How to Unblock Your DevSecOps Transformation

Wed, Oct 12, 2022 8:33AM • 55:01

SUMMARY KEYWORDS

devops, transformation, teams, organization, pattern, business, tool, devops transformation, improvement, agile, standardizing, technical, solution, onboarding process, stream, understand, development, practices, customer, quality

00:01
All right, great. So welcome. Good afternoon, everyone. Welcome to our webinar on how to unblock your dev SEC ops transformation. My name is Charles Maddox with the eye for group. I have my colleague, Emilio Osorio from system as tomatoes. And we're hoping to give you some good insights and some key principles around unblocking your dev SEC ops transformation. Alright, without further ado, we're going to jump right into it just for the sake of time. Next slide, please. Little bit bit of background on us. My name is Charles Maddox, again, the founder and principal at the AI for group, we've been in the digital transformation space for the last 10 years, we're based out of Dallas, Texas, some of our specialties and things of focus are in lean, agile training, Dev SEC ops, and also to cybersecurity product resale. millio, you want to give a quick background on yourself and system s minus?

01:06
Great, thank you, Charles. Well, my name is Emilio. I'm the founder and leader at system. So Manos, I'm the Principal Consultant, we have been doing technological transformations for the last 15 years, it's been an awesome ride, really. And currently, we are helping companies of all sizes, but mostly enterprise customers to do their technological modernization and transformation and baseline in that as a DevOps transformation. So I'm very happy to be with you. And I hope you find the information in this webinar useful.

01:43
Thanks. So what is a dev SEC ops transformation? So as we know, if anybody has been doing large scale enterprise level software development, for at least, you know, the last 10 to 20 years, you know, you've probably heard of the waterfall process where different stages of the software development lifecycle are very clear and explicit. And, you know, a gated process might even be in place, but assuming one step leads to another, and it's a very point based approach, very straightforward approach to delivering on a on a requirement for your customer. And, you know, there's challenges with that approach, right? So kind of showing you what, why some of the challenges that that the dev ops approach solves, and you can fast, you know, give me a click here, Amelia, that the DevOps approach is where we look at this as a cyclical cycle where there's feedback loops to previous stages, and we kind of the reality of of it is, is that customers are always giving us feedback, we understand our development teams, as a business owner, or as a product manager are, is giving us feedback. And it's a very cyclical approach. And we've known this DevOps approach, what

kind of supports the DevOps mindset, the cyclical way of working with our customers and throughout our organizations is an agile approach. So agile and DevOps in a way are very synonymous. And as a kind of agile as a, as a subset of, you know, some of the components of how we implement during the DevOps stages of the process, but very much supportive of one another DevOps and agile, we even say that you can't really implement a true DevOps approach without having good agile practices in place. So what about the SEC, this is about dev SEC ops. So security, and as we know, it, security is another non functional requirement that needs to be a part of the solutions that we're building on a large scale, in today's world, especially. And we really emphasize security these days more than ever, because of the hackers that are out there, the systems that are being, you know, broken into and, you know, social security numbers and passport information, you know, being thrown out there on the internet, and people doing a lot of bad stuff with that. So, we'd like to make sure that we're emphasizing security in our development approach, and that's where the dev SEC ops comes into play. And like any of the non functional requirements, whether it be you know, even call it some levels of you know, performance testing or some type of compliance testing, what how we want to support and kind of call that an operational areas or non functional requirements, if you will, but we want to have these type of items planned into how we build our products from the beginning. So as a part of our planning as part of our analysis, as part of our build process as part of our you know, our review or our code scanning process, in our all of our validation, even how we, you know, analyze maybe through telemetry and our customers are using our products. So security is just embedded into how we do things in We want to make sure that especially in today's world, again, with the nature of the internet seeming like the wild, wild west, that we want to have secure products being put out and deployed into the customer market. Alright. Amelia, you want to add anything to that? Well, I'll go ahead. Go ahead.

05:18

Well, really, one of the things we're seeing in the market in the real life is that most of the dev SEC ops transformation and not are not really meeting the expectations from business. There's been a lot a lot of information and things that the market of DevOps is trying to push through organizations to do mostly basic on toolings. They like having this idea that having a tool will actually speed the development, and there are a lot of promises that are not being kept on doing that actual transformation. That's essentially the main focus of this webinar. Why? Why are these transformations not meeting expectations, and this problem is really huge 75% of the websites formations are not going to meet expectations next year. And we're more on the on the same side, right now. So what's the reality of these transformations? Well, most of the things we have seen is that most people start with that team emphasis a team level emphasis. And the problem with the team level emphasis is that if you're a small company, that's perfectly fine, probably your team wins and run this the system. But if you are in a large enterprise, your team will never, never never in a traditional organization, be able to run to build and to run the floor. So we need other teams. And when we need other teams, then the things begin to break. We work in silos, we have management issues, we have political wars. And essentially, we don't have time to do the technological transformation deep enough to realize that the problems, and then there is the talent problems. Probably the most sought after talent right now is DevOps, DevOps engineers or DevOps, all developers who are already trained and practicing DevOps is a huge, huge, huge lack of talent right now. And people get frustrated when you get a development team initiated on DevOps practices, and you train them. And you promise that

things are going to be faster and better, but you keep getting political fights are just not prioritizing the refactoring work that needs to be done. In order to enable the transformation, the talent gets demotivated and they move up, essentially, the they move to another place. And we start, we need to start again, the transformations. So losing that key people is something that we are seeing a lot in the market, developers want to have greener pastures, they want to go to places where they understand how to work well, because their practices essentially are the most important thing for a developer, they they impact directly impact their quality of life. So what we found is most of management of leadership, who is listening to tool vendors, or tool providers are not realizing the depth of the organizational transformation that needs to be done. So we have seen some anti patterns on on this. And I don't know if you want to introduce this pattern thinking Charles.

08:48
Alright, yeah, sure. Well, so. So this is, you know, gave you the problem here about some of the challenges that we're having in organizations. So now we're going to give you some some areas in which to, that you can use as kind of some guidelines and some solution patterns on how to block unblock your, your DevOps transformation. So, I'm sorry, so the patterns here, though, that we see that are immaturity, you know, of the organization, just they're not, you know, your process maturity and how your culture and your organization about working together is just not at a level that's conducive to, you know, to a transformation of this nature to get to get this way of working, establish our resistance to change, you know, that that's a key thing that we all see in a lot of our organizations just from, you know, people that have been in the organization, you know, for 3040 years, you know, it may just be historical culture that we're inheriting from the past that people just want to do what they you know, do what they No, and they're pretty much sticking to the way that they like to do things. And any, anything that's going to shake that up is not is not going to happen. All right. And also to it could be from a resistance to change perspective that there might be incentives or just the way that the organizational culture is being communicated to certain individuals, that it's not beneficial for them to to change or make a change in the way that they work, because it might not be conducive with their bonuses or how they might be seen by their colleagues and their particular silo, if they if they work with this other group in a certain way. Right. So resistance to changes is, is very much clear and present, because of that. And then, and tangentially, just as I'd mentioned, why people may not want to change is because of bureaucratic processes, right? That there's a, there's processes in place to kind of keep the siloed way of thinking in place and not wanting to work with other groups and the incentives in the organization, maybe not to work kind of in it's not communicated directly, oftentimes, but in a way that you know, that's kind of leave it is kind of a common thing that happens with, you know, development in tests, like, I'm a developer,

11:11
you're a tester, I throw it over the wall to you, you do your thing, I do my thing. You know,

11:17
it's not on me, if it doesn't test out, okay, it's because you're testing the wrong thing. But then again, where's the communication breakdown? And why? Why aren't we testing the right things? So it's things of that nature that may keep us from preventing us from having the right culture that would really, you know, that's really conducive to having a successful DevOps transformation. Right? Yeah.

