Ep #76: How Scrum Lessons Can Improve Your Law Practice: Kanban for Lawyers Part 2

Full Episode Transcript

With Your Host

John E. Grant

Ep #76: How Scrum Lessons Can Improve Your Law Practice: Kanban for Lawyers Part 2

Back in the early 2000s, software and technology teams around the world underwent a major change in the way they conceived, built, tested, and delivered their software products. And this was largely due to the innovations from the Scrum methodology, which was an alternative approach to project management and product development and is probably the best known of the Agile methodologies under capital A Agile.

In today's episode, I'm going to give you some background on Scrum and give you three important things you can learn from it for improving the flow of work, the communication dynamics, and ultimately the efficiency of your law practice.

You're listening to the *Agile Attorney Podcast*, powered by Agile Attorney Consulting. I'm John Grant and it is my mission to help legal professionals of all kinds build practices that are profitable, sustainable, and scalable for themselves and the communities they serve. Ready to become a more Agile Attorney? Let's go.

Hey everyone, welcome back. So last week, I did sort of a deep dive in part one of this two part series. I went deep into the sort of Lean and Lean manufacturing origins of what we're doing in the Kanban method today. And all this is hopefully building for you, and it's certainly I'm trying to build for me, up to why some of these tools and practices work so incredibly well in legal practice. Although, I think they work better when you know some of the whys and some of the origin stories. So that's what I'm getting to.

And if you didn't listen to that episode, that's fine. Listen to it whenever it makes sense to you. I will say the three core takeaways of sort of the Lean manufacturing origins of Kanban and the Kanban method, just to reiterate, number one is the power of using visual signals, right? These visual tangible representations of work make invisible work visible. They also provide important cues even when we're talking about physical things like manufacturing or Kleenex or cash register tape.

Ep #76: How Scrum Lessons Can Improve Your Law Practice: Kanban for Lawyers Part 2

Number two is the problem with partially finished work and the sort of inventory of stuff that has been started on but isn't yet finished and what that does when it starts to stack up inside of your factory building or inside of your law practice. And so one of the reasons we use the Kanban method in conjunction with all of this stuff is to try to make sure that work is flowing as smoothly as possible. And, as you all know in legal, that's not always possible to make it flow completely smoothly, but we're constantly trying to get it to move past these sort of in-progress or these stuck stages so that we're focused on delivering value all the way at the end of the process, whether that's the done column on your Kanban board or whether that's the shipping dock of your factory floor.

The third takeaway is the importance of a pull based system as opposed to a push based system where we're not just letting new work into the door, into our law practice, into the factory floor because there's demand available. We're actually limiting the work that we take on relative to our capacity to finish that work and get it across that finish line. And in doing that, we avoid lots of the problems of this partially finished work stacking up, giving us things that we trip over or have to track or these other administrative hassles that don't really add value to the work either through the lens of the customer, but more importantly, it's not work that you can receive value in exchange for doing. It's just overhead. And we want to minimize the overhead costs of the processing of your work as much as possible. So, go back and listen to that episode if you didn't already, but you don't have to do it before you listen to this one.

Where I left off last week was this sort of transition or this idea that came originally out of software programming and these proto Agile and eventually Agile methods that very smart people looked outside of the domain of software to other tools and practices in order to find things that were going to make this particular version of knowledge work better. And the Kanban board became a really important tool, originally in the first core Agile methodology that gained a lot of traction is something known as Scrum.

And I realize I'm throwing a lot of jargon at you. I'm going to just use the jargon. As I tell people, yeah, some of the words are scary or weird, but at least if I tell them to you in the actual words, then you know how to go Google it and look up the Wikipedia article and figure out how to dive a little deeper if that's what you want to do.

Now, Scrum is interesting because there are several things going on behind the scenes or under the hood of the Scrum methodology that even though I actually don't think that of all the Agile methodologies, I don't think Scrum is the best one for most law practices most of the time. There still are some important lessons from Scrum, and so I'm going to give you a little bit of a detail or history of the Scrum method.

Now, one of the things that the Scrum method really focuses on, and this is sometimes hard to do in legal, and it's hard to do on a couple of levels, but a Scrum team is generally a single purpose team. So it is a group of people that work together to get a particular, in this case, software feature or piece of software all the way from conception to completion and delivery without having the administrative overhead of having separate groups or separate teams that are doing separate parts of the work. And so the best way to maybe describe it, right, the typical piece of software has to go through a few stages in order to get done. There is almost certainly a design or a graphic design or a user experience design stage where there are certain people who are professionals and are trained in the art of figuring out how are people going to actually use this feature in a piece of software.

