ProKanban.org

The KANBAN Pocket Guide

What No One Has Told You About Kanban Could Kill You

Colleen Johnson, Prateek Singh
Dan Vacanti

The Kanban Pocket Guide

What No One Has Told You About Kanban Could Kill You

Daniel Vacanti, Prateek Singh and Colleen Johnson

This book is for sale at http://leanpub.com/thekanbanpocketguide

This version was published on 2022-08-19

ISBN 978-0-9864363-8-3

Contents

Preface

What is Kanban?

To paraphrase Ryan Ripley and Todd Miller, "Have you read the Kanban Guide?"[1]

Published back in 2020, The Kanban Guide[2] is the definitive reference as to what Kanban really is:

"Kanban is a strategy for optimizing the flow of value through a process that uses a visual, pull-based system. There may be various ways to define value, including consideration of the needs of the customer, the end-user, the organization, and the environment, for example.

Kanban comprises the following three practices working in tandem:

- Defining and visualizing a workflow
- Actively managing items in a workflow
- Improving a workflow

In their implementation, these Kanban practices are collectively called a Kanban system."

The concepts included in this book are going to be based on that definition of Kanban as well as on the other elements of Kanban that are detailed in the guide. If you haven't read the Kanban Guide yet, then you should stop reading this book, go read that document first, and then come back here.

Why Does The World Need Another Kanban Book?

The truth is it doesn't. But the even bigger truth is that it really does. This book–as with the The Kanban Guide mentioned above–aims to strip Kanban down to its barest essence to make it as applicable across as many contexts as possible. As such, this book will focus on Kanban as a strategy to achieve flow and will purposefully ignore many unnecessary prescriptive elements that you may have seen erroneously bolted onto Kanban in the past.

This in no way means that you should not study other Lean-Agile practices that can be used alongside Kanban (e.g., Scrum, Theory of Constraints, etc.). To be sure, other practices may–and in most contexts should–be added to your repertoire when implementing Kanban. But it does mean that what is defined in the Kanban Guide, and what is explained in more detailed here, represents what the wider community accepts as Kanban and establishes the common language to be used when discussing the strategy of flow.

Who This Book Is For

This book is for anyone who has read The Kanban Guide and would like some assistance in filling in any gaps in their knowledge about what Kanban really is.

We wrote this book for the many people who have either tried Kanban or who are looking to try Kanban in either their professional or personal lives. To that end we have put special emphasis on some of the justifications for why Kanban exists as well on clearing up many of the myths that exist around Kanban today. We perpetually strive to clear up many of the misconceptions about Kanban, and this book represents just one step on that journey.

This book does not, however, represent a step by step guide to implement Kanban. Nor is this book the be-all-end-all reference for Kanban. It would be impossible to present all supporting theory, so all we've tried to do is to summarize some important concepts as well as give some tips on how to get started. For a more detailed list of references for complementary theory and practices, you can reference the bibliography of this book, or, even better, visit ProKanban.org's website (https://www.prokanban.org). Between this book and those references you should have fairly thorough understanding of what Kanban really is.

How To Read This Book

While this book has been designed to be read in order, there is nothing stopping you (as with any book) from jumping straight in at any chapter that looks interesting. We have tried as much as possible in each chapter to provide references to other parts in the book where important concepts are covered. In that sense, you could turn this pocket guide into a kind of "choose your own adventure" when reading. However, if you are new to Kanban then we strongly urge you to read the chapters in order so that you can progress your learning without any loss of continuity.

Final Thought

There is a lot of noise out there about Kanban. Search the interwebs and you will quickly get lost in the cacophony that is disinformation about flow. We hope this book will help you navigate some of that noise to get the most out of what we believe is the most exciting set of practices in the Lean-Agile world. So we'll stop droning on here and let you get to the interesting stuff in Chapter 1.

Have fun!

Endnotes

1. Todd Miller and Ryan Ripley, “Fixing Your Scrum” (The Pragmatic Programmers, 2020)
2. John Coleman and Daniel Vacanti, “The Kanban Guide” https://kanbanguides.org/

Prologue - The Secret History of Kanban and Why It Matters

The following text originally appeared as a blog post and accompanying presentation[1] by Darren Davis. It is reproduced here with Darren's full permission.

