# Lessons Learned From App Modernization Projects and How to Adopt a Winning Strategy

Michael Cinquino (00:00)
Welcome to Ascendle Unscripted. On this episode, we're diving into a fascinating challenge that nearly every successful company faces. When and how to modernize the technology that powers your business, AKA app modernization. We're joined by two leaders who guided companies through this journey. We have a newcomer to Ascendle Unscripted, very happy to have you here. Jeremy Jackson, Group CTO at ASG. Jeremy, welcome.

Jeremy Jackson (00:26)
Thank you.

Michael Cinquino (00:29)
And of course, our founder and CEO, Dave Todaro of Ascendle. Welcome, Dave. Welcome back. Of course, I'm Michael Cinquino, your curious generalist host. And as a curious generalist host, I also like to dive in. So maybe some kind of stage setting here, gentlemen, and I'll put this one up for grabs.

So the question is, imagining explaining to a business owner why they might need to rebuild their technology. What's your favorite analogy to help them understand this? I see Jeremy smiling. I set up for grabs, but I'm looking at Jeremy and I'm going to go with you, sir, if that's okay.

Jeremy Jackson (01:04)
I have my perfect analogy for this. Imagine it's Christmas and you're at the in-laws. They ask you to get the string of lights down from the attic and start to put them up on their behalf. But naturally, the lights have been up there for years. They're all jumbled up. No one's actually looked to check them in some time. And of course, you plug them in once you get them untangled and none of them work. You've got to check each light bulb individually to get it strung out.

Somewhere along that line, you're probably asking the question, do I just throw these out and buy my mother-in-law new lights? But no, they're sentimental value. They like the way they're strewn up about the house. And this is what it's like going into a company and trying to repair 18 years of tech debt, right? There's a lot of history. There's a lot of context. And there's a lot of people who've tried it before. But it's that buy-build decision-making that always comes to the forefront when I look at app modernization.

And that's the analogy I think of often when I think about.

Michael Cinquino (02:06)
Twisted Christmas lights. I love it. I don't love it, but I get what you're saying. Dave, turning to you for a moment, along those lines, what do you think the biggest misconception leaders have about modernizing their tech?

Dave Todaro (02:23)
Well, I've both been through a modernization effort myself and my previous life in a software company and worked with a lot of software leaders on their efforts. I think the big thing is that they forget that it took 20 years to get where they are now. And assuming that the product that took 20 years to build can be rebuilt with every single feature in six months. And there's a disparity there. Well, why do you think after 20 years that you can just turn around and rebuild it in six months? Well, we figured everything out already, so it should be pretty easy. It's not quite that simple. Now, I'm sure there's things where you probably would do it differently the next time around. We'd save some time, but it's very difficult to get some software leaders to realize that, we just need to take a little bit of a step back and think about if you do in fact need to rebuild everything, which oftentimes you don't, spoiler alert, it's probably going to take a little bit more than six months to do that.

Michael Cinquino (03:32)
So thinking to the 20 year, we built this over 20 years. What is the medium? That it's not going to take six months it sounds like, but it's also not going to take 20 years. How do you assess a timeline on that kind of, on that thinking?

Dave Todaro (03:50)
I think Jeremy and I both have a general rule of thumb. What's your rule of thumb, Jeremy?

Jeremy Jackson (03:55)
Yeah, well, my rule of thumb is you very rarely get through a full scale re-architecture rebuild in under two years. And worse than that is it takes another three years to customer migrate your existing customer base to new customer base. What does that mean? If you're looking at something in less than a five year time scale of a full scale re-architecture or rebuild, you're probably mistaken. You may be thinking about that first MVP to market, but you're not thinking about the full scale conversion from one workload to the next, from one revenue line to the next. And it's a pretty long time scale. But yeah, anywhere from six months to two years on the initial build and anywhere from one year to three on the customer migration time.

Michael Cinquino (04:40)
So is this why when you look at the knot of Christmas tree lights and the instinct might be to toss them, it might be better to untangle a little bit and not just toss them?

Jeremy Jackson (04:50)
That's right. That's right.

