THE ASTRIX DIGITAL TRANSFORMATION PODCAST SERIES

Data Integrity: Strategies and Tactics for Avoiding Disaster in Program Implementation

Program #4: March 2022

astrix

Data Integrity: Strategies and Tactics for Avoiding Disaster in Program Implementation

A conversation with Dave Dorsett, Principal Software Architect, Astrix

Program #4 - March 2022

Play the Podcast

The Astrix Digital Transformation Podcast Series presents thought leaders and their unique insights into enabling success in your scientific operations. Here's the transcript of this week's program.

About this Program:

While it may never happen to you, there ARE times when a data management system suffers a critical hit and no longer performs. The accelerated pace of efforts towards digital transformation to serve critical data science initiatives can be a contributing factor to failures. In this program, Dave Dorsett, Astrix Principal Software Architect, guides you through strategic approaches that will help remediate systems more efficiently, and even avoid situations leading to catastrophic failure.

Kevin: Welcome to Astrix Digital Transformation podcast series. These are interviews with thought leaders, providing expert insights and actionable advice designed to help you develop a successful digital first strategy to transform your business. My name is Kevin Miller. I'm your host for today's podcast, and thank you very much for listening.

So, before we jump into it, I want to do a quick introduction and discussion of what we're going to talk about today. We're glad to have a familiar face with us, Dave Dorsett's joining us, I'll introduce him in just a moment. He was with us in our last podcast, and he's going to be speaking about some different topics today, not necessarily a continuation of what we talked about last time, but very much focused around some tough subjects when it comes to digital transformation.

What are the strategies and what are the tactics to avoiding a disaster in your program implementation? Data management systems can suffer a critical hit, and when they do and they don't perform anymore, how do we respond to a problem like that, and how do we fix it?

So, kind of a crisis mode discussion today. We don't want crises to occur, but we invariably know that they're going to when we're dealing with technology and when we're dealing with data. So just like that we want our first responders to see the big picture, bring all their experience and their skills to act in a quick, calm and controlled manner, that's how we want all of our first responders, regardless of what the problem is, to work.

We have an expert with us on this topic today who's very uniquely qualified to guide you through the steps to bring your data integrity back from the brink of disaster —Dave Dorsett, Principal Software Architect at Astrix. Dave and I have had a few discussions in the past, so I'm always thrilled to have him and his expertise with us. Dave is a highly experienced leader in the R&D space, and he brings R&D informatics insights to this topic comes from direct experience leading global digital transformation initiatives. Dave's work has enabled improvements in the life cycle of R&D data, from early research all the way through product development.

So again, we're thrilled to have you Dave, thank you for making the trip. And if you want to just maybe give a quick little intro taken off from hopefully setting you up there?

Dave: Sure, I think it's unfortunately true, I do have too much experience in firefighting, I would say. It's been something that has been, well, the experience has been opportunistic, let's just put it that way, from several different points of view. I spent a couple of decades on the vendor side of R&D systems as a vendor providing critical data management systems to the R&D community. And so, from that perspective, I participated in a number of exercises with customers deploying systems and crises that came sort of in and around that, a fair amount of experience from that perspective.

I also spent about a decade from the customer side of things, at a large pharmaceutical company where I experienced systems that didn't behave as originally intended, let's put it that way, from the perspective of being a system owner inside of a very large company and having to handle those types of situations.

And then more recently I've been a consultant for the industry, I have worked a couple of gigs that have been what I would characterize as firefighting gigs in nature, where a system had gotten to a point where it just simply wasn't delivering what it actually needed to do or intended to do. And I've helped teams diagnose and fix those systems. And then also taken a lot of folks through the exercise of how not to have that happen again, or at least not to have the same thing happen again. So, this is an interesting topic to me.

Kevin: Well, you're being humble, Dave, I mean, I looked into your background, you've created data strategies across R&D, as well as system information integration architectures and roadmaps. You've done it for a wide variety of clients, you've worked with early stage biotechs all the way up to the top 20 multi-national pharmaceutical companies.

So, you know, you've got the experience on many different sides. You have the experience obviously from the customer, the consultant, the vendor side of the fence, so it's great that you have that wide field of vision, so to speak.

The question I have for you, are there some common threads you've seen that contribute to system failure, contribute to systems not meeting what the goals were in the beginning. Are there some common things that people need to be aware of?

Dave: Yeah, absolutely. I think, every crisis in and around a system like this is slightly different, in things, but there are some very common root causes to issues with the system. And I guess the first thoughts really are that underlying problems with systems are frankly more often than not more related to people issues than they are to technical issues.