In the Beginning

In the summer of 2006 I was working as a Development Manager for Corbis, a media licensing company owned by Bill Gates. Corbis had an uncanny ability to lose money. When Bill went from being the richest man in the world to being the second richest, I liked to say I played a part in it simply by working for Corbis. At the time our sustainment engineering effort was in a shambles, and I was on the team that was trying to pull it out of the tar pit and make it functional. At that point it was being run as a series of small projects, releasing every quarter, with a fixed scope, a little like scooping a load of socks from the dryer, and crab-walking them up a flight of stairs while trying to drop as few as possible. Needless to say, the process was consistently disappointing. The team had heard some crazy talk about a thing called Agile, and decided to include stand-ups in their process. These quickly degenerated into 30-45 minute morning marathons that succeeded in sucking the optimism out of all of those unlucky enough to drop by. It was clear things had to change, and a few of us were having discussions about how to improve the process.

At some point during the summer one of the members of our team, Rick Garber, heard a talk from a Scotsman, David Anderson, covering how he had fixed some of the very same issues on one of his teams at Microsoft. His methods, based on the Theory of Constraints and other work by the likes of Goldratt and Deming, eliminated explicit estimation from the process, and relied on data to provide a probabilistic means of determining when software was likely to be done. Frankly, he had me at eliminating estimation, but the rest of the theories were also compelling. It blew our minds to think of software as inventory that could go stale, or that by idling some resources we could actually make the whole system more effective. Several of us read his book and engaged in a series of conversations like hardcore converts, eyes shining with revolutionary zeal, eager to remake the world. Or at the very least, our moribund sustainment process. We began building out or very own Kanban-based system.

It took us several months and extended well into the fall of 2006. In the middle of that process, David joined Corbis as the Senior Director of Software Engineering, and I began reporting to him. Also on the team at that time were Dominica Degrandis, Mark Grotte, Larry Cohen, Rick Garber, and Steven Weiss. David guided us through the rest of the process and finally, by November, we settled on a design that had his blessing. We trained our team on the process, populated our queues, and with great anticipation and fanfare, launched the first significant implementation of a Kanban-based system we were aware of.

It promptly went nowhere. For months.

Kanban Launches, and Flops

Keep in mind that our system was based on David's first book, Agile Management for Software Engineering, not on his subsequent work. There were no real practical examples of how to implement his

theories in the first book, and we designed our system much like his one previous (and smaller) implementation, which he assured us had been a tremendous success. And yet our process was quickly proving to be a non-starter. At this point in its history, Kanban didn't look anything like the process we know today. What we had was about 25 work items stored in Team Foundation Server, organized in a series of approximately 14 queues, with a tangled web of transitions between the various states, and not much else. According to the theory, once the system was set up it would be self-managing. People understood their roles, would monitor their queues, do their work and pass it on. Every two weeks any work that had passed through the process and was sitting in "Ready for Production" would be released. Very few items were making it through to production though, and none of us, including David, understood why. People complained that they had no idea where things were in the process, devs and QA engineers had no visibility upstream to the work that was headed their way, and the customers were increasingly angry that they were getting nothing but the thinnest trickle of work out of our grand experiment. We kept insisting that it should work, that we'd designed the machinery in such a way that people could focus on their area alone, without concern for other areas of the system. "Just focus on your queue," we would say, "and when work comes in simply pick it up, do the work and move it on." But still they complained, and our customers were growing impatient. I would like to say we were hurtling toward disaster, but that implies too much velocity. We were grinding to a halt.