Michael Cinquino (04:54)
Got it. Can you share maybe, either one of you gentlemen, a rebuild story that went surprisingly well or maybe even terribly wrong?

Dave Todaro (05:07)
I've got one I can start with. So early on at Ascendle, we worked with a small company. And like many small software companies, it was the one person that wrote the software. So in this case, the one guy had written the product over five years. And it had gotten to the point where it met some of the criteria or most of the criteria to consider a modernization. It was built on technology that was becoming obsolete. It was difficult to get the devices on which the software would need to run. And it was hosted on computers in dusty back rooms in concrete manufacturing plants and was frequently kind of rolling over and dying. There was no cloud when it was built.

So we started working with that client. And the first thing that we did was ruthlessly prioritize, which was taking a look at everything that had been built over those five years and starting with the question, if you could only have one thing, what would it be? And we focused on that first. And so we started working on it and there was a trade show that was coming up. I think we started the rebuild in maybe August and the trade show was the following February. And we had predicted that to get to the minimum viable product that he really felt was necessary to start getting it into customers' hands, it was going to take about a year. But he said, hey, Dave, I really want to show this product off at this trade show. What can you do? And I said, well, we can get from between here and here based on how fast we're going. So let's just make sure that everything that you want to show at the trade show is above that line. And we did that.

He went to the trade show and he came back and he said, Dave, I've never had this experience before. I went to the trade show, I demonstrated the product, and first of all, it didn't crash, which is a new experience for me. It totally worked. And the other thing is that my customers, much to my surprise, said, I want that, and I want it right now. All those other things that he had talked about building for the MVP, he realized in showing it to the customers in a form that actually worked, they told him, I want it. So in fact, we shipped about four months early, which in the entire history of software, seldom happens that you actually ship early. So that was a big win for us and certainly a huge win for him.

Michael Cinquino (07:41)
Dave, I feel like this might tie into something that Jeremy shared earlier before we hopped on. Jeremy, rebuilding is the easy part, customer converting is much harder. Does that thought align with what he was talking about in that those customers went along and it sounds like the client was pleasantly surprised? And that could be a challenge for some folks.

Jeremy Jackson (08:01)
Yeah, I like to kind of distinguish between MVP and MLP, a minimum viable product and minimum lovable product. And, know, the old analogy is like, you're not going to sell a car with only one wheel, right? But yeah, there's a viable unicycle, but better a bicycle that someone's going to love, be able to get real transportation out of and be able to get to work on. It's finding that thinnest slice that is a minimum lovable product that actually functionally works for the customer is key. I find with rebuilds, they're often viewed as a tech rebuild, but they need to be a company rebuild with that customer lensing in mind first, before you even embark on that journey. And that customer lens is going to, to your question, lends itself to like, what is it going to take to get customers over the line? What differentiated value is this new rebuild going to drive that allows us to take customer A and moves it over to platform B. And that's the thing that is often misdone in rebuilds. I've had many failed rebuild stories to your earlier question, and the successful ones took the time to do the upfront work on the customer validation and that minimum lovable product.

Michael Cinquino (09:17)
So it sounds like minimum level product is really a more effective view of the product itself. Whereas for me, until you've made this distinction, Jeremy, minimum viable seems like, we're just looking at the product itself doesn't work without really the consideration that it needs, it sounds like, to make sure that it's actually a lovable, usable, really valuable product to the client. Is that an accurate distinction?

Jeremy Jackson (09:42)
Yeah, I think so.

Michael Cinquino (09:43)
Yeah. So when thinking about modernization and timing, when is a moment when folks realize or you know when the time is right to go, we need to modernize this. How do you, how do you come to that?

Jeremy Jackson (10:00)
Yeah, I can start and love your opinion, Dave. For me, I first want to define modernization. There's seven Rs typically when you're thinking about an app modernization. You can either retain the existing, you can re-host it, like take it to the cloud. You can redesign it, just re-skin it. You can re-platform it, move it from an old version of .NET to a new one. You can re-purchase it, or you can fully re-architect it, or retire it if you're done with it.