11:40
Yeah, there are some other more technical problems we're finding on the market. One is a huge technical theft. Technical debt has many definitions, one of my favorite definitions is all the things you have in production that you know, are going to break some worry in the future that you didn't ship with the quality you wanted to. There are some other some even some things that you don't really know how they work with, they work and you don't want to damage them. So we see a lot of legacy systems where the developers are no longer in the company, nobody knows how to fix things, or break things. And they are very, very afraid of making changes to that. So technical depth can also be understood as having platforms that are really difficult to modernize. So one of the main things we are doing right now is helping companies to understand how they can achieve that technical depth in the form of legacy systems can be reduced. But there is no clear path to do that there is no recite to do that. So this management challenge challenge, essentially of prioritizing that is a big blocker. People want to do, they may have the culture to do that. But they have technical difficulties in achieving that. And lastly, one of the saddest things I have seen in the market is this emphasis on tools standardization, it seems from some vendors that having buying tools, and integrated tool is what you need to do DevOps, when in reality, most of the companies have a really diverse set of technologies or real slew of technologies. So if one technology one DevOps stack work for cloud native things, it not necessarily will work in mainframe or as 400, or something like monolithic Java development. So this forces, it's a standardization that brings up his internal fighting, if you're a developer who has a really tight schedule, to release things, you are not going to go through the work. That means trying to understand a new tool that someone higher in the organization is trying to push you. And you're going to defend against that and in a radio world style, so you're going to find some reasons to show that you can get into the above picture, because you are not really comfortable with the higher decisions. And doing that one of the main things of this open DevOps movement is to let the teams standardize the tools they need, not on a corporation or enterprise basis, and forget about trying to optimize costs and licensing. Here we need to optimize the speed to earlier. So this design, these are five patterns we have been found finding in our coaching and consulting things. And this pattern thinking thing is really useful. When do you need to find solutions? Because if you understand the root causes of these patterns of cycles, then you can provide solutions. And that's essentially the main goal here.

15:01
Alright, thanks familiar. So as Emilio mentioned that, you know, these are some common patterns. And we're going to, you know, cover these patterns a little bit more in depth, but there's some solution patterns that can align with these areas as well. And so that's what we're going to talk about next. So, solution pattern number one, organic adaption versus organizational immaturity. And a little bit of background on this is that, you know, trying to force a square peg into a round hole, you know, some some organizations want to adopt a transformation or, or a new approach to working, but they're not ready to their culture is not ready to their teams are not educated properly, their leadership is not educated properly. And that's organizational and maturity, they are not ready to adopt things at a certain level. So what we're going to talk about here are some organic ways of adoption, which is trying to put the round peg in a round hole, and what will kind of be an approach that will work for certain organizations and meet them at their place. And Millie, I'll turn it over to you.

16:02
Thank you. Thanks, Jeff. Well, the first thing is, you need to know where you are at. And not every team is at the same level of technical maturity as the other one in we are working with the organization, they have 200 themes, I'm probably just doing a general analysis of them, probably 60% of them are they don't have the technical maturity, they will want to do release on demand. But doing release on demand implies to have a you have automated building of of the software, you have automated testing, you have some way of deploying first and last and recovering for Fall fell deploys fast, and they don't have the technology or they don't know how to do it. So a hire manager wants to do DevOps, and they promisor to do release on the demand for the whole company, they're not going to be able to do it, if they try to follow a recycling, each team has a different path to get to DevOps maturity. So normally, you should start with a DevOps assessment. That's the best way to do it. There are lots of models of maturity models on how to do that, that sat and scaled agile provides some maturity models that I find them quite useful, but quite useful, because they are really easy to implement. The other things, there are some basic things that you need to build, you need to be the next rough structure to actually do the deployments. That's I think that one of the easiest parts to do. But then you got to go forward and try to make your own path may lead the things make their own path to improvement. One of the main drawbacks of trying to use a process that fits all the themes is that they're not going to fit all the themes. And when you find that you're the organization is trying to push you to do something you are not ready to do, you're going to spend time and the results are not going to be as good enough. So for first recommendation is to start with you are that simple. You need to have the transparent way of thinking the maturity to accept that probably DevOps is a long journey, you're not going to get to the vast majority in three months or one year, it's going to probably take longer, but you need to adapt the way that you are. First starting where you are and then selecting the business problem you're trying to solve by doing some improvement. For example, if you have quality issues, and it's clear that you have quality issues, probably automated deployment is not going to solve the quality issues you need to go to better your automation practice and on testing. And that's why a maturity model is a good way to do that in a maturity model jusu you should advance on all the fronts at the same time as you soon have the spikes, where you do things really way better than the standard, but you have some other lagging way behind. So having a maturity assessment, which can be done really easy. This one we are presented here is one of the agility health maturity assessment, in particular, the Scaled Agile Model on maturity, I find it really quite useful to start a conversation with business and plan and a specific roadmap on how you can transform everything.

19:33
Thanks Emilio. So solution pattern number two, and it's kind of very explicit here voluntary transformation versus resistance to change and, and how do we get people over that hurdle of being resistant to change? So before we kick off in this Neil, one of the themes that that I see that you know, it's kind of throughout this whole topic here of you know, trying to unblock your transformation is about culture, culture, culture culture. And that's actually the first principle usually always. And the you know, in any DevOps model is about cultural culture about how do we have a shared responsibility into this into into this effort to get better. So with that said, I'm going to turn it over to you, Emilio, give us a little bit more insight on solution pattern number two.

20:21

Yeah, well, this is this concept in SCADA, yet we use this concept called NB tension based transformation. But invitation based transformation implies something different. And the implied thing is a voluntary transformation. Even though you you know, as a leader, that your organization needs to move forward in the DevOps maturity, you shouldn't you should enforce teams to go in a transformation, when they are not at the moment to do that. There are different stages a team is, if you're probably doing a migration, or doing some huge work, which is really tacked on the scale is not the same, it's not the best moment to start the transformation, you shouldn't wait on the onto the rhythm of the release. Let's do handle the transformation. This voluntary transformation should be included in whatever framework you're using for using Agile management. One of the things that the voluntary transformation should take into account is to specify clearly what are the benefits the team are going to are going to receive by doing this DevOps transformation? What are the business benefits on doing that? And what resources are you willing to put at the disposal to do that? So having an onboarding model, and having a clear process that is understandable by all teams have what it takes to move to a higher level of DevOps maturity, is really an essential part of this. The other thing that we find it really, really helps is to start from the ground up with an onboarding process. The onboarding process is really important because it gives guidelines to the whole organization, and how to move forward. And also this onboarding process can be improved. When you do voluntary transformation. And at the same time to define our really explicit onboarding process, you get huge benefits. One of the first benefits is if you are transforming the first teams, they will be highly motivated to do that. And they will help you fine tune your onboarding process. So on the curve of change, it's always better to start with a team that is relevant, that's also important, you will want to refer more teams that are relevant to the business so they can, so you can have more business support and during the transformation, and share with them that you're building your onboarding process. This is just an example of the phases we need. We have found useful to design a particular onboarding process. One of the main reasons to have a particular onboarding process is to take into account the cultural context in the transformation. Not every company takes decisions the same way. And the onboarding process should be defined, but it comes when it especially if they have difficult or higher number of teams know. So you need a discovery phase, this discovery phase, you need to understand the delivery pipelines do cool, get a baseline of essential metrics, to understand what the business benefit of this transformation they're going to achieve. In my own practice, I have found that most of the technical teams want to once want to do DevOps transformations, but they are not being prioritized by their product owners, product managers or business owners. So this discovery process, don't wait. The leadership can help you provide that transformation growth, but it is out actually achievable, but also supported by business. And that means resources. That means training. That means preparation. The other thing is we shouldn't expect that teams go into DevOps in their day to day work without providing them the allocation of capacity to train themselves or to be trained by someone else. And I also tried to propose this specific enablers of activities they need to do. So the preparation phase is essentially having that map having that practice training so they can begin implementing and then you need to initiate that. We have we have found that actually lounging and providing a specific phase when they know they're starting and celebrate that our first

24:55

real practical results on that is highly He motivating for the developers and on the whole team's doing this. And after we initiated that, and we define initiation as the first time you're doing things, the first time you are doing continuous integration, the first time you're doing automated testing, the first time is not is not the final. The final stage, after you do the for this invitation of practices is really convenient to study like, establish, sterilize them, you need to make them flow, you need to make them become part of the useable day to day work. And that should be the focus of the next phase, achieving flow on these new practices. And after you achieve close, then you can begin to expect improvements. You cannot do improvements if you haven't stabilized practices deep enough in order to begin thinking and improvement. Finally, there are some ways to guide what are the actual work should be done. There's this whole movement of shifting left, what dot what those shifting left means, when shifting left means that the earlier the earlier in the development process, that we are automating get a focus on quality, the cheaper it will be to implement these practices is always cheaper to do good requirements than to do hot fixes in the last day of the release, because we find out that we were not really understanding the business requirements. And when we show them that system demo to them, they are not fully satisfied with where they think or we are not solving the problem. So we normally start the DevOps transformations by standardizing management, backlog management, getting right into the features, the story and the communication tools that you need to have business owners, product owners, product managers in sync with what needs to be achieved, that really is a no brainer, you should do that. You should be weaving with backlog management and the actual agile management part of the stack. And after that, as soon as you begin to automate builds, you need to get into testing. Testing is the most sought after improvement in the first stages of, of work. And I know we all know, we all want to push a button and release into production and then push another button and roll back. But that's really very late on on the actual transformation process. And the business benefit of that really doesn't compare to trying to automate management or ultimate tasting early on. So you need to focus on that focus on shifting left on your transformation. Why I say that, because most of the tool of benders, they try to sell you the end goal, which probably will be months ahead. So try to keep that in mind when you're designing your transformation.