At the beginning of February the CIO (and David's boss), Stephen Gillett called me into his office and said, "If you don't fix this thing I'm going to have to fire somebody." I didn't think he was saying I personally needed to fix it, but I did think he was threatening to fire me if the team couldn't get the process running (turns out he wasn't referring to me at all, but that's a story for another day). I got together with Mark, Dominica, Rick and Larry (David's

leadership team), and we talked about how to get things unstuck. It's important to note, I think, that we didn't set out to modify or "fix" the process. We simply, naively thought that it was only a problem of combustion, that if we could get the thing started it would take off. To that end, we decided to start running a stand-up each day. It was Dominica's suggestion, and we all thought it was a reasonable thing to do. Our intent was to do the stand-up for a month, until we started to see some momentum, at which point we'd go back to running it the way we had at the start. We decided to start the stand-ups on the following Monday. I insisted on running them because I knew if I ran the stand-up it would never be longer than 15 minutes. If I was ever dogmatic about anything, it was that stand-ups should be short. I still had psychic scars from the stand-ups our previous sustainment process had run. Interestingly, this single change, intended as a temporary nudge, drove all the other modifications that became Kanban as we know it today.

Micro Improvements and Happy Accidents

I had never run a stand-up before so I didn't really know what I was going to do that first Monday. Since people were complaining that they didn't have visibility into where things were, I thought it made sense to put the work up on a whiteboard so we could talk about it. I didn't have a clear idea of the format though, and decided to bounce it off some of our devs. That same week, as luck would have it, they were working with Daniel Vacanti doing Color Domain Modeling, a process of using color-coded post-it notes to puzzle out the design of software systems. When I told them I was thinking of putting the work up on the whiteboard one of the devs, Kurt Quamme, suggested using different colored post it notes to represent the work. It seemed like a good idea, so I went home that weekend and sketched out a fairly simple plan.

The most common color of post-it was yellow, so we'd use those to represent feature requests. To represent bugs we would use blue, because, well to be honest, because both "bugs" and "blue" start with the letter b. And that was pretty much it for the first board. I got to work early on Monday, drew up a very crude set of queues, wrote up some post-its and waited for people to arrive. We'd made it clear that if something was assigned to you, you were required to attend the stand-up, but that first morning most of the team attended. It may have been because people were curious to see what was going on, or because they wanted to feel like they were part of the team, but most likely it's because the board was set up in a fairly public area directly behind my desk and during the stand-up it was difficult for people nearby to do anything but join in. Whatever the reason, it was a big group that first morning, and would continue to be a large, inclusive gathering for most of the rest of my time there.

The first, most pressing problem we were trying to solve, in fact at that point the only problem we were trying to solve, was why work wasn't moving through the queues. In the stand-up that was our focus. It seems obvious now that focusing on blocking issues is the primary goal of a Kanban stand-up, but at the time it was a radical departure from the more orthodox, "Open Mic Night" style of stand-up common to Scrum, the kind of approach that goes around the circle and asks people what they did yesterday and what they plan on doing today. I wasn't trying to be radical, I simply didn't know any better. Over time, as a group, we would refine the process a bit, settling on a few standard questions:

- Is there anything blocking you that's not on the board?
- With the issues on the board, is there somebody actively working to clear the issue?
- Do you need anything from management to get the issue cleared?

We always kept the meeting under 15 minutes, often under 10, but found that people would "stay after class" and discuss specific issues in groups of 2 or 3. In a sense it was Meeting 101: don't waste

people's time, only discuss issues in front of the whole group that the whole group needs to hear. We also got the sense that the team was more focused, more energized than they had been when the old stand-ups dragged on so long.

I don't recall who suggested using bright pink stickies to identify blocking issues, or when exactly, but it happened very early on. I wish I knew so I could buy them a drink. It was another simple but brilliant innovation that helped us stumble onto one of the key benefits of contemporary Kanban: the inherent power of visualizing the work in progress. We didn't invent the concept, by any means, and if we had read up on the work of Edward Tufte we may have gotten to these ideas a lot sooner, but as the team iterated on the form and content of the board we became increasingly aware of its importance. By putting a pink sticky note on the feature or bug it was blocking, it quickly communicated to even the most casual observer that something was wrong. Color coding and other cues allowed us to get different levels of detail out of the same board. You could stand back and get a picture of the overall system health, where the bottlenecks were, what the batch size of the next release would be, or you could stand closer and see that the same developer was assigned to multiple items and needed to be refocused on a single task. The board also became the focal point of discussions and planning for a variety of people on the team. Our tester, Tom Utterback, and our build engineer, Doug Buros, would use the board to plan how builds would move into QA. We started collecting all of the items we released by sticking them to the walls on the far right side of the board. It started out as a kind of joke, but quickly became an advertisement and reminder of our success as a team. All these things are obvious now, and widely used, but at the time we really were just making them up, experimenting, discovering their value as we went along.