And ideally it's rearchitect and then retire. So you're not spending on both platforms at the same time. But often folks jump to that last one, rearchitect without first considering the other alternatives. So for me, I like to go through that calculus. And if we finally arrived that it really needs true app modernization that requires a full re-platform or re-architecture, that's when I start to look at the time equation where, for me, owning businesses in the P space, we have a five to seven year hold. If you're coming to me in the last two years of the

hold on a full scale rearchitecture, I'm probably a no on that because even though there's merit on the need to do it, the time value of money means that I will never recoup any sort of ROI on that because there's no ability to finish that and get customer conversion and drive revenue in that two year time scale.

So timing is everything, not just timing of your investors, but timing of the market. I've done many a rebuild that was too early to market and many a rebuild that's too late. And so when you start the rebuild is maybe the most important question, second to like, should you do it at all? And that's why I like that seven hours framework to try and frame that up.

Michael Cinquino (11:52)
Thank you for that. Dave, do you have thoughts around that?

Dave Todaro (11:57)
Yeah, so we love doing app modernizations or technology rebuilds, but the first thing I do with the software leader is try to talk them out of it. Because it really is, to Jeremy's point, it's almost at the end of that list of the seven Rs. It's again, it's easy for software leaders to fall into the trap of, we're just going to rebuild it with the sexy new technology. It's because this new development technology is going to go really fast.

And of course, the technical folks oftentimes are licking their chops because they want to get their hands on the new technology because they're stuck in the technology that was used 15 years ago when the product was started. And when we start talking about what it will really take, to Jeremy's point, it's not just getting the technology rebuilt, it's getting user adoption because you can't just drop it in someone's lap and have them move over, especially since you're not going to be rebuilding everything most likely, to rebuild everything that you've put in the product, including the things that people don't use because someone thought it was a good idea 15 years ago, but no one ever used it. That's going to take some time. So I think the first thing is let's just stop and talk really seriously about what are all of the factors that are contributing to your desire to rewrite it, other than looking at maybe some newer competitor's product that looks all sexy because it's been used, has the latest UX design as opposed to my 1998 looking user interface and getting all enamored with that and forgetting what it's actually going to take to deliver on that goal.

Michael Cinquino (13:47)
I'm starting to get the impression that from a strategy perspective, there's of course a few parts to it. The first part is asking questions that you may not have asked to begin with. And then making sure that those questions are going to align with timing. Jeremy, as you mentioned, and all the other things that can happen. And I was going to ask Jeremy, as far as how do you, you've created things that were too soon, you've created things that were too late. And I thought to ask you, how do you know the right timing?

And then I thought to myself, no, Michael, why don't you just ask him if he has a crystal ball? So timing can be elusive. Do either of you have any thoughts about timing? Some indicators, maybe some leading, some lagging indicators of how to time things out.

Jeremy Jackson (14:37)
Timing's hard. Let me reframe the question into when do you run out of options? There are two types of rebuilds, of rebuilds by necessity and rebuilds by choice. Rebuilds by choice, you need to be very pragmatic on like the market timing, the opportunity, the ROI. But oftentimes it's not that, it's the rebuild by necessity in which the timing, for timing, is not the consideration. It's like, what is the expiring clock on our end of life product and our end of life technology. So that's often how I'm looking at timing. Is there a necessity to make this decision today or this month or this year because of the end of life risk with the platform? And that is the first order of the first kind of pragmatic question I asked. Is this by necessity or by choice? It's by necessity. I want to know what our total time end of life risk is and how long we can kind of bleed out the application. If it's by choice, I want more analysis done. I want to understand the competitive landscape. I want to understand the market timing. I want to understand the investment hold. I want to understand the like even a margin and kind of the burn up, burn down chart. There's a lot of financials in the timing of a rebuild by choice, but a rebuild by necessity is usually a, again, a ticking clock that you have a window of opportunity to actually rebuild in. And that's, those are the two that I often look at timing from, the lens I look from.

Michael Cinquino (16:03)
Dave, how do you balance the excitement of a new technology with the reality of business needs? Jeremy just mapped out those two, you know, we have to do it and we're thinking of doing it. How do you assess, like you guys talked about little earlier, excited to get something new versus business needs?