Now, underneath as you start to pull back things in a project, you'll many times more often than not find technical things to do. You'll find deployments that have to be altered, infrastructure that has to be changed, features and functions that have to be changed in certain ways. But the true root cause of a lot of these things are actually team issues.

And particularly in the implementation phase. And more often than not, the end result for a system, the outcome of a system not reaching its goals is lack of clarity on what those goals actually are at the very beginning.

"more often than not, the end result for a system, the outcome of a system not reaching its goals is lack of clarity on what those goals actually are at the very beginning."

And by lack of clarity, I don't mean not recognizing that our systems have to be performant, right. They have to be performant, they have to be reliable, I don't think anybody would disagree with anything like that. What I don't see happening a lot is being more specific and measurable about those things in early on stages.

So, when you say the system needs to be fast, what does it mean? Be more specific, set more specific goals about what it means to be fast. What does it mean to be reliable? What are those actual things that we can test and measure early on and during a project. Project teams more often than not, unfortunately leave those non-functional requirements -- performance, security, scalability, administrability, management, operations -- all the way to the end of the project. They don't treat them the same way that we treat our functional requirements.

And so, we might do a performance test, because we know it's a good thing to do. So, we do a performance test at some point before deploying volume testing, load testing etc. But we don't leave time in our projects to do anything with the results of the tests. I've seen this over and over and over again. People write project plans, they do a performance test, and they don't leave any room to do anything with those results. So not thinking these things through and incorporating them fundamentally, and it is pretty much the leading root cause.

"Project teams more often than not, unfortunately leave those non-functional requirements -- performance, security, scalability, administrability, management, operations -- all the way to the end of the project. They don't treat them the same way that we treat our functional requirements."

Kevin: Okay, that makes sense. And you know, we talked in another podcast about how a big part of digital transformation is really kind of figuring out the right ways to acquire our data, to get our data into these new systems, whether those are machine learning, or artificial intelligence etc. But obviously we do this because we want to get value out of the data. That value out of the data I think is one of those maybe esoteric little things that maybe aren't clearly defined in the beginning. You look and say, we're going to do all this great stuff, but you don't stop to think, well, what's ultimately the value we want to get from the data.

In your experience, do you know what maybe some common errors are, or oversights that could specifically limit the ability of these systems to get meaningful insights out of the data?

Dave: I think again, measurability, I'll come back to the comment I just made before about defining value of the data in some way that's measurable. You're not trying to be wholly quantitative about it, I wouldn't recommend that. I wouldn't recommend putting tons of effort into deep analysis and deep thought about exactly how all you know, 20 metrics that we're going to use to define whether or not our system is meeting all these qualitative goals, right, like reusability of data and the quality of data.

You can be quantitative about some aspects of it. I like to encourage teams to think through what types of proxy measures we could actually use to do the value of data. So maybe it's, how many people are searching and looking up data. So, tracking things like that in the system. How many times do people use the system to retrieve someone else's data? Not the data they generated, but the data that someone else tried. That's a relatively simple thing to engineer into a system, and measure and track that is a reasonable proxy to being able to reuse and improve the quality and usability of data. It's not going to tell you, 'did somebody have a great scientific insight' based on looking at that. That's a very qualitative thing, very impossible to measure from a systems perspective.

But spending enough time to think through some of these proxy measures is a good way to establish, how are we going to actually create the feedback loop, and is this system performing the way we want it to perform when we first started the project? And, again, making an allowance, like I said in the previous comment about performance testing. Making an allowance for changing your course, when you measure those things and you start to understand that's not going as you originally had anticipated or expected, making allowance to come back and figure out why, what can we change about this system in order to actually improve that?

So, if people aren't accessing other people's data, go find out why. Work, work, work the problem through. You can't always expect happy paths in systems. That's another critical common root cause I see in failures of systems to actually deliver is, designing your whole training strategy and your testing strategy and your deployment strategy and everything else around happy paths, only happy paths.

It's like building a house which I'm going through right now, not every day is sunny and conducive to construction. So, if you build a plan to build a house that's relying on the weather being cooperative 90% of the time, or 100% of the time, even worse, you're probably not going to be on schedule, you're probably not going to wind up where you want to be. And so, designing our systems and thinking through the unhappy paths, like common things, like what happens when the scientist is in the of middle of experimentation and hasn't produced the final results yet and gets sick and is gone for two weeks? What happens from a systems perspective with people who are waiting for the data, with all of these other aspects of things?

Relatively simple unhappy path thinking, and incorporating that into your planning and your testing and your system design strategy is another good key to actually ultimately making this system perform the way we would expect it to.