Then there are the coders, the developers, the software engineers, the people that are going to actually, in whatever programming language they use, are going to write the lines of code that achieve both the look and feel that the designers have sort of given it and also the functionality that the design is meant to empower. And so that actual coding, I think of it as kind of akin to the drafting stage of a lot of legal workflows.

Then there's actually two other roles after this sort of development stage, and the order can depend on how the team likes to operate. But there's usually a quality assurance or a quality control phase. And in the world of software, QA, quality control, that is a standalone discipline. It is a profession in and of itself. And they write all sorts of different tests and scripts and different things trying to anticipate all the different ways that a user might use this particular piece of functionality and also all the ways that piece of functionality might break or might not work for the user. And they're trying to obviously get out ahead of any potential user problems with the software tool. And so this quality assurance, quality control phase of work is really important in terms of making sure, did this new function actually accomplish the things that it is meant to accomplish, and is it sort of usable? Is it done in a way that people can actually sort of wrap their heads around?

There's another important part of the quality control process and this is a little bit tied up with the fourth phase that I'll talk about in a few minutes, but that's making sure that this new piece of code that is usually going to be put inside of a larger software product or webpage, whatever, and so another function of the quality assurance team is to make sure that nothing in this new piece of code breaks some other part of the code. And so this idea of quality assurance and software testing is sort of an iterative process and a lot of the scripts that they develop actually wind up getting reused over time to make sure that something you do in iteration 50 of the software doesn't break something from iteration two or three.

And then that last role that I hinted at is sort of this integration function or this sort of go live piece, which is, okay, we've written this piece of code, it does what we think it's going to do in a vacuum, in a test environment, whatever. Now, how do we actually roll it into the larger piece of software so that we can actually roll it out as a function to our customers, to our users? And each of those roles is a standalone discipline. And sort of in old

style software development, each of the people who are involved in this process would be part of a different team.

So there would be a UX team or a design team. Then there would be a coding team. Then there would be a quality assurance team and then there would be an integration team. And what they found is that there was a lot of what I will commonly refer to as fence chucking, right, where I've done my piece of the work. I'm going to throw it over the fence to the next person down the chain and it's their job to sort of figure it all out.

And what the originators of Scrum realized was that was a suboptimal way of working. Number one, because nobody really had ownership in the success of the final product, right? As long as you design something that looked pretty, yeah, whatever. It's up to the developers to figure out how to turn it into code. Same thing on down the line with quality control integration, etc.

But when you put those four roles in the same team and they're all focused on the same sort of chunk of code or particular feature that they're trying to deliver, now they are incentivized and sort of empowered to get themselves aligned up front and have those communications in a way where maybe the quality control person says to the designer as part of the planning phase, yeah, that thing that you're trying to design, you know, we've seen that before and users have trouble with it, or calling this particular function, we don't maybe even know why, but it breaks this other piece of code. And so we want you to avoid this.

And what it does is it, number one, accelerates, even though it takes a little bit more time and effort up front to do that planning and to get those, in this case, four people on the same page, the time you invest in getting those people on the same page pays huge dividends once you actually go to create the work, deliver the work, get it live, because you've anticipated a lot of the problems and you've been able to sort of get on the same page,

get all your oars in the water at the same time, whatever metaphor works for you, so that we're now delivering a feature that works both for its intended purpose and also inside of the context of the larger software tool.

Now, once you have that team together, the sort of next thing that started to emerge was this notion that, oh, this team has finite capacity. And you hear me talk about this all the time, but the team itself can only develop and deliver so much work in a given chunk of time. And one of the other interesting things about the Scrum process, and then this sort of translated to Agile processes more broadly, is that it flipped the script in terms of the relationship between scope of work and the amount of time and resources it takes to deliver that work.

And I'm going to backtrack just a minute because in all sorts of project management, and if you ever take a project management course, one of the first things that you'll learn is this concept called the Iron Triangle of Project Management or the Triple Constraints of Project Management. And what this iron triangle basically shows us, and it's usually a graphic, each of the corners of the triangle is labeled, and actually the middle of the triangle is labeled as well.