Welcome to Thunderdome, the Process of Prioritization

The process of selecting and prioritizing work was done in a weekly meeting of the VPs from various parts of the company. Marketing, Finance, Sales, Imaging, would all show up on Monday for an hour to select the next items pulled in for work. The meeting was run by Diana Kolomiyets, who was also instrumental in keeping things flowing through the queues and managing the releases. The meeting went through several iterations, like everything else, but eventually settled on a multi-vote system. Each representative had three votes. The meeting would start by going over new requests for the week, then asking for nominations. Nominating an item to be worked on was, essentially, a plea for support, and there was a fair amount of horse-trading as part of these sessions. Once a group of candidate items was determined, everybody would vote, with the top vote getters being added to our work queue (we called it Engineering Ready). The mechanism was pretty simple, and it was well established early on that if you didn't make the meeting, there was little chance your pet item would make it into the work queue. While we were building out the initial process, before any of the modifications that actually made the system work, we were checking in regularly with our customers. One of them, Drew McLean, was the VP over the imaging department. He was a former Marine, and had worked much of his career in manufacturing, most recently at Boeing. He understood the theories well, but insisted that we include a Silver Bullet, an ability to expedite a request. We resisted, thinking that everything would become a Silver Bullet, but it's hard to stand up to a former Marine who knows what he wants. We included a provision to expedite a request but added two rules:

• There can only be a single expedited request in the whole system at any one time. If there's already an expedited request a new one couldn't be added until it was released to production.

• The decision to mark an item as Expedited had to be a consensus of the various stakeholders from all the customer groups (Marketing, Sales, Finance, Imaging, etc.).

We were surprised to find, over time, that this mechanism was rarely invoked. When items did get marked for expediting they had a very noticeable negative effect on the rest of the system. Things would have to wait for the Silver Bullet, leading directly to longer lead times for the regular items. Because of this, it was not easy to get everybody to agree that a particular item was so much more important than the others that it deserved to be rushed.

Another interesting effect of better flow and throughput was with our SLAs. Initially we decided on a 21 day SLA, based purely on a guess. Since we had no data to begin with, we had to start somewhere and 21 days seemed like a defensible number. We agreed to revise this number over time as we got more data. We never hit that 21 day SLA, not even once. We bumped up the number to 28, but the best we ever did was right around 31 days. Oddly though, our customers didn't seem to mind. Because we were moving items through the system with pretty good regularity, and they had total visibility to where items were, they seemed mostly content to let the system work without holding our feet to the fire too much about the SLA. It may also be that the system was so dysfunctional for so long we'd succeeded in setting very low expectations.

So What's Your Point?

Why is any of this important? I think it's important for a variety of reasons, some so small as to almost be petty, but others that I think have much broader implications. The small reasons first: I think it's important that the people who actually did the work get some credit. I know how history works, and I know that more often than not attention is focused on an individual as being responsible for

a movement, even one that wouldn't have happened without the contributions of a much larger group. It's easier to label somebody the "Father of Kanban" than to continually point out all the aunts and uncles. This is a small attempt to correct that, or at least add some names to the record. The people mentioned above all played a role in the evolution of Kanban, but there are certainly others that I've forgotten as well. It's also important because it illustrates the gulf that often exists between theory and practice. I love theory, really I do, but only so far as it's effective in practice. However, Kanban as we know it in the industry today would not exist if a group of people hadn't been specifically tasked with making it work. The solutions we came up with worked for us, at that time, in that context. I don't think anybody was trying to create a methodology, we were just trying to make it work so we wouldn't get canned. With regard to the innovations that made our system work, nobody was guiding the team, approving changes to the process, or deciding which innovations to try next. Everything we did was driven by the team and evaluated solely on the basis of whether it was effective or not. Anybody who tells you differently is trying to sell you something.