28:13
Alright, Thanks, Amelia for that. So to introduce solution pattern number three, just want to get a little background on this particular topic. One of the things that with any DevOps transformation, kind of one of the primary things that you do right up front is to understand what the value streams are within an organization and how, how value flows within an organization. So and that's critical, right? Because if we don't understand how value flows within an organization, we can optimize around value flow. But oftentimes, there's bureaucracy around that there's silos, there's, you know, there's walls of bureaucracy between one group and another. So how do we work around those walls? Yeah, and that's, I think that's what you're going to give us some solution patterns around?

29:04
Well, we come from a SCADA yellow background. And in Skelly the lien is one of the main things we try to achieve the Lean Flow. So what is organized around value means? Well, essentially, it means that we shouldn't get the hierarchical structure, which is the basis for bureaucratic process. We are not against bureaucratic processing in DevOps or indeed agile, we understand they have the value. When you get into bureaucratic process, you

essentially are saying, I want to control things, I want to control things. I want to make sure that things are being done in a way that the enterprise Finns seems fit and you get into something called sub optimal local optimization. What essentially this means is that every part of the organic Session is trying to optimize his work at process to get the metrics and the things they need. For example, if you're in a testing organization, the sub optimal local optimization will mean to find bugs and try to prevent the work to go into production. But if you go too deep on trying to find bugs, I have found teams who report a bug, then the word or the color of the logo of the port paste, your tracing is not essentially the same, that the corporation marketing procedure stays to do so. And that's really relevant for the use value a business value. So when you get into suboptimal optimization, you get bureaucracy with the bad kind of era cracy, that can that use C's for their silo? And you need to organize around value because you want faster delivering high reward caring quality. How do you do that? Well, you do that by introducing the concept of value stream. Even in this year, the concept of value stream is really strange to most organization, and is one of the biggest things right now in the DevOps space to understand that value stream optimization is the web tool. What is a value stream? Well, a value stream essential is the set of steps that a customer needs to go through in order to receive value from whatever the organization is to in this example, this is a loan, some people want to get a loan from a financial institution. And there are some people who are probably marketing assigned to attract customers, they probably do LinkedIn campaigns or some kind of direct mail campaigns, they need infrastructures to go to those channels, then you go to a web portal where you can the rates and complete the loan application, some people assess your capacity to get the loan and give you terms and then your loan gets awarded, and then it gets separated. And the corporation finally gets the loan repayment with interest in a process of time. And this value stream this set of steps, which is business oriented, not technically oriented, is an operational value stream. This operational value stream is supported by a set of systems or solutions that are the ones that the software development teams are developing the software development, these different channels or different systems solutions, they normally have teams to support them. But these teams are silos in themselves. If you are in the channel, during the portal development, you probably doesn't know anything out with a loan origination system or credit organization or scoring system. But the actual service you provide to the to the customer and customer needs, that if you change some feature, on the marketing side, probably the long organizations in the credit scoring systems, they're going to need changes. So the idea of organizing around value in DevOps is to understand that all these systems shall be a part of part of a team of teams in scaling agile, we mentioned this as an agile release train, which is essentially a team of teams, and do them prioritize and optimize the whole development value stream, independently of which teams are working in this. So this is a really, really useful procedure. And normally, when you begin a DevOps transformation, you see you should do this transformation around value streams. So abolishment of formation is is I think, not difficult to do is really quite easy. You can find videos and texts everywhere. To do that, if you want to have some quick guidance, go to the scaled agile.com.org website and you can find that this means the same information how to do that. So you shall begin by identifying an operational value stream that you need to improve then in identify the solution or systems that you need to optimize in order to opt in to get this business value. Identify the people who develop this and diversify the value streams to build the system that development value streams is the actual delivery pipeline we manage in DevOps. So when you get into this stage, you shouldn't go around trying to get the themes all in the same pace. Every development value streams has constraints, not all the themes

will have the same constraints and one thing we have found really useful to drive and align the transformation to business objectives is to fall to put the basis of this constraint transformation probably testing on a clear business objectives related to the operational value stream. If you do that, we you will get help from business owners if you go go around and try to implement practices. Without understanding the impact on the operational value stream. Business owners are not going to help you, because they don't understand why you need to invest that kind of money or that kind of time on providing some technical value, which you you haven't been able to map to clearly map to business outcomes.

35:24
Alright, thanks Mineo for that. So solution pattern number four is prioritizing enablers versus technical debt. Technical, let's talk about technical debt first kind of being the culprit here, we identified it as one of the blocker the patterns of blockage, that technical debt can be thought of oftentimes, too as a tool, you know, it is something you know, it's an often it's like a credit card, right? A credit card can be beneficial at times to get a quick win or quick purchase, and maybe get a quick release into the production environment and have customers use our solution, but you're paying, you're paying for it, you've done something potentially to you know that there's a quality cleanup, or there's something that we have done that we normally wouldn't do if we had the cash to pay for it up front. But we want to do it maybe because of speed and because of convenience. But the problem is, is when we get too much technical debt when a credit card bill gets too high. And then we got all these credit card bills. And then we're, we're you know, we're drowning in debt. And it's hard for us to get value in anything. So a lot of organizations are stuck in that rut, because they may have legacy systems or things that have been around for a long time. It's just costing them a lot of aches and pains to maintain. And, you know, they're struggling with that. So there's just so much problems at hand, versus having a planned approach and a structured approach to it, which is prioritizing your technical enablers and being able to have a part of your plans, a way of addressing the improvement areas within our our DevOps transformation. So nearly all that you take it away from here.

37:11
Thank you, thank you. One of the things that I think that I Giles transformation has wrapped up in the last 15 years, which is essentially

37:22
what we have been trying to do in the last 15 years is to let businesses drive development. Before agile, what you got was very strong technical organizations that practically set the pace on how to do the actual development, if a business wanted something, they they will have to push the technical organization to try to achieve faster results for the technical organization or was retain the plans. And it was a so separate the technical prioritization from the business prioritization, that's in must be the organization, the technical side, one, not clearly, but providing, I don't know, Slack, a lot of a slack on their solutions, they will make it first of all, to do some things because they have the technical knowledge that these things were going to be better. But it was obscure, it was not transparent how to do that. Most of the reasons the organizations are going into agile or went through the gel transformation was to provide clarity to the business owners and product owners and product managers on where the time was going to spend. But then we the pendulum swung to the to the other side, we now find agile teams, who they have a good product owner or a

defined product owner. And the organization's think that going Agile is going fast, they don't understand that reality has the capacity to change course, with just as a speed. So I have found in the market, that when organizations go Agile, they want to deliver features be business functionality as soon as possible. And then that causes an imbalance between good having a good architecture and a good platform which which is man illogical, who can provide the non functional requirements needed to actually perform and business features and we are seeing a lot of applications and systems that are not really there in quality terms. And one of the solutions is to empower back the technical organization, these roles of architects who should be at the same level as the product ownership or product manager, or product management, they should agree and have a consensus on how to prioritize this product. dose enablers, what are enablers, enablers are things that you need to do in order to provide functionality of value. And these things that you need to do probably will be implemented continuous delivery or continuous integration software, you probably need to do some automated testing on code quality. Both those things, if product management doesn't understand the business value doing that they're not going to be prioritized. And if product managing product management is the sole property, owner of the capacity of the teams, and you don't have architecture, or engineering on the same table, on a collaborative basis, you are not going to get the time or the capacity to do the actual transformation. So our advice here is make sure in whatever model you're using in safe, this is really explicit, you got the system architect, which isn't the same level of capacity location as a product manager. Even though the product manager is the leader, he knows he needs to listen to architecture and engineering in order to provide continuous improvement in their delivery process. So it's really useful to make that explicit as soon as possible, not just product management is going to prioritize functionality, you need to prioritize improvement in the form of enablers, the whole DevOps transformation can be seen as an improvement. What's the practice here to get good adapt? Well, you need to do to have some kind of policies to do capacity allocation. In every sprint in every period of development program increment, you should have previously stated capacity location for new features for technical depth, and for technological enablers like doing DevOps. And you should respect that you should get that clear as soon as possible. Because if you do, do not assign a specific capacity to do that, and probing is not good, it's not going to be magically brought up is simply not going to be done. And you're going to be getting into more depth. As long as you don't do this capacity allocation for improvement. The other thing that we have found on the technical depth side of things, is that most of technological modernization nowadays is based on doing migration, we just a couple of weeks