Dave Todaro (16:25)
It all comes back to something I mentioned earlier. My favorite two words are ruthlessly prioritize and prioritizing to Jeremy's point earlier through the lens of the customer understanding. And now if the product's been around for a while, you probably have a pretty good understanding of how your customers are using your product. That's probably, it's the 80-20 rule, right? There's probably about 20 % of the product that is being used either by 80 % of the customers or being used 80 % of the time. What is the critical piece? And let's start there. And that solves for a few problems. One is you need to produce some pretty quick results because everybody is watching. To Jeremy's point, when you do a modernization effort, it's not a technology project. It's the entire company or the entire division, whatever it is going to be impacted by that product. It's everybody that's involved.

So the first thing it solves for by prioritizing is building consensus and excitement around not something shiny and new that looks great, but something that actually does what the customer wants and ideally does it better. Another mistake I see some companies make is to just take the product exactly as is and just rebuild it exactly as how it works today as

opposed to really thinking about, well, what have we learned in the past 15 to 20 years in terms of how our customers use our product? And let's rethink things. And another favorite saying is, when in doubt, leave it out. You can always add more, but keep it simple. Build, apply that 80-20 rule again and again and again. Take the 20 % of the product that people use and build 20 % of that functionality. And then expose it to your customers, which is going to both get them excited about the fact that you are making progress in this direction, and it's going to give you the feedback and validation that you are in fact building the right thing. The other thing it does is it lets you live with the new technology in the most important part of the product for the longest amount of time. So it accelerates risk. It might be harder to build that core of the product early on. So for example, if we were building, rebuilding our e-commerce system, I probably want to focus on checkout, not necessarily filtering the list of products. Because if I get checkout wrong, I'm dead in the water.

So if I build that most important part first, I'm going to live with it the longest. I can get the most amount of time to expose it to my customers and make sure that we are in fact nailing it. And then because usually the most important part is also the most difficult to build, I'm making sure that that technology risk is accelerated to the beginning and I can get that out of the way. And then it tends to be more of a, I'm climbing the big hill at the beginning and then I'm going to be coasting a bit down as we continue to build that product. The other last thing that that does is I may be thinking early on, I need to build the whole thing. Spoiler alert, that's probably the worst thing you could ever do because you're just going to waste money. If you're prioritizing, you can get the most important things done while understanding how long is it actually going to take. Because the other problem with software is highly unpredictable, and we don't exactly know how long it's going to take. If we build the fun things that aren't really that important, we might find out that we've run out of time, money, or both.

Jeremy Jackson (20:11)
Can I double underline that, Dave, which is the Big Bang approach never works. So when it comes to timing, if you tell me you're heads down for two years and then we'll come back with the full thing ready to launch, I am telling you no every time as an investor because I have a half dozen examples of where that's failed miserably. You got to get early market validation on that differentiating value piece of the product that helps you get conviction on maintaining the rebuild, maintaining the path. And to Dave's point, like validating the teams come up to speed, learning at the right level and the technology we've chosen is the right technology. And so that point is so salient to me because it's the one that's missed often, which is we're just going to go heads down and build the whole thing over two years.

Michael Cinquino (21:00)
So Jeremy, while the first rule of Fight Club is you do not talk about Fight Club, the rule of microservice sounds, services sounds like you don't talk about microservices until you can articulate the scale challenges. Can you talk about that a little bit?

Jeremy Jackson (21:19)
Yeah, I think there's a moment in time where with all things like technology as fads, but there's a microservice fad where everyone wanted to convert or build anything new and microservices. And while on in premise, that makes sense for large scale applications, call it like 20,000 concurrent. Most applications aren't that. So you don't actually need to go straight to the newest and hottest technology and the newest architectural approach.

Um, and so oftentimes, yeah, my, my pithy saying is if you talk to me about microservices before you can articulate the scale challenge, I'm going to throw up all over it because, uh, most people don't have basic observability. Most people don't actually know how many users are going through their site or what their actual service architecture constraints are or their data constraints, is where the constraint usually is. And so I find there's a lot of tech for tech sake in our industry.