"You can't always expect happy paths in systems. That's another critical common root cause I see in failures of systems to actually deliver is, designing your whole training strategy and your testing strategy and your deployment strategy and everything else around happy paths, only happy paths."

Kevin: Yeah, I guess maybe that's part of what I was going to follow up with you on, was this whole notion of doing system audits. Can you do system audits that lead to targeted improvements or system performance before disaster strikes, are there things that you can audit and look for and identify ahead of time?

Dave: Absolutely. None of the crises I get involved with, you know, happen overnight. I mean no case ever, you know, it's been there, it's been latent, it may be even more than latent, it may be well recognized and just not acted upon frankly. Misbehavior of systems is never really a surprise, there's always something that should have been noticed that was the early warning signs essentially of things.

Kevin: We talked in another podcast I remember about, just people being enamored with the latest and greatest technology, and the pace of new technology could potentially drive the pace of efforts to serve critical data science initiatives overall, and a rush to achieve digital transformation, a rush to get to that next level, to get that next technology. To what extent do you see that pace of these efforts causing these problems, or causing system breakdowns?

Dave: Yeah, I think that's definitely a contributor. I mentioned before the lack of attention to the non-functional aspects of building out our systems. That's almost always a direct result of the imposition of a schedule essentially in some ways. And schedules are typically built from a functional point of view, almost always built from a functional point of view. We're going to build these features, we're going to have people test them, we're going to integrate them with other features, lather, rinse, repeat.

And we get to the end of those and we find it always takes more time than we've allocated, because as we build features and let people use them, we discover more things about them. That's a pretty natural way of doing, but unfortunately not generally recognized when it comes time to put the schedule together and get the financial improvements, approvals and the executive approvals on your plan and your schedule and your spend and everything.

So, all of the non-functional things that actually are very key contributors to the success of a system once it's in production, the ability to administrate it, the ability to diagnose when problems happen -- unhappy path. Bugs happen in software, they are a natural part of software, so you have to make allowances for how you're going to deal with those things in the real world.

But we spend all of our project time running towards the date, running to get something out and in use, and not actually thinking about the consequences of it being in use. And taking the time, and it doesn't take a lot of extra time, but taking the time to think through what happens after we go live, and making those plans and trying to understand as you're building the system, how better to handle that is a critical part to avoiding crises. So yeah, date driven obsessions are definitely contributors to this problem.

Kevin: I can imagine you run into similar things building your house, we need to get that in by this date, or else.

Okay, so projects, especially the ones that we've been talking about on this podcast are super, super complex. These are not -- I'm going to stick a LIMS in and we're going to turn it on, and then we're going to walk away and this thing is going to be great, and everything's going to be fine -- that's foolish to think that way. We're dealing with big complex, people driven, process driven projects.

One of these things goes south, whether pieces of it goes south, or two departments go south or whatever the end symptoms are, phone rings -- Dave, we need you to come in, we're going to try and get the ship righted. What are some of the strategic approaches, or tactics or tips when you first walk in that door and you've got a head of lab operations screaming that something's not working? What are some of the tips to get that thing going in the right direction?

Dave: Yeah, I think to me typically where I start generally for most problems, not all problems are the same, but generally with most problems we'll start with a two pronged approach essentially. With respect to system issues and stuff, one prong is definitely technology oriented. I'm a data driven guy who's trained as a scientist, so I'm a very data driven person. So, there's a whole raft of technical things on the system side of things that I want to be able to understand.

What is the technical architecture underneath this? What's the network architecture? What's the deployment architecture, database architecture, on and on. And then what's all those metrics and what's all that data actually going to tell me about how a system is actually behaving in the wild, in the real world? So that's one set of work that that gets kicked off. It's very important to this.

The other side of it though, is more towards what I was speaking about before. It's more to understanding what are the aspects of the system behavior that are what were expected, that the system may not be meeting, and where do those expectations actually come from? So, this is a softer side essentially of the problem, and it's targeted at getting a better understanding at, do we have a problem here where the expectations were actually unrealistic to begin with? And we kind of have to have to restructure that, have that kind of a discussion. Look what you think the system would do for you, it's just not on target, it will never do that. So, we have to look at the problem in a bigger sense and figure out it's not just about the system in that case, it's about everything in and around the system.

Kevin: Let me just stop you there. Do people typically have those goals written down? Like when you walk in? No. You have to go find somebody that says we deployed this multi-million dollar AI system because we wanted to blah blah blah.

Dave: The things that relate to crisis resolution essentially aren't the functional specs, aren't the design specs....

Kevin: But they won't have their expectations written down, that's something a little bit more....