42:31

earlier, have a conversation with a new FinTech customer, which was trying to release some migration of their baseline main value stream software, which was a technological migration, Big Bang, technological migration, and I think, expanding almost $3 million. And that technological migration, they found out that they really, they they was, wasn't really with the quality or the vision to provide them another platform to do better things. So they just shut down that initiative. And we have seen a lot of things in that regard on doing technology commercialization, is seems that the only thing that we can achieve is doing big bang transformation. And that's not the way to do that. There is this thing called strangler pattern in which you can't you've had monoliths, whatever it's mainframe as 400. Java, it's really weird to say that Java is legacy, but I mean that age now. Java is the new mainframe or the new COBOL. But you get monoliths and you know, if you want to go cloud native or you

want to be cloud ready, do you need to deploy things on smaller machines with containers. And my advice is don't get into big bank transformations, but do use this triangle pattern. And you need to assess which part of the services they are providing which part of the functionality, has the more vitality vitality been defined as need for change? What's the what are the parts of this monolith that normally get more change request and strangle that get that out of the monolith and make the monolith consumed this new architecture and new service and just don't kill the monolith make it irrelevant with the passage of time. This is a easier way to manage a big bang. Transformation. And it has a lot of benefits is safer, is faster, and it's cheaper. So make business know that if you are right now kind of trapped in an old technology legacy technology, there is a way out and he's doing a slice by slice. That's the strangler pattern in essence, in essence

45:01
Alright, thanks for that Emilio in our last pattern here, improvement, standardization versus tool standardization. So in our kickoff here, we talked about how, you know, committing to a certain tool set too early and in pretty much locking yourself into, you know, things that may not work out for you in the future could get very expensive to, you know, scale back and, and start over again, very political, even though we've selected all these tools, these are the tools that we want to go with that that could prevent innovation and from you moving forward in your transformation. And the alternative here is improvement standardization. And there's a there's a saying from Edward Deming, that always resonates with me, Edward Deming being the kind of the guru of lean, he says that there's kind of two ways of him saying that he says that you can't have innovation without standardization. And what he means by that, though, is another kind of stain that he always says that you can't have quality, you can't you have to specify quality, in order to have quality, what is your baseline? What are you standardizing on in terms of your quality level, so that you can show continuous improvement. And we need to standardize on that. So we show that we're moving in the right direction, or maybe I'll kick it off to you.

46:21
Thank you, that's, that's really important, there's really not that much value on standardizing on tools. Every team has different tool set, and the cost of change, of getting all the teams on the same page. And using the same tools. Normally, it doesn't get that really much business value, or take here is that as leaders of these transformations, we should focus on what really matters and what really matters. And that's the essence of DevOps, is this continuous cycle of improvement. If you improve 2%, every month, really, really fast. By the law of compound interest, you're going to have huge improvements in a few months. And that's what you were to. So starting by the tools is not the way to go starting by defining a really small set of measure of metrics that you need to keep track up and lead the teams free on how to achieve those improvements has a lot of a higher business value than us just doing a standardization on tools. What kind of metrics Well, one of the metrics these these are the famous Dora metrics, Dora is an organization which do research on DevOps. And this Dora metrics are kind of this golden standard on improvement and DevOps. The first one is deployment frequency, how, how many times a year, quarter, a month, a week, a day, does your does your team releases things to production, this kind of frequency is a measure of how heavy a release process is or not. It also related with the lead time to changes the tempo changes. What measures is the time when the organization sees that needs a change, probably a bug fix or a new feature, it's changed, both of them are changes when

they first identify the need to do that, to the time when they're actually solved that by putting something in production in a modern modern organization with more or less agility that shouldn't go for features more than a quarter probably. And there are some huge organizations like Amazon, Google, or the big ones that probably have that measured in days, when they detect something that needs to be changed to the day, the day they have it changed. They have that lead time in the order, or days or hours. The only thing is, how many of these changes actually failed when we put them in production. That's a general overview of quality of what we are doing. I did doesn't just test quality on the software development side of things, also on the operations on the actual deployments and the technical process to prepare to put code in production. So that takes Facebook rate is a measure of quality on the whole DevOps process, and then the other the recoverability. Now one of the things that it's really useful is to design for recoverability. What that means, if we put something in production, we should design this to get a roll back soon enough, really fast and not as fast as we are putting that online. So this time to restore service is also a good metric to keep track of and you should employ some are I'm tools that get track of that this image is from Atlassian. We are partners of Atlassian. And we use it lesson because we are convinced that the vision of open Devoxx was Atlassian is the one to go forward. What's this vision on tooling?

50:14
Well, we understand that you got agile, you got DevOps, you got it operations, which are processes related to one another. And you and this and this stage, most of the tool vendors provide our single source solution completely integrated. But that is not necessarily apply well to real life themes. Real Life teams use different things. They don't use the labor, they use Git for doing transformations, but they use GitLab. Or, I don't know, whatever provider of good they got, they may use Jenkins to provide continuous integration. And they are not part of the stack of Atlassian, for example, and one of the things you should look in tool provider is the capacity to integrate a diverse set of tools, or things. But these tool shoots should provide you with the metrics and the measures to actually manage the improvement. And that's one of the things that we need to focus on the teams are standardizing practices, they're not standardizing tools, if you follow that pattern, things are going to be well, because the teams are going to be motivated to invest in practices, not unlearning new tools, nobody has time to learn new tools, when you are entitled scale. So that's one of the recommendations use tool stack that is open enough. So you can use whatever the teams are already using to you to do the development work and standardize on the measures. And I think that's got more or less covers what we wanted to share with you. I don't know, if you want to do some closing remarks, Charles?

51:56
Yeah. Well, thanks, Amelia, for that excellent coverage of that open DevOps model? And yeah, so we presented those, those five, you know, problem patterns and the five solution patterns. And as we mentioned, a minute you can kind of go to the next slide, and you can close out that. Yeah, you know, as Emilio mentioned, our partnership here does include that, you know, we have we have a partnership with Atlassian. But the open DevOps model is, is an open model that is flexible enough to account for, you know, a variety of tool stacks and to our our pattern that we just communicated that, you know, standardizing on a certain toolset is an anti pattern. So, keep it open, keep the open DevOps model in mind. And that way, you have the flexibility to adapt to your environment, and the market going forward with

your DevOps approach. And Emilio, I like to kind of ended though, too, with some of the services that might be available for anyone listening on the DevOps dev SEC ops, service options that we might be able to help them with.

53:01
Sure, sure, just to be consistent, I think that the first thing you should do to begin or to unblock your DevOps Transformations is to assess where you are. And I know if the if you're working in a big corporation is going to be very hard to get funding on things. So what we decided to do is to have this model of being a free assessment, which includes the obstacle identification, the particular pattern that is getting into your organization, and then giving you back after this free assessment, or transformation strategy. We as a team, I for and system, as a manner, can help you to do that transformation guidance, we can put coaches and consultant with your team with your own roadmap and guide you to transformation. And if you aren't, if you don't have time, or you don't want to invest in actually learning a lot of new tools, we even have the option of building a platform team that can be managed and consume as a managed service for you. So you are interested in that you're going to have the presentation here. Some are, or emails for further questions, and everything and myself. I wanted to thank you for your patience. We know we have some inconveniences at the start. But I'm glad you were here. And we will love to hear your questions.