So the middle of the triangle is usually labeled with a quality product, right? So we want to maintain a high quality product, a high quality release. And the way the triple constraints work is that if you're trying to maintain quality, then there's three different variables that come into play, but if you're going to adjust one variable, then at least one other of the variables, and sometimes both of the other variables, also has to move or has to float. And so the three corners of the triangle, number one is time or duration. Number two is scope, or what is the thing that we're actually trying to deliver. And then number three is sometimes money, sometimes resources, but kind of getting at the same thing, right? How many tools or people or whatever do we need in order to deliver this work?

And so taken all together, in order to build a high quality mousetrap that has a certain sort of specification or a certain scope, you need a certain amount of time and a certain amount of resources. If you need to deliver the mousetrap faster, you have two choices, broadly speaking. And both of these choices don't always necessarily work, but one of the choices is to reduce the scope, right? To make the mousetrap, make the widget simpler. And by reducing scope, then you can generally produce a product in less time.

The other is to add resources to the team. And that's the one that doesn't always work, right? Doesn't matter how many moms you put into the room, you can't birth a baby whatever you want to birth in less than the amount of time it takes for that animal to gestate a baby, right? So it doesn't always work that you can add resources, but sometimes it works that you can add resources. You can add another developer to the team, you can add another tool into the manufacturing line, whatever it happens to be.

Right. And all three of the points of the triangle can be adjusted, right? So if someone comes in and says, hey, we got to cut budgets and your team is now smaller, then one of two things is now true. Either that team can produce less work, right? The scope has to go down, or it's going to take longer for that team to deliver the same amount of work, right? It's an iron triangle. Something else has to give. You can't just adjust one thing without also adjusting some of the others.

Now, in traditional project management or what sometimes is now called Waterfall project management, the way sort of the assumption of the managers in a business or on a team or whatever sort of unit of workers you're talking about, would basically say, hey, we're going to do a new release of this software product, right? Windows 95. And we need it to contain all of these features. And there would be this sort of massive exercise in visioning and envisioning and incorporating user feedback,

incorporating customer wants and needs in terms of what they're looking for from a product.

And all these different folks would sort of do this horse trading around what should be in scope. And they spent lots and lots and lots of time worrying about scope. And then they would go to the developers, the programmers, the people responsible on the delivery team and say, okay, here's the scope, how long's it going to take? And then again, there was all this massive investment in scoping and hypothesizing and trying to be able to say, okay, well, we think this is going to take about this long and eventually they'd come out and say, okay, we think we can build this thing in nine months, whatever it happens to be.

Then there were some other funny things that would happen, like just normal human psychology as a result of this. Right, the developers who are the ones that ultimately are kind of going to get held accountable to these estimates, they would sandbag, right? And it was a natural human instinct. So if they thought it was going to take six months, they were going to tell you it's going to take nine or maybe 12 because they knew that the estimates are really only guesses and that you never really know how long something's going to take until you dive into it and start to figure it out. So they were hesitant to give short time estimates because they didn't want to get caught out on having gone over their time budget.

Then sort of the leadership of the company would say, well, we can't possibly take that long. How can we deliver it faster? And they'd go back to the scope people and say, you got to cut some scope. Well, because that became a pattern, what the scope people learned is that in the first iteration of or the first sort of time that you go present a new piece of work, you also inflate the scope. You put in way more things than you actually want or need because you know leadership is going to come back and cut something. And so you want to throw a few sacrificial lambs into your initial scope request so that when that inevitable time comes back where they

say, hey, you got to cut scope so we can meet this launch schedule, you actually have something to cut.

All of this, as you can maybe start to cotton on to, is a huge amount of administrative overhead. Like we haven't delivered anything yet. We're just doing this sort of back and forth horse trading, estimating type work. It's necessary under the model in order to get the work delivered, but it's not a great use of anybody's time because you're not really satisfying a customer need or a customer want. You're just talking about those wants and needs.

And then the other problem, or one of the other problems, there were many that would come into play sort of after this initial round of horse trading happened and we actually said, okay, great, we're going to freeze the scope and we're going to start development. And again, we're using this old linear timeline where there's a design phase and then there's a development phase and then there's a quality control phase and then there's an integration phase. And each step along the way, it was sort of this chunk of work got chucked over the fence to the next team, right? There wasn't a lot of integrated conversations.