The larger reason I think this is important is because I believe as an industry our focus is often in the wrong place. We seem to focus on learning a methodology like Scrum or Kanban, understanding the ceremonies and artifacts that define those approaches, and becoming as well-practiced at them as we can. We look outside our organizations for consultants or coaches who can come in and help us learn these techniques and apply them to our often unique and idiosyncratic challenges. Huge amounts of money are spent on justifications for various methodologies as a way of adding legitimacy to somebody's opinion of what we should or shouldn't be doing, like a growing sense of orthodoxy around what is "agile" and what is not. But it's my opinion that a Scrum Team, or a Kanban Team, is of far less value to any organization than a team of agile thinkers, people who have a good understanding of the

theories behind agile, but are empowered to question everything, to experiment, to fail, to learn and move on. I'll confess I'm not a fan of Scrum, but I like Kanban and have used it to great effect in a number of organizations now. As much as I like it, I know it won't last forever. Other ideas must and will come up that prove more effective in delivering value to our customers. It's evolution, and it's relentless. As great as an idea may be, it won't be the theorists that demonstrate its value. It will fall to the practitioners of the craft of software engineering, people in the trenches every day figuring out how to make those ideas work in the real world.

To do that requires an agile mind.

-Darren Davis

Endnotes

1. Darren Davis, "The Secret History of Kanban and Why it Matters" https://www.youtube.com/watch?v=7VT_Wqs4cRg&ab_-channel=ConfEngine

Chapter 1 - The Single Most Important Part of Kanban

"How would you teach someone to read a Cumulative Flow Diagram?" (Spoiler alert: the CFD is most definitely **NOT** the single most important part of Kanban.)

It was circa 2010 and I had just met Frank Vega for the first time at a Kanban meetup. In all honesty, I may have met Frank in passing before that, but 2010 was the first time I had a chance to really talk to him. At that time, Frank was one of the few people in the Kanban community who really knew what he was talking about (in my opinion, of course). On a Skype call shortly after that meetup Frank asked his fairly innocuous question, "How would you teach someone to read a Cumulative Flow Diagram?" (Incidentally, this was a completely rhetorical question as Frank knew very well–better than anyone else as it turns out–how to read a CFD). In the decade or more since I have been asked that question, most of my career has been based around trying to come up with a competent answer. My search has led me to following conclusion:

Most of what has become Kanban orthodoxy is simply wrong. Kanban doctrine as it exists today is based mostly on rumor and misunderstanding and is seldom based on science, much less fact. The early days of how the community talked about Cumulative Flow Diagrams (not only how to read them, but their importance altogether) was a perfect example of this triumph of ignorance.

Bear with me for a moment while I explain.

At the heart of what makes a Cumulative Flow Diagram (CFD)

work is a relationship known as Little's Law. Dr. Little even uses a CFD in one of the proofs of his eponymous law[1] (note: he called it a Cumulative Arrivals/Departures diagram as shown in Figure 1.1):

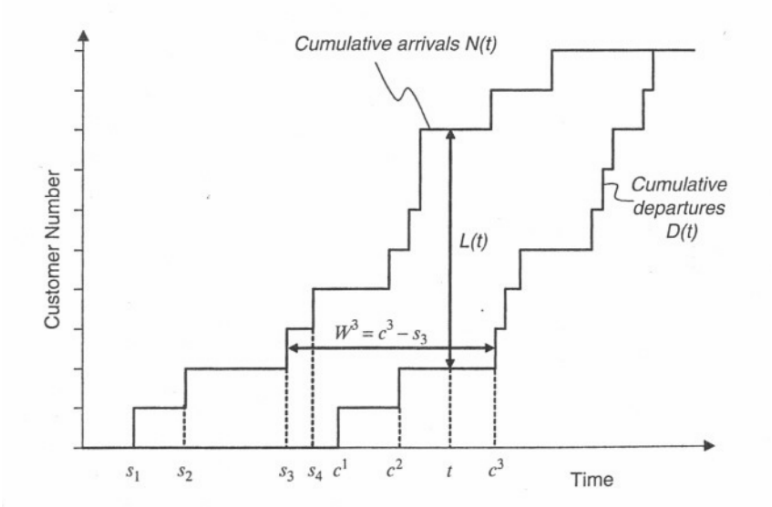

Figure 1.1: A Cumulative Arrivals/Departures Diagram

I bring up Little's Law because that equation is often held up as the foundational, irrefutable mathematical justification for Kanban. And it is. Its just not for the reasons that you think it is. Frank's question forced me into a deep dive of Little's Law (LL) and was the impetus for me to discover why all of us should really care about this simple equation (Spoiler alert: Little's Law itself is also **NOT** the single most important part of Kanban).