54:28
Thank you, everyone. Yeah, please send us a note. We're going to end the webinar here right at the hour a little bit over. But again, thank you so much for joining. This video will be posted. And yeah, please reach out if you have any questions or like us to follow up with you on anything that we covered in our webinar. Thank you guys. You guys. Have a great day. Thank you so much. Thanks, Amelia. Thank you. Thank you, Charles. Thank you all. Bye bye. Bye bye

54:58
Thank you

How to Unblock Your DevSecOps Transformation

Wed, Oct 12, 2022 8:33AM • 55:01

SUMMARY KEYWORDS

devops, transformation, teams, organization, pattern, business, tool, devops transformation, improvement, agile, standardizing, technical, solution, onboarding process, stream, understand, development, practices, customer, quality

00:01
All right, great. So welcome. Good afternoon, everyone. Welcome to our webinar on how to unblock your dev SEC ops transformation. My name is Charles Maddox with the eye for group. I have my colleague, Emilio Osorio from system as tomatoes. And we're hoping to

give you some good insights and some key principles around unblocking your dev SEC ops transformation. Alright, without further ado, we're going to jump right into it just for the sake of time. Next slide, please. Little bit bit of background on us. My name is Charles Maddox, again, the founder and principal at the AI for group, we've been in the digital transformation space for the last 10 years, we're based out of Dallas, Texas, some of our specialties and things of focus are in lean, agile training, Dev SEC ops, and also to cybersecurity product resale. millio, you want to give a quick background on yourself and system s minus?

01:06
Great, thank you, Charles. Well, my name is Emilio. I'm the founder and leader at system. So Manos, I'm the Principal Consultant, we have been doing technological transformations for the last 15 years, it's been an awesome ride, really. And currently, we are helping companies of all sizes, but mostly enterprise customers to do their technological modernization and transformation and baseline in that as a DevOps transformation. So I'm very happy to be with you. And I hope you find the information in this webinar useful.

01:43
Thanks. So what is a dev SEC ops transformation? So as we know, if anybody has been doing large scale enterprise level software development, for at least, you know, the last 10 to 20 years, you know, you've probably heard of the waterfall process where different stages of the software development lifecycle are very clear and explicit. And, you know, a gated process might even be in place, but assuming one step leads to another, and it's a very point based approach, very straightforward approach to delivering on a on a requirement for your customer. And, you know, there's challenges with that approach, right? So kind of showing you what, why some of the challenges that that the dev ops approach solves, and you can fast, you know, give me a click here, Amelia, that the DevOps approach is where we look at this as a cyclical cycle where there's feedback loops to previous stages, and we kind of the reality of of it is, is that customers are always giving us feedback, we understand our development teams, as a business owner, or as a product manager are, is giving us feedback. And it's a very cyclical approach. And we've known this DevOps approach, what kind of supports the DevOps mindset, the cyclical way of working with our customers and throughout our organizations is an agile approach. So agile and DevOps in a way are very synonymous. And as a kind of agile as a, as a subset of, you know, some of the components of how we implement during the DevOps stages of the process, but very much supportive of one another DevOps and agile, we even say that you can't really implement a true DevOps approach without having good agile practices in place. So what about the SEC, this is about dev SEC ops. So security, and as we know, it, security is another non functional requirement that needs to be a part of the solutions that we're building on a large scale, in today's world, especially. And we really emphasize security these days more than ever, because of the hackers that are out there, the systems that are being, you know, broken into and, you know, social security numbers and passport information, you know, being thrown out there on the internet, and people doing a lot of bad stuff with that. So, we'd like to make sure that we're emphasizing security in our development approach, and that's where the dev SEC ops comes into play. And like any of the non functional requirements, whether it be you know, even call it some levels of you know, performance testing or some type of compliance testing, what how we want to support and kind of call that an operational areas or non functional requirements, if you will, but we want to have these type of items planned into how we build our products from the beginning. So as a part of our planning as part of our

analysis, as part of our build process as part of our you know, our review or our code scanning process, in our all of our validation, even how we, you know, analyze maybe through telemetry and our customers are using our products. So security is just embedded into how we do things in We want to make sure that especially in today's world, again, with the nature of the internet seeming like the wild, wild west, that we want to have secure products being put out and deployed into the customer market. Alright. Amelia, you want to add anything to that? Well, I'll go ahead. Go ahead.

05:18
Well, really, one of the things we're seeing in the market in the real life is that most of the dev SEC ops transformation and not are not really meeting the expectations from business. There's been a lot a lot of information and things that the market of DevOps is trying to push through organizations to do mostly basic on toolings. They like having this idea that having a tool will actually speed the development, and there are a lot of promises that are not being kept on doing that actual transformation. That's essentially the main focus of this webinar. Why? Why are these transformations not meeting expectations, and this problem is really huge 75% of the websites formations are not going to meet expectations next year. And we're more on the on the same side, right now. So what's the reality of these transformations? Well, most of the things we have seen is that most people start with that team emphasis a team level emphasis. And the problem with the team level emphasis is that if you're a small company, that's perfectly fine, probably your team wins and run this the system. But if you are in a large enterprise, your team will never, never never in a traditional organization, be able to run to build and to run the floor. So we need other teams. And when we need other teams, then the things begin to break. We work in silos, we have management issues, we have political wars. And essentially, we don't have time to do the technological transformation deep enough to realize that the problems, and then there is the talent problems. Probably the most sought after talent right now is DevOps, DevOps engineers or DevOps, all developers who are already trained and practicing DevOps is a huge, huge, huge lack of talent right now. And people get frustrated when you get a development team initiated on DevOps practices, and you train them. And you promise that things are going to be faster and better, but you keep getting political fights are just not prioritizing the refactoring work that needs to be done. In order to enable the transformation, the talent gets demotivated and they move up, essentially, the they move to another place. And we start, we need to start again, the transformations. So losing that key people is something that we are seeing a lot in the market, developers want to have greener pastures, they want to go to places where they understand how to work well, because their practices essentially are the most important thing for a developer, they they impact directly impact their quality of life. So what we found is most of management of leadership, who is listening to tool vendors, or tool providers are not realizing the depth of the organizational transformation that needs to be done. So we have seen some anti patterns on on this. And I don't know if you want to introduce this pattern thinking Charles.

08:48
Alright, yeah, sure. Well, so. So this is, you know, gave you the problem here about some of the challenges that we're having in organizations. So now we're going to give you some some areas in which to, that you can use as kind of some guidelines and some solution patterns on how to block unblock your, your DevOps transformation. So, I'm sorry, so the patterns here, though, that we see that are immaturity, you know, of the organization, just

they're not, you know, your process maturity and how your culture and your organization about working together is just not at a level that's conducive to, you know, to a transformation of this nature to get to get this way of working, establish our resistance to change, you know, that that's a key thing that we all see in a lot of our organizations just from, you know, people that have been in the organization, you know, for 3040 years, you know, it may just be historical culture that we're inheriting from the past that people just want to do what they you know, do what they No, and they're pretty much sticking to the way that they like to do things. And any, anything that's going to shake that up is not is not going to happen. All right. And also to it could be from a resistance to change perspective that there might be incentives or just the way that the organizational culture is being communicated to certain individuals, that it's not beneficial for them to to change or make a change in the way that they work, because it might not be conducive with their bonuses or how they might be seen by their colleagues and their particular silo, if they if they work with this other group in a certain way. Right. So resistance to changes is, is very much clear and present, because of that. And then, and tangentially, just as I'd mentioned, why people may not want to change is because of bureaucratic processes, right? That there's a, there's processes in place to kind of keep the siloed way of thinking in place and not wanting to work with other groups and the incentives in the organization, maybe not to work kind of in it's not communicated directly, oftentimes, but in a way that you know, that's kind of leave it is kind of a common thing that happens with, you know, development in tests, like, I'm a developer,

11:11
you're a tester, I throw it over the wall to you, you do your thing, I do my thing. You know,

11:17
it's not on me, if it doesn't test out, okay, it's because you're testing the wrong thing. But then again, where's the communication breakdown? And why? Why aren't we testing the right things? So it's things of that nature that may keep us from preventing us from having the right culture that would really, you know, that's really conducive to having a successful DevOps transformation. Right? Yeah.