And in fact, it kind of devolved in a lot of situations into something of an adversarial relationship where there was a lot of finger pointing between the teams, especially if work started to be late. And again, part of the manifestations of this problem is that the designers were kind of sitting pretty because they were at the front part of the timeline and so if it took them a little bit longer to get to a design, or maybe they'd present a design and the managers and the powers that be were like, oh, that's not quite right, and we want to tweak it here and there. Everyone felt like, oh, we can do these tweaks in the design stage because we've got plenty of time on the timeline. It's not that big a deal. Yeah, we're another week, another two weeks, another month, right? It becomes a little insidious, but in the design phase, right, which is kind of partly the fun phase for people to look at anyway, we tended to overwork the problem.

And so then it gets to the developers. The developers dive in and like, oh, okay, even though we sandbagged our estimates, this is even more complicated than we thought it was going to be. It takes a little bit longer. You got to figure this thing out. We got to research this tool, whatever it happens to be. And so now the development has taken longer and now all of a sudden the timeline's in trouble.

And the thing that kept happening over and over and over again when the timeline gets into trouble is the part of the process that gets squeezed is quality control. And which means that now you're launching software because you have to do the integration, right? You got to get it into the product itself. That's sort of non negotiable. And so they would launch these software products that were just full of bugs and full of problems. And any of you who are, you know, Gen Xers or older, maybe a lot of you millennials probably remember, like you didn't want to be, and this is still true today sometimes, frankly, you don't want to be on the first version of a new product because it's going to be buggy. And the reason it was buggy was because QA is the one that got pinched.

And so the way that Scrum, right, again, these original Agile methods sort of turned this on its head and tried to solve for it and did a really pretty good job of solving for it. It's not perfect. It's got its issues, but I think it's on the whole a massive improvement over the original status quo is they said, number one, we're going to get rid of these siloed teams and we're going to create this integrated delivery team that is inherently more lowercase A agile, right? They're more nimble, they're more responsive.

Number two, we're going to do a different way of approaching this iron triangle of project management. And we're going to say, instead of estimating and sort of doing all of this exercise around, okay, what should be in scope for this release? We're going to say, no, we're going to actually fix the time frame. We're going to engage in concerted effort of work over the course of a short period of time, and we eventually started calling that

period of time a sprint. And a sprint for a Scrum team is typically between one and maybe four weeks, but I would say the most common increment of time is usually a two week sprint.

And so we now have the resources fixed, right? Because we've established the team, we know that's fixed. We have also now fixed the time by saying we are only going to work on this for two weeks for the length of our sprint, which means that the only thing that can float is the scope. So now, instead of the question being how long will it take you to deliver this scope, the question instead is how much scope can we deliver in two weeks?

Now, that in turn does two important things. Number one, it is fixed capacity. You're saying, okay, we're forcing this honest reckoning with capacity. We can only do so much scope. And so going back upstream to what are called the product owners or sometimes product managers, it's like, number one, how can you sort of break down the work into these manageable chunks so that a particular Scrum team can actually deliver some incremental unit of value inside of one of these two week sprints. And sometimes they got so good that they might break it down and there would be two or three chunks of scope that could be delivered in a sprint.

Sometimes there would be something that was bigger than a sprint and they'd have to break it down and say, okay, well, we're going to have the term they use is an epic, but it's just a word for a bigger project. And we'll say, okay, we've got this epic that is actually going to require three or four or maybe even 10 sprints in order to deliver the full thing, but we're going to engage in a work breakdown structure, which, that's a concept that isn't new to Scrum, that was in project management to begin with. But we're going to do this work breakdown structure. We're going to boil it down to these incremental pieces, then we're going to deliver each one of these pieces.

The other thing that it then did though, is once we've said, okay, we have finite capacity, we can only take so much scope, is it forces the product owners to prioritize which features they believe are going to be most important or most impactful for the user, for the customer. And so we're trying to really make sure that we're using the Scrum team, right, this finite unit of capacity to its highest and best use or in a way that is going to generate the best return on investment for the limited resources that we have. You may have heard me talk about iterative development, right? We're developing in iterations instead of as one big bang. That's what I'm talking about.

Now, the third thing that comes from this short delivery timeline, right, these sprints, is number one, we try to be really focused, and I apologize, I'm numbering a lot of things, and I've probably lost track of the total order. But with respect to this particular topic, we're trying to deliver a usable piece of work, something that we can actually point to and says adds value and something that a customer can use, a client has asked for, whatever it happens to be. And like I said, it's not always the case. Sometimes we do have these bigger things that we have to chunk out and we don't deliver the actual value until all of the parts of the epic are complete.