Let me say first that a full explanation of LL is beyond the scope of this book (for a fuller discussion please see "Actionable Agile Metrics for Predictability"[1]); however, I will need to take a few moments to summarize some of its relevant points.

LL can be stated as CT = WIP / TH where CT is the average Cycle Time of your process, WIP is the average Work In Progress of your process, and TH is the average Throughput of your process.

LL is exact in its calculation and this equation can be applied to any flow system. Before I explain that statement further, let's pause for a second and try an experiment. If you know the average TH, CT, and WIP of your process (for the last month, for example), I'd like you to plug those numbers into the LL equation now. Try several different permutations. Maybe first divide your WIP by your TH and see if you get your CT. Then try multiplying your CT by your TH to see if you get your WIP, etc. What do you see? My guess is your numbers aren't quite coming out the way you would expect them to or as predicted by LL. Not only are they probably off, but in some cases they are probably off by quite a bit.

What's going on here? The LL calculation is indeed exact, but it is only exact in contexts when a specific set of assumptions are fulfilled. Those assumptions are (for the time period under observation):

1. The average departure rate must equal average arrival rate
2. All items that enter the system must finish and exit the system
3. The amount of WIP is roughly the same at the beginning and end of the time interval under observation
4. The average age of WIP is neither growing nor declining
5. Consistent units are used for the measurement of TH, CT, and WIP.

[**Note**: Assumption #5 is given here for completeness purposes only as this last assumption is trivial. All it is saying is that if you want to measure CT in days, then TH needs to be measured per day and average WIP must be measured by day. Mixing units is a big no-no (e.g., CT in weeks and TH in story points), but that should be intuitively obvious. And if anyone on your team struggles with this point then you have bigger problems than how to best apply LL.]

Because your calculated numbers didn't come out as predicted by LL, that tells us that your process explicitly or implicitly violated one or more of the LL assumptions at least once and probably

at multiple points over the time period that you chose for your calculation. The net effect of violating LL's assumptions is that you have destabilized your process–as evidenced by the equation not working.

System stability (from a LL perspective) is so important because it is impossible to optimize a process that is inherently unstable. Your experience tells you this. How easy is it to optimize a process where the number of things you are working on increases every day? How easy is it to optimize a process where all the things you work on get blocked by dependencies on other teams? The LL assumptions, therefore, act as a powerful guide to policies we should implement to help prevent our process from destabilizing.

Any time you are applying Kanban to your context, you care about all 5 of Little's Law's assumptions (whether you know it or not).

And of those five, there is one assumption that rules them all—as promised by the title of this chapter.

The Most Important Part of Kanban: Work Item Age

A thorough understanding of what it means to violate each of LL's assumptions is key to the optimization of your delivery process. So let's take a minute to walk through each of those in a bit more detail.

The first thing to observe about the assumptions is that #1 and #3 are logically equivalent. I'm not sure why Dr. Little calls these out separately because I've never seen a case where one is fulfilled but the other is not. Therefore, I think we can safely treat those two as the same. But more importantly, you'll notice what Little is **not** saying here with either #1 or #3. He is making no judgment about the actual amount of WIP that is required to be in the system. He

says nothing of less WIP being better or more WIP being worse. In fact, Little couldn't care less. All he cares about is that WIP is stable over time. So while having arrivals match departures (and thus unchanging WIP over time) is important, that tells us *nothing* about whether we have too much WIP, too little WIP, or just the right amount of WIP. Assumptions #1 and #3 therefore, while important, can be ruled out as *the* most important.