And what I encourage folks to think about is just think more broadly about the incremental scale challenges and when you can always add microservice later, you don't have to start from that point of view. So think about the appropriate architectural setting for your recommendation and at what scale, like you can have a now next later plan that shows microservices in the later, but the now is just working on maybe a shared authentication mechanism or a login or something more central to a replatform or rebuild. And so that's the thing I often stress our engineering leaders on is like, don't go tech for tech's sake, don't go microservices until you can truly articulate the scale, the constraints and the build path that gets you there.

Michael Cinquino (23:05)
Jeremy, thank you for that. I eventually want to close on both of your insights and your views on how AI is going to change the approach to rebuilds. But before that, I would love to hear both of your takes on human elements that might get missed. And I'm not going to ask a specific question, but when I say human elements or human interactions, what comes to mind for both of you in this process, the process of rebuilds, of adjustments?

Jeremy Jackson (23:36)
Yeah. Well, I'll start by saying every failed rebuild I've encountered shared the same pattern, which is we didn't have a team with high conviction in the rebuild. We didn't have a team with experience in the rebuild, and we didn't have the right makings of the team. And by that, I mean the right counterbalance of new and old, a team that could sustain the existing product and a team that was primed to build the new.

And so I look for those three things when I hear pitches on rebuilds on app modernization is I look for, do we have a leader and team who has strong conviction in the plan and is going to doggedly pursue it? Have we thought about the team dynamics in terms of the overall structure to support the team? And, you know, do we have a partner who can help us advance the ball? Think strategically, bring new patterns of development to bear. And

those are the things where I want to see a kind of mixed calibration of team on a new rebuild because as it always goes, right, we'll got you here, won't get you there. You can't rapidly overnight accelerate an end team to learn a new technology. You're going to have to bring new folks to bear, new partners to bear to help you get to fruition.

Michael Cinquino (24:59)
Jeremy, thank you for that. Dave, human element.

Dave Todaro (25:01)
I think there's two major areas, just to underscore what Jeremy said. The ability of the team, even though the team might be using contemporary technology and contemporary tools, which do in fact let you move more quickly, if they're not familiar with that technology or those tools, they're not going to be able to reap that benefit. So getting some folks that are familiar with that and working shoulder to shoulder with your current team as they upskill.

Your current team knows the product inside and out. And you can't hire for that. You're not going to hire somebody off the street, whether it's an employee or a partner. And they're going to understand the domain of the product. But if you couple the folks that know about the newest technology and tools with the folks that know a lot about the product, that's where you can definitely accelerate. think the other thing is that, coming back to the software leader, think that oftentimes,when the decision is made to do the rebuild, they think their job is done, but in reality, it's just beginning. What I've seen from a failure standpoint is that software leader stepping away, and then the rebuild is happening, and then all of sudden they look at what's happened over the past three months and they say, what is going on here? Why are you guys working on that? What's going on over here?

In one extreme case, we worked with a client where the team had been working on login, and then they didn't really like how it came together, so they rebuilt it in another login technology. They didn't like that, so they rebuilt it again. By the time I had talked to the software leader, I said, Dave, I think something's going on. They've been working on login for 14 months. I said, well, what else have they built? He said nothing. So he had distanced himself.

If he had been showing up every two weeks and asking those hard questions along the lines of, I mean, I understand logins important guys, but I mean, that's not really a differentiator in our product. Everyone has login and it works the same way. When are we building the stuff that I can get my hands on that's going to in fact be different and better for our customers? So those are the big things, making sure that the team is empowered and they have that shoulder to shoulder technology and tooling support and making sure that as a software leader, you stay engaged and as the saying goes, inspect what you expect. Not in a micromanaging way, you're not coming along and telling everybody what to do every day, but just understanding what exactly is going on every couple of weeks and ensuring that the product is coming together and that ruthless prioritization is in fact

happening. I'd rather show the basics of the core of my product to somebody without any login and get some feedback while we're going to solve, maybe we come along and solve login a few weeks later, then building login, which I can't show to anybody because they wouldn't know if it's my product or somebody else's except for the logo.