But in theory, at least, we're trying to deliver something that the user can use every two weeks. And you almost certainly see this process in play as the apps on your phone update, right? So almost every developer today is using some version of Scrum, some version of iterative development. And the reason why your Spotify app, your Google app, whatever tools you're using on your smartphone, the reason why they seem to update every one or two weeks is because they're on these Scrum cycles, these sprint cycles. And they're delivering incremental improvements to the user on this very regular and consistent basis.

Now, in delivering the work, we actually are now also creating a feedback loop. And so despite the best laid plans of the sprint team and the quality

assurance team, right, you don't really know how something is going to perform until you put it out into the wild, right? There's the great Mike Tyson quote about everyone has a plan until they get punched in the mouth. Right? And so these are all the best laid plans, but actually releasing the software into the world is oftentimes the punch in the mouth. And so you find out how it actually performs. And most of the time it performs as designed. It performs well and you get to feel good about it and you can patch yourselves on the back and you can say, yes, we just accomplished something important because we delivered a unit of value to the customers of the business.

But sometimes we find out that it doesn't work, right? And maybe despite our QA tests, we did break something, or maybe despite all of our user design, we made something that users can't wrap their heads around or can't figure out how to use correctly. And you want that feedback too, right? You don't want to delay the amount of time that it takes in order to get this information back to the team. And the other good thing about this sort of sprint approach or the Scrum methodology is that let's say that the release is an unmitigated disaster and you wind up having to roll it back and pull it out. Well, the nice thing is you've sort of capped your downside because you've only invested, I mean, yeah, it sucks to like waste that two week sprint, but you've only lost one two week sprint. Like that's a manageable way or manageable sacrifice in order to learn that, oh, we really missed the boat on this one. We've got to figure out a different way to deliver it.

And so by iterating, by focusing on delivering small units of value, all these great things happen, right? A lot of the overhead cost of the communications starts to go down and the horse trading that used to happen. The inevitable optimism bias or worse, the little sandbagging things that people do that are part of human nature, when we're trying to have these discussions and do this horse trading, they go away, or at least are reduced. They don't go away completely.

And then we get in this habit of doing consistent and regular delivery of increments of value to the customer. And that's something customers like and notice, right? It's like, oh, this is getting better all the time, right? And I don't have to wait between Windows 95 and Windows 98 in order to get a piece of functionality I want, right? I can actually sort of get this thing and it improves bit by bit. And different software, Windows obviously still has major releases, operating systems still have major releases, but they also have minor releases and they have a lot more of those minor releases than they used to for any number of reasons, but a lot of them have to do with this adoption of Agile Scrum methodologies.

And then, of course, the last thing is we are sort of limiting our downside. So even if we fail, even if we get something 100% completely wrong, which is rare but possible, we haven't overinvested the team and the business in doing all this stuff that is wrong, right? That doesn't actually work.

Okay, and with that, I can now see that what I thought was going to be two episodes is going to be at least three. This is my own optimism bias getting in the way. I am going to try to wrap up this particular episode and kind of leave it on this initial dive into the Scrum methodology. But let me tie back some takeaways around things that I think you can use out of Scrum that will be effective in your legal practice.

So number one, taking from that last example first is resist the urge to overinvest in work product, any kind of work product, before you've validated that you're on the right track. And you can use work breakdown or iterative processes so that you can do this validation in an intentional and also inexpensive way. And this goes for your marketing pieces, this goes for client facing templates. This could go for your legal briefs or motions, right? This iterative development approach lets you build in smaller chunks, gather feedback sooner, and ultimately reduce wasted effort. And I'll give you one example, right?

And I sometimes refer to the minimum viable motion. And this minimum viable product idea comes from the Lean startup, which is a cousin of Agile, even though it uses the word Lean in it. I won't go deep down that rabbit hole right now. But the typical way that a firm might build, let's say, a summary judgment motion would be to sort of gather up all of the possible arguments and do the research on all the work and then write the motion from start to finish and then probably the associate writes the motion and then sends it to a more senior attorney for review.

But this iterative approach might say, hey, let's not write the whole thing. Let's pick of this universe of arguments that we could make, let's pick the one that we think is our winner. We think this is the best argument that we have. And let's write a limited scope motion that just addresses this argument. Now, this isn't necessarily going to go to the judge or go to the court, right? But it's an exercise for the associate to say, okay, let me actually break this down, make this legal argument, maybe write the fact pattern that supports the legal argument, maybe write the intro that leads into the fact pattern, write the conclusion. At the end of it, you've got, again, not something you would necessarily submit as your exact perfect motion that is what you hope to get to, but by throwing that minimum viable motion over into the quality assurance process, which is to say sending it to a senior attorney for review, you now have accelerated the feedback loop and that senior attorney can read through it and give feedback to the associate based on their knowledge and experience that says, oh, yeah, this actually is a really good argument, or, you know what, now that you've fleshed it out, this isn't as good an argument as we thought.