Dave: Yeah, exactly. Their URS --User Requirement Specifications, everybody does the same set of paper documentation. And a lot of times that stuff is valuable, I'm not dismissing all that. It's valuable in understanding a problem at some level, but these core expectation oriented things are almost never captured in any reasonable way.

So that's a one on one kind of process with the key stake holders of the system, with the R&D leadership generally speaking about, look, why did you start this project? What were your actual goals? And again, it needs to be in a pretty concrete sense, it's not just we wanted to make life easier. Like that's not actionable, right. That's not something that you can take and sort of use as a criteria to figure out how to make the system, make your life easier. There's just too many possibilities where that will go.

And on the flip side, what you typically see in the terms of the specification documents are about the level the buttons need to be green. The fact that the buttons are blue and not green, probably has no real issue with this being a crisis. So somewhere in between is where you want to be, and that stuff is almost never written down really, not in my experience. It would be difficult to write down actually in a lot of ways.

Kevin: So, I guess when you walk in the door, you've got to be part private investigator, part data science, part scientist, part technologist. You have to have a good scope, which thankfully you do. You know, leave us off with, we've talked about digital transformation projects, how big they are in scope, how important they are, all the moving parts from the planning, to the people, to the end goal, which is oftentimes the technology piece or the end piece to it is the technology piece. Things are going to break, projects are going to go sideways, sometimes worse than sideways. Leave us off with some of your in the trenches thoughts of, what you've seen and how you can make things okay?

Dave: I think expecting things to not be all the happy path is the biggest thing, honestly. I think, we have to expect that the software is going to have bugs, that the project is not going to properly fully define all of the requirements, some of those are going to emerge as we're actually executing things. I think miss-set expectations from the point of view of the execution of a project is the biggest thing that we could do to help sort of manage these kinds of things.

Bad stuff is always going to happen, unplanned stuff is always going to happen, and even more so with large complex programs, multi system types of things, there's so many variables involved and so many different ways of approaching different ways to solve the problem etc., that there's just no predictable path through it.

And I'm not fond of a lot of these buzzwords, but at the core of the agile methodology is a buzzword, the core idea is, do what you know you need to do and then figure out what the next thing to do should be. And that kernel of thinking needs to apply to more of our projects. Figure out what you know, do that, learn from it, and then figure out what the next thing is going to do. You could do that from an overall umbrella point of view with having a time frame in mind and cost and all of that. You can use those constraints, but pre-planning, and always planning for the happy path is the big thing that I would like to caution people just to be aware of.

Expect that things are not going to go well and be able to deal with it when it happens. Look at it and recognize it, say that didn't go the way we thought it was going to go, but that's okay because we can fix it, and we can do it differently and move on. And give the people working on the project the freedom to come forward to the project management and say that ‘it's not going well, let's fix this’.

Kevin: That could apply to a lot of things in life, as could setting proper expectations. If that's probably one of the things that we leave you all with today, it's going into these big projects with the proper expectations, well defined expectations. Maybe take a cue from Dave and write your expectations down, even though they're not part of the software scopes, have a clearly defined mission statement for what your project should look like when you get to the end of it, because when it comes time to call the firefighters in, it might make it easier for them to find the fire.

So, Dave, thank you again for joining us and talking to us about digital transformation. These are obviously big, complicated projects, so it's great to have someone with your vision and expertise in.

I want to thank all of you for joining us for today's podcast and listening to it. We hope you enjoyed the content. We hope you'll pay attention to the other podcasts that are coming, and the ones that are already on the Astrix website. We're now finished up, have a great rest of your day.

About Astrix

For over 25 years, Astrix has been a market-leader in delivering innovative solutions through world class people, process, and technology that fundamentally improves scientific outcomes and quality of life everywhere. Founded by scientists to solve the unique challenges life sciences and other science-based business face, Astrix offers a growing array of strategic, technical, and staffing services designed to deliver value to clients across their organizations. To learn the latest about how Astrix is transforming the way science-based business succeed today, visit www.astrixinc.com.

THE ASTRIX DIGITAL TRANSFORMATION PODCAST SERIES

Astrix enables leading laboratories to realize their visions for LIMS implementation, providing comprehensive services that encompass:

- Business requirement analysis
- Cloud Migration
- Computer systems validation
- Configuration
- Data migration
- Design and implementation
- Computer systems validation
- Configuration
- Performance management
- Project management
- Software development
- Solution architecting
- System integration/consolidation
- Testing and transition
- User training

To learn more about why Astrix people and processes are the preferred partner for a growing number of digital transformation projects, visit www.astrixinc.com or connect with Michael Zachowski, VP of Professional Services, at mzachowski@astrixinc.com

astrix