Jeremy Jackson (28:12)
If I can add to that, Dave, the other trope is, okay, we brought the MVP to market, then they step away. And so the job's not done until the revenue is converted, until you've actually finished not just the rebuild, but the build out the backwards compatibility and converted customers. And so I see people tagging out there when they've gotten approval, tagging out there when they've launched the first MVP and tagging out before we even got real revenue attributed to it. And then back to the people, we've been focusing our framing on the engineering teams and leaders, but as I said, rebuild's app modernizations are a full company plan. And the place that it falls over is when you haven't been thoughtful enough about incorporating marketing and product into the conversation, building product marketing collateral before you bring it to market, really driving sales with arming them with what the differentiating value is and how to articulate it, finding focus group customers to test it. And this blends the product and tech world, but it is a challenge when I think about people and teaming concepts that we think as engineers as our first team is the end team. Well, the first team is your executive team and ultimately, you're all serving the same customer and that is the thing that you can't lose sight of in any app modernization effort.

Michael Cinquino (29:36)
Jeremy, thank you for that. I'd to stay on you for a moment and talk about the future. The rapidly changing approach to rebuilds via AI. So I'd love to hear Jeremy, your take on AI and how it's affecting the approach. Dave, you as well, of course. So Jeremy, how will AI change the approach in 2025, 2026, and maybe even beyond in your view?

Jeremy Jackson (30:04)
Yeah, it wouldn't be a conversation if we didn't have AI in the mix. For me, I've had the pleasure and pain of seeing almost two dozen rebuilds and replatforms. And what has been historically true? Historically, it took two years and $2 million, call it, as a barrier to entry to build new things in market. What I think is going to be true is an erosion of those economics, an ability for someone coming in, we call it a quarter million dollars, to get Hi-Fi design prototypes, test in market, build it with AI coding companions, and get it tested and infrastructure deployed all with an army of AI servants, thus lowering the cost to bring you something to market. And that is true. Today, you could come in, use UX pilot today, create Hi-Fi prototypes of a full 16 step onboarding flow in five minutes time, where it used to take a designer six months, a full focus group. You can get a very senior engineer who understands architecture concepts, pair them with Super Maven or quad or a GitHub co-pilot, and they can build out full service architecture, build out unit testing in the same code commits and double, triple their own personal velocity as compared to just a year ago.

And then with infrastructure, they now have AI programs that allow you to take your code and it'll create infrastructure's code for you, model out what the infrastructure deployment should look like and deploy that in, without a DevOps person, without a team, all in the, in the span of, of, know, a couple of weeks. So what I just described was six months of effort with six different parties involved that now you can do single-handedly if you're savvy enough with these army of AI tools and go from concept to design, to iteration, development, to deployment in the span of a week. And that is pretty impressive. So I worry about that on the whole. What gives me some conviction as an investor that that's not going to happen tomorrow is there aren't that many savvy people in the world that know all aspects of that. You still need to have some design sense to manage your AI design. You still need to understand the core architectural tenants to manage your development and you still need to know what good secure cloud deployment models look like to manage that. So there still requires a lot of oversight, but it's less than a few years ago and less than even a year ago. And I think that ball will only advance from here on what I'm calling the modern STLC across all phases of development, thus lowering the costs and barrier to entry, thus creating a ripe opportunity for a lot of people to disrupt new markets.

So that's the techno-optimist. The techno-pessimist in me says there's been ways of this before and none of them has accelerated quite as fast and as hard as we thought they would. So I think that this too will take time and we shall see.

Michael Cinquino (33:23)
Jeremy, thank you for that. Dave, from where you sit, what do the next couple of years look like with AI and rebuild?

