been something that I've been thinking about a fair bit over the past few years. I think it is essential. I think that the measurement techniques that we have right now are in their infancy. So I think you mentioned lines of code. I think that is a one particular measure that people have A broader understanding of that's not a reasonable thing to measure and there were also previous efforts you know from my past to the software developer to measure productivity and complexity and efficiency of teams and i think a lot of those have been debunked and then there's dora.

The door metrics is like one of the last one of the latest sort of like innovations in terms of how we measure developer productivity. I think there's a lot of really important stuff that came out of the door work. But the door [00:23:00] authors themselves will describe Dora as the starting point.

And I think when I think about Dora and measuring developer efficiency, I think there are still challenges that we're facing primarily things like All the metrics that we have are proxy metrics around real efficiency. Some things are extraordinarily difficult to measure, like change failure rates, for example, in Dora.

It's extraordinarily challenging to measure that. So even the Dora metrics themselves have methodology problems. So to your question, incredibly important. I think we need more innovation to help us get to a state where we trust the metrics. And then the last thing I'll say is developer efficiency is Necessary it's a necessary set of conditions to measure but it's not sufficient you can have an enormously impactful or an enormously efficient team That is producing little to no business impact.

And so there's a completion of that story which is okay, I can measure how efficiently My [00:24:00] pipelines are measure how efficiently I'm able to deliver change from a local developers desktop to production, but then there's something beyond that, which is like, how good is the change that I'm shipping, how impactful it is to the business, and we have to unlock that as well.

Is anybody doing it well yet, John, that you've seen? I think in some scenarios, it's easier than in others to do especially that last piece, I'll pick a a dichotomy that I see, and it might not be the right dichotomy, but it's something that I've observed is you look at organization like companies like Spotify or their BTC companies, and you think about the impact of those teams.

A lot of those teams are focused on like growth initiatives, impacting KPIs, etc. And so they can be much more quantitative about the impact that they're having. And so they have a better ability to connect efficiency with measurement of business outcomes compared to For example, B2B companies where that type of like growth style building is tougher and attributions [00:25:00] harder.

You choose to invest in building something, you build a capability and then let's say you

have 300 enterprise customers. That's the entirety of your customer base and you have an adoption rate of 10 percent among that. So now you're talking about. 30 customers impacted. Now you have to connect that to their contract values.

Somehow that attribution is really hard. So was that investment worth it? That's a harder question to answer versus, we know that for the Spotify example 30 seconds more obsession time equals X uplift to subscription renewal, much more scientific. And I think Dave, your point comes back to modern operating model.

So when you have a modern operating model that's business aligned, product based, it's easier to show the value stream, the connectivity, you've got rid of the layer cake of traditional it working. And then you don't have this obfuscation of metrics that occurs and how that team measures themselves. It's much easier to relate back to the business outcomes you discussed.

And I think that's very important. Another good reason to have a modern operating model is the full cost of associated with and the value of delivering that outcome. Is [00:26:00] in a domain that's easier to get control of understand and measure as an entity where you get this horrible mutualized it structures of the passage of the 1990s called they want the right to operate in model back it becomes so hard to measure value through shared teams that you just put your head in your hands and you go, how do we ever get out of this mess?

And I think that pivot is really important to the point. I wonder if this focus on sustainability and sustainable code helps drive the right behaviors as well here john do you think I think it does I think it's again like a necessary prerequisite if there is massive debt in terms of maintaining the code base all of your efficiency metrics are gonna tank.

So yeah, I do see that connection. Yeah. It seems to be fledgling at the moment, but it really does seem to me that sort of sloppily written code that's misusing resources on an ongoing basis is not only less sustainable for obvious reasons, but it's just going to cost you more money. Yeah, I will say, though, that the [00:27:00] ability to measure that the quality of the code itself, it's extremely rudimentary, even more rudimentary than our efficiency metrics.

If, like the risk associated with changing a piece of code or the debt associated with code, it's super hard to measure and Yes. The things that I've seen in my past cyclomatic complexity or things like that not very good at measuring the quality of the code. I do see lots of customers still struggling with this to put this in into place correctly.

Setting up a wide variety of metrics is extremely important. Metrics that are based on reaching the business goals for successfully delivering the software and quality of the product and the business outcomes, and not only for the individual developer, but for the team as a whole, and giving this insights to the development team can really improve your product, but also can support them to become more efficient.

Very good. Thank you, Sjoukje. Good discussion. Good provocation. John, thank you also for a great conversation [00:28:00] today. Really thoughtful and interesting to see where, modernization technologies are going to get us to, where drivers like sustainability and putting a different pressure on how we efficiently develop code is a really fascinating, emergent area, I think.

So thanks for your time today. Thank you, it was a pleasure. Now we end up every episode of the show by asking our guests what they're excited about doing next. Now that could be I'm going to see a great film at the weekend, or that could be a really exciting thing from a business perspective you're looking forward to.

So John, what are you excited about doing next? I am excited about reading a book next. And the book I'm about to read, I'm hoping to read that's next on my queue is this book called scaling people tactics for management and company building. And I think it's a strike press book. I'm really excited.

I got, I've gotten a few good recommendations from people I trust on the quality of the book. So looking forward to that. Great recommendation. Who's it by? It is by Claire Hughes Johnson. So a huge thanks to our guest this week. John, [00:29:00] thank you so much for being on the show. Thanks to our producer Marcel, our sound and editing wizards, Ben and Louis, and of course, to all of our listeners.

We're on LinkedIn and X, Dave Chapman, Rob Kernahan, and Sjoukje Zaal. Feel free to follow or connect with us and please get in touch if you have any comments or ideas for the show. And of course, if you haven't already done that, rate and subscribe to our podcast.

See you in another reality next week.

About Capgemini

Capgemini is a global leader in partnering with companies to transform and manage their business by harnessing the power of technology. The Group is guided everyday by its purpose of unleashing human energy through technology for an inclusive and sustainable future. It is a responsible and diverse organization of over 360,000 team members in more than 50 countries. With its strong 55-year heritage and deep industry expertise, Capgemini is trusted by its clients to address the entire breadth of their business needs, from strategy and design to operations, fueled by the fast evolving and innovative world of cloud, data, AI, connectivity, software, digital engineering and platforms. The Group reported in 2022 global revenues of €22 billion.

Get The Future You Want | www.capgemini.com