And that's important learning because particularly if the associate was definitely going down the wrong path, you've now sort of capped the amount of investment that they make. You've accelerated the feedback loop, you've capped your losses, and therefore you can go back and make adjustments without having to chuck this whole maybe 20 page motion as opposed to a five page motion.

If they are on the right track, now you've actually also got an interesting opportunity because if you've got plenty of time before the motion is due, then you might want to say, okay, great, we're on the right track. Let's add our second best argument. Let's add our third best argument. Let's really build this into something that's useful.

But let's say you don't have enough time. Let's say the deadline is approaching. It might be a reasonable response to say, you know what, this is pretty good. We're going to go ahead and submit this. And yeah, it's not like got the exhaustive work that we could have done around all of the possible arguments we might make, but because our capacity is finite, because our timeline is limited, we still can submit something that we feel really good about even if it's not as robust as we might like to.

And even as I say that out loud, I recognize a lot of lawyers, a lot of you probably are kind of cringing at this idea. It's like, no, I've got to create the best motion, right? We're all strivers, we're gunners, we're whatever. Like we want to get to this place. But we all have to make these trade offs in lots of parts of our lives, lots of parts of our practices. And even when you think you're putting in, right, the page count is a constraint. So you can only make so many arguments as you can fit into the page limit, assuming you're in a jurisdiction that has page limits, which I think most of us are at this point.

So whatever it happens to be, right, we are constraining the capacity, we're constraining, we're having that honest reckoning with capacity, which forces us into choices about prioritization. And this minimum viable iterative approach to legal research, to brief writing, to template development. Again, any part of your practice can benefit from this.

And that kind of segues into the second point I wanted to make, which is the point I always make, which is your capacity is finite. By recognizing that your capacity is finite, you now are forced to make smarter decisions about

your priorities. And so the emphasis in Scrum on fixed capacity and short delivery cycles is a great reinforcement of this honest reckoning with capacity idea. It forces us into, okay, if I only have a short amount of time, what is the thing that I should allow into my perception, my universe, my world, that I can apply that time and attention to in order to have the best possible outcome for the client, for the firm, for the team, etc.

The third thing I hope you'll take away is the usefulness of having these integrated teams and sort of integrated flows of work, being intentional about the flow of work through your law practice and recognizing that you probably too have a similar set of phases to your legal delivery work as a Scrum team does, right? There's a design stage. Maybe we call it strategy development or case strategy, whatever it is. But a lot of the firms I work with when I first come in, they're not intentional about that strategy phase or that design phase.

And so I'd really encourage you to be more intentional about that. Then you've got your development or your drafting phase. And then the next one is quality assurance. And this is another one where, yes, we do sort of an attorney review or we have these vague notions of quality control in legal, but I really think we have a lot to learn from the software industry and from other industries about creating consistent tests and standards against which and processes for conducting quality control.

And so being intentional about that, I think is important. And then that last one is, okay, great. How does this particular chunk of work, right, this discovery request, this particular part of an estate plan, how does it integrate back in with the whole, right? How does it meet the strategic goals of the overall matter and how do we show that this unit of work is helping the whole matter or the whole case progress towards the outcome that we or the client or both are trying to accomplish with this work.

Ep #76: How Scrum Lessons Can Improve Your Law Practice: Kanban for Lawyers Part 2

So, this is now part two of three. As I said, next week I will go deeper into how the Scrum method inside of the software development world morphed into the Kanban method or how the Kanban method sort of came in to be better than Scrum in certain situations, although not as good as Scrum in other situations, but also why I think that Kanban is really the better tool set for most law practices most of the time.

If you have any questions about any of this, as always, you can reach out to me at john.grant@agileattorney.com. You can get great resources from the Agile Attorney community. It's a free membership. You can sign up. I've got some stuff in there. I'll be putting more things in there over time. And if you have anything that you want to talk about or have me talk about on the podcast, please don't hesitate to reach out to me with those as well.

As always, this podcast gets production support from the great team at Digital Freedom Productions and our theme music is Hello by Lunara. Thanks for listening and I will catch you next week.