Dave Todaro (33:33)
Yeah, I think the AI revolution, to speak, AI has been around for 50 plus years, but with generative AI, everything has changed, as we all know. The easy stuff will get easier and faster. I think the challenge, to Jeremy's point, is that we can't replace the smart stuff, meaning we still need those folks that understand product, that understand architecture, that understand integration with other products. Bottom line is that we have years and years and years of technology that we need to typically integrate with. Most contemporary applications don't live in a bubble. They have to integrate across an ecosystem of other products. And I think it just emphasizes the importance of the strategic side, because I think what AI will allow you to do is move extraordinarily quickly in the wrong direction if you don't pick the right direction to go in. I think what will be more emphasized is the product folks, the user experience folks. It was interesting - I watched a YouTube video about someone who was experimenting with these new UX tools. And he said, really what this has allowed me to do with this UX tooling is move from a designer to a producer. So it wasn't that he was just some guy off the street that didn't know anything about UI UX. It's just that he was able to elevate himself and get the easy stuff delegated to an AI tool and allowing him to focus more on the hard stuff that really required his level of expertise.

So I think that we're going to be able to move more quickly. I think the other exciting thing is, especially with a legacy code base, leveraging UI tools for analysis. Like, let's analyze this code base and understand more about it, which will help guide what we need to do moving forward. And I think where you can also use AI, if you do have the right instrumentation in place, you can use AI to analyze the data and say, all right, given this data set about how my product's been used over the last six months by my current customer base, if I was to rebuild it, what's the 20 % that I should build first? And how would I prioritize it based on the actual usage of the product? So are we in the mode of where I can take my 20-year-old product and snap my fingers and run it through an AI, and it's going to spit out a fully developed replacement in a couple of days? Not quite yet, maybe in a few years. But I think that when we leverage those tools with the correct strategy, it can accelerate our time to market. And to Jeremy's point, it doesn't count until we have revenue associated with the effort.

Jeremy Jackson (36:33)
Yeah. I like the framing of in today's world, you can deploy an army of AI interns, but that doesn't negate the fact that you still need thought leadership strategy and experts to guide it. Cause interns are going to intern and create things for you. Not all of which are going to be great.

Michael Cinquino (36:53)
Gentlemen made me think of a curve ball question here. Are you ready? So it sounds like the creative application of AI is very valuable. Like you're saying like, I can do this now. But if you've never really done it without the AI, what's really your level of creative capability and understanding how to get the most leverage from it, to use your words, Dave. How do teams or leadership continue to leverage that AI, you that they haven't had access to in the past, you know, I've to do the job on their own in a way. But how do they leverage it at the same time they don't degrade their skills at the same time? And the analogy that I would give is the planes fly themselves, but we need those pilots landing the plane without the AI every so often. So their skills don't degrade. Is there anything that you gentlemen can think of or recommend to that two pronged approach of making sure you're leveraging it and getting what you need out of it, using your experience and your repetitions that you have, but at the same time, not finding it like losing, degrading those skills that are going to allow you to leverage the AI.

Dave Todaro (37:58)
Yeah, I think that's one of the tricks. And as we talked about in software circles, well, AI is going to allow senior people to be more effective. But how are the senior people going to be replaced? Because if we don't have any junior people writing the easy code by hand, how are they going to accumulate those skills over time to become a senior developer that can then leverage the AI tooling? So to me, that's not a problem that's been solved yet.

But I think to your point, Michael, there are some things that we do in software and coding that are very routine and don't leverage a lot of brain power. Just a simple example, when we write a test to exercise some part of our code, usually coming up with the basics of the tests might require some insight from me. But then there's 14 other ways that I should run this test to make sure that all the different ways this code can run is in fact working. Do I really need to come up with that on my own? That's not really leveraging a lot of brain power for me. It's certainly leveraging experience as I think through all the ways this code can break. But that's a perfect application of AI. Hey, I've given you this model. I have this procedure that I need you to test. I've come up with one scenario, suggest to me the other 14. And then I realized there weren't 14, there were 23. And the AI can just write that code. And so the question is, do I still need to be able to do that? Being able to land a plane if automation fails is a much different challenge. Being able to come up with those 23 use cases or scenarios and then tediously write all of the unit test code to do that, I'm not sure that anybody needs to do that any longer.

Michael Cinquino (39:50)
Powerful distinction. Jeremy?