11:40
Yeah, there are some other more technical problems we're finding on the market. One is a huge technical theft. Technical debt has many definitions, one of my favorite definitions is all the things you have in production that you know, are going to break some worry in the future that you didn't ship with the quality you wanted to. There are some other some even some things that you don't really know how they work with, they work and you don't want to damage them. So we see a lot of legacy systems where the developers are no longer in the company, nobody knows how to fix things, or break things. And they are very, very afraid of making changes to that. So technical depth can also be understood as having platforms that are really difficult to modernize. So one of the main things we are doing right now is helping companies to understand how they can achieve that technical depth in the form of legacy systems can be reduced. But there is no clear path to do that there is no recite to do that. So this management challenge challenge, essentially of prioritizing that is a big blocker. People want to do, they may have the culture to do that. But they have technical difficulties in achieving that. And lastly, one of the saddest things I have seen in the market is this emphasis on tools standardization, it seems from some vendors that having buying tools, and integrated tool is what you need to do DevOps, when in reality, most of the companies

have a really diverse set of technologies or real slew of technologies. So if one technology one DevOps stack work for cloud native things, it not necessarily will work in mainframe or as 400, or something like monolithic Java development. So this forces, it's a standardization that brings up his internal fighting, if you're a developer who has a really tight schedule, to release things, you are not going to go through the work. That means trying to understand a new tool that someone higher in the organization is trying to push you. And you're going to defend against that and in a radio world style, so you're going to find some reasons to show that you can get into the above picture, because you are not really comfortable with the higher decisions. And doing that one of the main things of this open DevOps movement is to let the teams standardize the tools they need, not on a corporation or enterprise basis, and forget about trying to optimize costs and licensing. Here we need to optimize the speed to earlier. So this design, these are five patterns we have been found finding in our coaching and consulting things. And this pattern thinking thing is really useful. When do you need to find solutions? Because if you understand the root causes of these patterns of cycles, then you can provide solutions. And that's essentially the main goal here.

15:01
Alright, thanks familiar. So as Emilio mentioned that, you know, these are some common patterns. And we're going to, you know, cover these patterns a little bit more in depth, but there's some solution patterns that can align with these areas as well. And so that's what we're going to talk about next. So, solution pattern number one, organic adaption versus organizational immaturity. And a little bit of background on this is that, you know, trying to force a square peg into a round hole, you know, some some organizations want to adopt a transformation or, or a new approach to working, but they're not ready to their culture is not ready to their teams are not educated properly, their leadership is not educated properly. And that's organizational and maturity, they are not ready to adopt things at a certain level. So what we're going to talk about here are some organic ways of adoption, which is trying to put the round peg in a round hole, and what will kind of be an approach that will work for certain organizations and meet them at their place. And Millie, I'll turn it over to you.

16:02
Thank you. Thanks, Jeff. Well, the first thing is, you need to know where you are at. And not every team is at the same level of technical maturity as the other one in we are working with the organization, they have 200 themes, I'm probably just doing a general analysis of them, probably 60% of them are they don't have the technical maturity, they will want to do release on demand. But doing release on demand implies to have a you have automated building of of the software, you have automated testing, you have some way of deploying first and last and recovering for Fall fell deploys fast, and they don't have the technology or they don't know how to do it. So a hire manager wants to do DevOps, and they promisor to do release on the demand for the whole company, they're not going to be able to do it, if they try to follow a recycling, each team has a different path to get to DevOps maturity. So normally, you should start with a DevOps assessment. That's the best way to do it. There are lots of models of maturity models on how to do that, that sat and scaled agile provides some maturity models that I find them quite useful, but quite useful, because they are really easy to implement. The other things, there are some basic things that you need to build, you need to be the next rough structure to actually do the deployments. That's I think that one of the easiest parts to do. But then you got to go forward and try to make your own path may lead the things make their own path to improvement. One of the main drawbacks of trying to use

a process that fits all the themes is that they're not going to fit all the themes. And when you find that you're the organization is trying to push you to do something you are not ready to do, you're going to spend time and the results are not going to be as good enough. So for first recommendation is to start with you are that simple. You need to have the transparent way of thinking the maturity to accept that probably DevOps is a long journey, you're not going to get to the vast majority in three months or one year, it's going to probably take longer, but you need to adapt the way that you are. First starting where you are and then selecting the business problem you're trying to solve by doing some improvement. For example, if you have quality issues, and it's clear that you have quality issues, probably automated deployment is not going to solve the quality issues you need to go to better your automation practice and on testing. And that's why a maturity model is a good way to do that in a maturity model jusu you should advance on all the fronts at the same time as you soon have the spikes, where you do things really way better than the standard, but you have some other lagging way behind. So having a maturity assessment, which can be done really easy. This one we are presented here is one of the agility health maturity assessment, in particular, the Scaled Agile Model on maturity, I find it really quite useful to start a conversation with business and plan and a specific roadmap on how you can transform everything.

19:33
Thanks Emilio. So solution pattern number two, and it's kind of very explicit here voluntary transformation versus resistance to change and, and how do we get people over that hurdle of being resistant to change? So before we kick off in this Neil, one of the themes that that I see that you know, it's kind of throughout this whole topic here of you know, trying to unblock your transformation is about culture, culture, culture culture. And that's actually the first principle usually always. And the you know, in any DevOps model is about cultural culture about how do we have a shared responsibility into this into into this effort to get better. So with that said, I'm going to turn it over to you, Emilio, give us a little bit more insight on solution pattern number two.

20:21
Yeah, well, this is this concept in SCADA, yet we use this concept called NB tension based transformation. But invitation based transformation implies something different. And the implied thing is a voluntary transformation. Even though you you know, as a leader, that your organization needs to move forward in the DevOps maturity, you shouldn't you should enforce teams to go in a transformation, when they are not at the moment to do that. There are different stages a team is, if you're probably doing a migration, or doing some huge work, which is really tacked on the scale is not the same, it's not the best moment to start the transformation, you shouldn't wait on the onto the rhythm of the release. Let's do handle the transformation. This voluntary transformation should be included in whatever framework you're using for using Agile management. One of the things that the voluntary transformation should take into account is to specify clearly what are the benefits the team are going to are going to receive by doing this DevOps transformation? What are the business benefits on doing that? And what resources are you willing to put at the disposal to do that? So having an onboarding model, and having a clear process that is understandable by all teams have what it takes to move to a higher level of DevOps maturity, is really an essential part of this. The other thing that we find it really, really helps is to start from the ground up with an onboarding process. The onboarding process is really important because it gives guidelines

to the whole organization, and how to move forward. And also this onboarding process can be improved. When you do voluntary transformation. And at the same time to define our really explicit onboarding process, you get huge benefits. One of the first benefits is if you are transforming the first teams, they will be highly motivated to do that. And they will help you fine tune your onboarding process. So on the curve of change, it's always better to start with a team that is relevant, that's also important, you will want to refer more teams that are relevant to the business so they can, so you can have more business support and during the transformation, and share with them that you're building your onboarding process. This is just an example of the phases we need. We have found useful to design a particular onboarding process. One of the main reasons to have a particular onboarding process is to take into account the cultural context in the transformation. Not every company takes decisions the same way. And the onboarding process should be defined, but it comes when it especially if they have difficult or higher number of teams know. So you need a discovery phase, this discovery phase, you need to understand the delivery pipelines do cool, get a baseline of essential metrics, to understand what the business benefit of this transformation they're going to achieve. In my own practice, I have found that most of the technical teams want to once want to do DevOps transformations, but they are not being prioritized by their product owners, product managers or business owners. So this discovery process, don't wait. The leadership can help you provide that transformation growth, but it is out actually achievable, but also supported by business. And that means resources. That means training. That means preparation. The other thing is we shouldn't expect that teams go into DevOps in their day to day work without providing them the allocation of capacity to train themselves or to be trained by someone else. And I also tried to propose this specific enablers of activities they need to do. So the preparation phase is essentially having that map having that practice training so they can begin implementing and then you need to initiate that. We have we have found that actually lounging and providing a specific phase when they know they're starting and celebrate that our first

24:55

real practical results on that is highly He motivating for the developers and on the whole team's doing this. And after we initiated that, and we define initiation as the first time you're doing things, the first time you are doing continuous integration, the first time you're doing automated testing, the first time is not is not the final. The final stage, after you do the for this invitation of practices is really convenient to study like, establish, sterilize them, you need to make them flow, you need to make them become part of the useable day to day work. And that should be the focus of the next phase, achieving flow on these new practices. And after you achieve close, then you can begin to expect improvements. You cannot do improvements if you haven't stabilized practices deep enough in order to begin thinking and improvement. Finally, there are some ways to guide what are the actual work should be done. There's this whole movement of shifting left, what dot what those shifting left means, when shifting left means that the earlier the earlier in the development process, that we are automating get a focus on quality, the cheaper it will be to implement these practices is always cheaper to do good requirements than to do hot fixes in the last day of the release, because we find out that we were not really understanding the business requirements. And when we show them that system demo to them, they are not fully satisfied with where they think or we are not solving the problem. So we normally start the DevOps transformations by standardizing management, backlog management, getting right into the features, the story and the communication tools that you need to have business owners, product owners,

product managers in sync with what needs to be achieved, that really is a no brainer, you should do that. You should be weaving with backlog management and the actual agile management part of the stack. And after that, as soon as you begin to automate builds, you need to get into testing. Testing is the most sought after improvement in the first stages of, of work. And I know we all know, we all want to push a button and release into production and then push another button and roll back. But that's really very late on on the actual transformation process. And the business benefit of that really doesn't compare to trying to automate management or ultimate tasting early on. So you need to focus on that focus on shifting left on your transformation. Why I say that, because most of the tool of benders, they try to sell you the end goal, which probably will be months ahead. So try to keep that in mind when you're designing your transformation.