Jeremy Jackson (39:52)
Yeah, I like to think of AI today as a pair programmer, uh, doing for you what you don't want to do in writing in putting in documentation and doing these other things. And I think there's a lot of perils and there's a lot of promise to AI. And with my portfolio, at least with 700 plus developers across multiple dev teams, we've just been focused on this train the trainer model of like.

Document those promise and perils document those areas. It's good at those areas where it's weak, those areas where you should own it versus the copilot and put that into the typical poor crust process. The reason why I trust AI in this fear is because it's a human intervened process, right? You may develop with an AI, but some other human is going to review the poor crust review and sanity check the work before it goes into production.

So I love the closed loop cycle there and the fact that there is, stages to validate the usage and the pragmatism of AI and one use case over another. Over time, I still lean to the place where you gotta get people adopting it, using it, and training them in the ways to better use AI, train them in advanced, you know, end shot prompting and prompt engineering and train them in ways to make the AI better. Because the answer is if you don't, someone else will and then they will beat you to market and they will be more competitive. And so I try not to think of the ways that it will fail us as much as I try and think about the ways that it can accelerate us if done in the right way, if done in the right, you know, closed loop environments where we have human intervention. And that will always be true.

But I do worry to your point, Dave, about our junior engineers, the folks coming up now, it's just a very complex and dynamic environment to be a young engineer. I want to create

avenues for those folks to learn both ways, the old way and the new way of work, because work's always changing, the dynamics always changing.

Dave Todaro (44:32)
I think the big thing is that just be deliberate and take a step back when you're thinking about rebuilding your product. It's not something to be taken lightly. So there's absolutely times where rebuilding your product makes sense, particularly when the current technology is just rolling over. To Jeremy's point, that's the easy time to think about it. There's also other opportunities to chase down that don't require you rebuilding your product. I think the other thing to remember is that there's software that doesn't look sexy and new that works just fine and customers love. In fact, they would tell you that you will pry it out of their cold dead hands. Do not take my product away. So you might look at it and think, well, this is kind of 2000s looking and I really love this to be all sexy like an iPhone, but there's no actual problem. You are inventing a problem that doesn't exist. So just be cognizant of the fact that your current product might be great. It might be an opportunity to look at other things that you can do that then augments your product. The other thing to keep in mind is that unless you are leveraging a separate team while you're rebuilding that product, you're working on getting back to the place you are today. So you're not actually making any progress in the customer's eyes. So you've got this new shiny tool that doesn't have all the functionality they use today, but it looks new to them, you've actually gone backwards. So why or what is the argument? What is the business argument? What's the revenue-based argument for actually making that decision?

Michael Cinquino (43:50)
Jeremy, any parting thoughts?

Jeremy Jackson (46:19)
Yeah, I just like to lens technology challenges in this order. People, process, and then tech. And for me, before evaluating any decision criteria, do we have the right people at the table making these decisions? Are they adhering to the right processes where if we add in new tech, new dynamic, new challenges to the fact that we're going to keep the same pace, continue to develop? And if you don't, I challenge you to go look at that first, go look for a partner, go look for someone who's been there, who's done that, because that is where things fall off quickest. It's not the technology being thrown into the mix. It's that you didn't have the right people and team structure and you weren't following the right processes to get work developed, let alone adding in AI to complicate those processes. And then when it comes to tech, I think we've talked to kind of a nauseam of the various decision criteria and things you should think about before going through a full-scale app monetization or a platform. But for me, it's always last. And you just have to remember starting or getting to that decision point is not delivering it and converting the revenue. so know that you are signing yourself up for a long journey and make sure you are equipped with partners and folks who can help you achieve it.

Michael Cinquino (45:14)

Jeremy, thank you for that. Jeremy Jackson, Group CTO ASG. Dave Todaro, founder and CEO of Ascendl. Thank you, gentlemen, for being on this episode. If you're viewing us on YouTube, hit that subscribe button if you like what you saw to get notified when new episodes come out. Until next time, we'll see you on the next episode of Ascendle Unscripted.

Dave Todaro (45:34)
Thank you, Michael.

Michael Cinquino (45:36)
My pleasure.

Jeremy Jackson (45:36)
Cheers.