28:13
Alright, Thanks, Amelia for that. So to introduce solution pattern number three, just want to get a little background on this particular topic. One of the things that with any DevOps transformation, kind of one of the primary things that you do right up front is to understand what the value streams are within an organization and how, how value flows within an organization. So and that's critical, right? Because if we don't understand how value flows within an organization, we can optimize around value flow. But oftentimes, there's bureaucracy around that there's silos, there's, you know, there's walls of bureaucracy between one group and another. So how do we work around those walls? Yeah, and that's, I think that's what you're going to give us some solution patterns around?

29:04
Well, we come from a SCADA yellow background. And in Skelly the lien is one of the main things we try to achieve the Lean Flow. So what is organized around value means? Well, essentially, it means that we shouldn't get the hierarchical structure, which is the basis for bureaucratic process. We are not against bureaucratic processing in DevOps or indeed agile, we understand they have the value. When you get into bureaucratic process, you essentially are saying, I want to control things, I want to control things. I want to make sure that things are being done in a way that the enterprise Finns seems fit and you get into something called sub optimal local optimization. What essentially this means is that every part of the organic Session is trying to optimize his work at process to get the metrics and the things they need. For example, if you're in a testing organization, the sub optimal local optimization will mean to find bugs and try to prevent the work to go into production. But if you go too deep on trying to find bugs, I have found teams who report a bug, then the word or the color of the logo of the port paste, your tracing is not essentially the same, that the corporation marketing procedure stays to do so. And that's really relevant for the use value a business value. So when you get into suboptimal optimization, you get bureaucracy with the bad kind of era cracy, that can that use C's for their silo? And you need to organize around value because you want faster delivering high reward caring quality. How do you do that? Well, you do that by introducing the concept of value stream. Even in this year, the concept of value stream is really strange to most organization, and is one of the biggest things right now in the DevOps space to understand that value stream optimization is the web tool. What is a value stream? Well, a value stream essential is the set of steps that a customer needs to go through in order to receive value from whatever the organization is to in this example, this is a loan, some people want to get a loan from a financial institution. And there are some people who are probably marketing assigned to attract customers, they probably do LinkedIn

campaigns or some kind of direct mail campaigns, they need infrastructures to go to those channels, then you go to a web portal where you can the rates and complete the loan application, some people assess your capacity to get the loan and give you terms and then your loan gets awarded, and then it gets separated. And the corporation finally gets the loan repayment with interest in a process of time. And this value stream this set of steps, which is business oriented, not technically oriented, is an operational value stream. This operational value stream is supported by a set of systems or solutions that are the ones that the software development teams are developing the software development, these different channels or different systems solutions, they normally have teams to support them. But these teams are silos in themselves. If you are in the channel, during the portal development, you probably doesn't know anything out with a loan origination system or credit organization or scoring system. But the actual service you provide to the to the customer and customer needs, that if you change some feature, on the marketing side, probably the long organizations in the credit scoring systems, they're going to need changes. So the idea of organizing around value in DevOps is to understand that all these systems shall be a part of part of a team of teams in scaling agile, we mentioned this as an agile release train, which is essentially a team of teams, and do them prioritize and optimize the whole development value stream, independently of which teams are working in this. So this is a really, really useful procedure. And normally, when you begin a DevOps transformation, you see you should do this transformation around value streams. So abolishment of formation is is I think, not difficult to do is really quite easy. You can find videos and texts everywhere. To do that, if you want to have some quick guidance, go to the scaled agile.com.org website and you can find that this means the same information how to do that. So you shall begin by identifying an operational value stream that you need to improve then in identify the solution or systems that you need to optimize in order to opt in to get this business value. Identify the people who develop this and diversify the value streams to build the system that development value streams is the actual delivery pipeline we manage in DevOps. So when you get into this stage, you shouldn't go around trying to get the themes all in the same pace. Every development value streams has constraints, not all the themes will have the same constraints and one thing we have found really useful to drive and align the transformation to business objectives is to fall to put the basis of this constraint transformation probably testing on a clear business objectives related to the operational value stream. If you do that, we you will get help from business owners if you go go around and try to implement practices. Without understanding the impact on the operational value stream. Business owners are not going to help you, because they don't understand why you need to invest that kind of money or that kind of time on providing some technical value, which you you haven't been able to map to clearly map to business outcomes.

35:24

Alright, thanks Mineo for that. So solution pattern number four is prioritizing enablers versus technical debt. Technical, let's talk about technical debt first kind of being the culprit here, we identified it as one of the blocker the patterns of blockage, that technical debt can be thought of oftentimes, too as a tool, you know, it is something you know, it's an often it's like a credit card, right? A credit card can be beneficial at times to get a quick win or quick purchase, and maybe get a quick release into the production environment and have customers use our solution, but you're paying, you're paying for it, you've done something potentially to you know that there's a quality cleanup, or there's something that we have done that we normally wouldn't do if we had the cash to pay for it up front. But we want to do it maybe because of

speed and because of convenience. But the problem is, is when we get too much technical debt when a credit card bill gets too high. And then we got all these credit card bills. And then we're, we're you know, we're drowning in debt. And it's hard for us to get value in anything. So a lot of organizations are stuck in that rut, because they may have legacy systems or things that have been around for a long time. It's just costing them a lot of aches and pains to maintain. And, you know, they're struggling with that. So there's just so much problems at hand, versus having a planned approach and a structured approach to it, which is prioritizing your technical enablers and being able to have a part of your plans, a way of addressing the improvement areas within our our DevOps transformation. So nearly all that you take it away from here.

37:11
Thank you, thank you. One of the things that I think that I Giles transformation has wrapped up in the last 15 years, which is essentially

37:22
what we have been trying to do in the last 15 years is to let businesses drive development. Before agile, what you got was very strong technical organizations that practically set the pace on how to do the actual development, if a business wanted something, they they will have to push the technical organization to try to achieve faster results for the technical organization or was retain the plans. And it was a so separate the technical prioritization from the business prioritization, that's in must be the organization, the technical side, one, not clearly, but providing, I don't know, Slack, a lot of a slack on their solutions, they will make it first of all, to do some things because they have the technical knowledge that these things were going to be better. But it was obscure, it was not transparent how to do that. Most of the reasons the organizations are going into agile or went through the gel transformation was to provide clarity to the business owners and product owners and product managers on where the time was going to spend. But then we the pendulum swung to the to the other side, we now find agile teams, who they have a good product owner or a defined product owner. And the organization's think that going Agile is going fast, they don't understand that reality has the capacity to change course, with just as a speed. So I have found in the market, that when organizations go Agile, they want to deliver features be business functionality as soon as possible. And then that causes an imbalance between good having a good architecture and a good platform which which is man illogical, who can provide the non functional requirements needed to actually perform and business features and we are seeing a lot of applications and systems that are not really there in quality terms. And one of the solutions is to empower back the technical organization, these roles of architects who should be at the same level as the product ownership or product manager, or product management, they should agree and have a consensus on how to prioritize this product. dose enablers, what are enablers, enablers are things that you need to do in order to provide functionality of value. And these things that you need to do probably will be implemented continuous delivery or continuous integration software, you probably need to do some automated testing on code quality. Both those things, if product management doesn't understand the business value doing that they're not going to be prioritized. And if product managing product management is the sole property, owner of the capacity of the teams, and you don't have architecture, or engineering on the same table, on a collaborative basis, you are not going to get the time or the capacity to do the actual transformation. So our advice here is make sure in whatever model you're using in safe, this is really explicit,

you got the system architect, which isn't the same level of capacity location as a product manager. Even though the product manager is the leader, he knows he needs to listen to architecture and engineering in order to provide continuous improvement in their delivery process. So it's really useful to make that explicit as soon as possible, not just product management is going to prioritize functionality, you need to prioritize improvement in the form of enablers, the whole DevOps transformation can be seen as an improvement. What's the practice here to get good adapt? Well, you need to do to have some kind of policies to do capacity allocation. In every sprint in every period of development program increment, you should have previously stated capacity location for new features for technical depth, and for technological enablers like doing DevOps. And you should respect that you should get that clear as soon as possible. Because if you do, do not assign a specific capacity to do that, and probing is not good, it's not going to be magically brought up is simply not going to be done. And you're going to be getting into more depth. As long as you don't do this capacity allocation for improvement. The other thing that we have found on the technical depth side of things, is that most of technological modernization nowadays is based on doing migration, we just a couple of weeks

42:31
earlier, have a conversation with a new FinTech customer, which was trying to release some migration of their baseline main value stream software, which was a technological migration, Big Bang, technological migration, and I think, expanding almost $3 million. And that technological migration, they found out that they really, they they was, wasn't really with the quality or the vision to provide them another platform to do better things. So they just shut down that initiative. And we have seen a lot of things in that regard on doing technology commercialization, is seems that the only thing that we can achieve is doing big bang transformation. And that's not the way to do that. There is this thing called strangler pattern in which you can't you've had monoliths, whatever it's mainframe as 400. Java, it's really weird to say that Java is legacy, but I mean that age now. Java is the new mainframe or the new COBOL. But you get monoliths and you know, if you want to go cloud native or you want to be cloud ready, do you need to deploy things on smaller machines with containers. And my advice is don't get into big bank transformations, but do use this triangle pattern. And you need to assess which part of the services they are providing which part of the functionality, has the more vitality vitality been defined as need for change? What's the what are the parts of this monolith that normally get more change request and strangle that get that out of the monolith and make the monolith consumed this new architecture and new service and just don't kill the monolith make it irrelevant with the passage of time. This is a easier way to manage a big bang. Transformation. And it has a lot of benefits is safer, is faster, and it's cheaper. So make business know that if you are right now kind of trapped in an old technology legacy technology, there is a way out and he's doing a slice by slice. That's the strangler pattern in essence, in essence

45:01
Alright, thanks for that Emilio in our last pattern here, improvement, standardization versus tool standardization. So in our kickoff here, we talked about how, you know, committing to a certain tool set too early and in pretty much locking yourself into, you know, things that may not work out for you in the future could get very expensive to, you know, scale back and, and start over again, very political, even though we've selected all these tools, these are the tools that we want to go with that that could prevent innovation and from you moving forward in

your transformation. And the alternative here is improvement standardization. And there's a there's a saying from Edward Deming, that always resonates with me, Edward Deming being the kind of the guru of lean, he says that there's kind of two ways of him saying that he says that you can't have innovation without standardization. And what he means by that, though, is another kind of stain that he always says that you can't have quality, you can't you have to specify quality, in order to have quality, what is your baseline? What are you standardizing on in terms of your quality level, so that you can show continuous improvement. And we need to standardize on that. So we show that we're moving in the right direction, or maybe I'll kick it off to you.

46:21
Thank you, that's, that's really important, there's really not that much value on standardizing on tools. Every team has different tool set, and the cost of change, of getting all the teams on the same page. And using the same tools. Normally, it doesn't get that really much business value, or take here is that as leaders of these transformations, we should focus on what really matters and what really matters. And that's the essence of DevOps, is this continuous cycle of improvement. If you improve 2%, every month, really, really fast. By the law of compound interest, you're going to have huge improvements in a few months. And that's what you were to. So starting by the tools is not the way to go starting by defining a really small set of measure of metrics that you need to keep track up and lead the teams free on how to achieve those improvements has a lot of a higher business value than us just doing a standardization on tools. What kind of metrics Well, one of the metrics these these are the famous Dora metrics, Dora is an organization which do research on DevOps. And this Dora metrics are kind of this golden standard on improvement and DevOps. The first one is deployment frequency, how, how many times a year, quarter, a month, a week, a day, does your does your team releases things to production, this kind of frequency is a measure of how heavy a release process is or not. It also related with the lead time to changes the tempo changes. What measures is the time when the organization sees that needs a change, probably a bug fix or a new feature, it's changed, both of them are changes when they first identify the need to do that, to the time when they're actually solved that by putting something in production in a modern modern organization with more or less agility that shouldn't go for features more than a quarter probably. And there are some huge organizations like Amazon, Google, or the big ones that probably have that measured in days, when they detect something that needs to be changed to the day, the day they have it changed. They have that lead time in the order, or days or hours. The only thing is, how many of these changes actually failed when we put them in production. That's a general overview of quality of what we are doing. I did doesn't just test quality on the software development side of things, also on the operations on the actual deployments and the technical process to prepare to put code in production. So that takes Facebook rate is a measure of quality on the whole DevOps process, and then the other the recoverability. Now one of the things that it's really useful is to design for recoverability. What that means, if we put something in production, we should design this to get a roll back soon enough, really fast and not as fast as we are putting that online. So this time to restore service is also a good metric to keep track of and you should employ some are I'm tools that get track of that this image is from Atlassian. We are partners of Atlassian. And we use it lesson because we are convinced that the vision of open Devoxx was Atlassian is the one to go forward. What's this vision on tooling?

50:14
Well, we understand that you got agile, you got DevOps, you got it operations, which are processes related to one another. And you and this and this stage, most of the tool vendors provide our single source solution completely integrated. But that is not necessarily apply well to real life themes. Real Life teams use different things. They don't use the labor, they use Git for doing transformations, but they use GitLab. Or, I don't know, whatever provider of good they got, they may use Jenkins to provide continuous integration. And they are not part of the stack of Atlassian, for example, and one of the things you should look in tool provider is the capacity to integrate a diverse set of tools, or things. But these tool shoots should provide you with the metrics and the measures to actually manage the improvement. And that's one of the things that we need to focus on the teams are standardizing practices, they're not standardizing tools, if you follow that pattern, things are going to be well, because the teams are going to be motivated to invest in practices, not unlearning new tools, nobody has time to learn new tools, when you are entitled scale. So that's one of the recommendations use tool stack that is open enough. So you can use whatever the teams are already using to you to do the development work and standardize on the measures. And I think that's got more or less covers what we wanted to share with you. I don't know, if you want to do some closing remarks, Charles?

51:56
Yeah. Well, thanks, Amelia, for that excellent coverage of that open DevOps model? And yeah, so we presented those, those five, you know, problem patterns and the five solution patterns. And as we mentioned, a minute you can kind of go to the next slide, and you can close out that. Yeah, you know, as Emilio mentioned, our partnership here does include that, you know, we have we have a partnership with Atlassian. But the open DevOps model is, is an open model that is flexible enough to account for, you know, a variety of tool stacks and to our our pattern that we just communicated that, you know, standardizing on a certain toolset is an anti pattern. So, keep it open, keep the open DevOps model in mind. And that way, you have the flexibility to adapt to your environment, and the market going forward with your DevOps approach. And Emilio, I like to kind of ended though, too, with some of the services that might be available for anyone listening on the DevOps dev SEC ops, service options that we might be able to help them with.

53:01
Sure, sure, just to be consistent, I think that the first thing you should do to begin or to unblock your DevOps Transformations is to assess where you are. And I know if the if you're working in a big corporation is going to be very hard to get funding on things. So what we decided to do is to have this model of being a free assessment, which includes the obstacle identification, the particular pattern that is getting into your organization, and then giving you back after this free assessment, or transformation strategy. We as a team, I for and system, as a manner, can help you to do that transformation guidance, we can put coaches and consultant with your team with your own roadmap and guide you to transformation. And if you aren't, if you don't have time, or you don't want to invest in actually learning a lot of new tools, we even have the option of building a platform team that can be managed and consume as a managed service for you. So you are interested in that you're going to have the presentation here. Some are, or emails for further questions, and everything and myself. I wanted to thank you for your patience. We know we have some inconveniences at the start. But I'm glad you were here. And we will love to hear your questions.

54:28
Thank you, everyone. Yeah, please send us a note. We're going to end the webinar here right at the hour a little bit over. But again, thank you so much for joining. This video will be posted. And yeah, please reach out if you have any questions or like us to follow up with you on anything that we covered in our webinar. Thank you guys. You guys. Have a great day. Thank you so much. Thanks, Amelia. Thank you. Thank you, Charles. Thank you all. Bye bye. Bye bye

54:58
Thank you