Assumption #2 is one that is frequently ignored. In your work, how often do you start something but never complete it? My guess is the number of times that has happened to you over the past few months is something greater than zero. Even so, while this assumption is again of crucial importance, it is usually the exception rather than the rule. Unless you find yourself in a context where you are always abandoning more work than you complete (in which case you have much bigger problems than LL), this assumption will also not be the dominant reason why you have a suboptimal workflow.

Which leaves us with assumption #4. Allowing items to arbitrarily age is the single greatest factor as to why you are not efficient, effective, nor predictable at delivering customer value. Stated a different way, if you plan to adopt Kanban (or if you are already practicing Kanban), ***the single most important aspect that you should be paying attention to is not letting work items age unnecessarily!***.

More than limiting WIP, more than visualizing work, more than finding bottlenecks (which is not really a Kanban thing anyway), the only question to ask of your Kanban system is are you letting items age needlessly?

Why You Should Care About Aging

Before we get into aging, we need to take a step back and first talk about Cycle Time (CT). Most people think that the reason Kanban

emphasizes CT so much is so that we can pressure Agile teams into getting more things done faster. Nothing could be further from the truth. The reason that Kanban cares about CT is because CT represents the time to customer feedback.

We'll see in a later chapter that until a work item is actually in the hands of the customer, that item represents only *hypothetical* value. Value can only determined by the customers themselves and that determination can only be made after the item is delivered. Thus, CT is really a measure of "time to validated feedback".

However, CT itself can only be calculated at or after the moment when the item has actually finished. Before it has finished all we know is the item's age. That aging process starts immediately once work begins. Further, work items will continue to age until they are ultimately delivered to the customer. Thus, the more items age, the longer we delay precious feedback from the customer.

That delayed feedback increases the chances of something going wrong with delivery. Maybe the business environment changes, maybe customer requirements change, maybe a global pandemic takes over–it's impossible to know what might happen to change a customer's needs. But what we do know is that longer age represents higher risk. And the ultimate risk is that we spend a long time working on something that ends up not being valuable. As my friend and colleague Prateek Singh likes to say, "it is all about finding out how wrong you are as quickly as possible." By letting items age unnecessarily, you are not just sabotaging your ability to deliver, you are sabotaging your ability to deliver what your customers really want.

How To Prevent Work Items From Aging

So if aging is so bad, how do we prevent it from happening?

A question I love to ask in my workshops is "what are the two most effective ways to prevent items from aging unnecessarily?" This question usually stumps attendees because they want to run back to the dogma that they've been previously taught. You'll get answers like "lower WIP", or "clear blockers", or the like. But as we've just seen those answers don't necessarily lead to shorter age.

The first way to prevent items from aging is to finish them. It's that simple. If an item finishes, it is no longer aging. We can then begin the process to get customer feedback.

The second (and probably even better) way to prevent items from aging is to not start them. How many times are you and your team pressured to start work when you are not ready just for the sake of looking like you are making progress? From an LL perspective, that is the absolute worst thing that you can do.

Now let's put this all together. If you finish work as quickly as possible and don't start work until you are ready to do so, what have you just done? You guessed it, you've just controlled Work In Progress.

The real reason to control WIP is to prevent unnecessary aging.

We can take this logic a step further and assert that **all** Kanban practices can be derived from the basic principle that we don't want items to age unnecessarily. Why visualize work? So we can see where work is piling up and items are aging unnecessarily. Why mark work as blocked? So we can see where flow is not happening and items are aging unnecessarily. Why implement pull policies? So some items aren't allowed to jump the queue which would cause other items to age unnecessarily. And so on.

All Kanban practices can be derived from the singular motivation of not wanting items to age unnecessarily.

One final and very important thing to mention about how to prevent items from aging too much: If an item is taking too long to flow then the biggest culprit is probably that the item is too big. One

of the first things you should look at for a "stuck" item is creative ways to break it up into many smaller items. Keep in mind that the idea here is not to break items up just to make our numbers look good. Rather, we want to find ways to break a big, valuable piece of work into several smaller–but still valuable!–pieces of work. In flow terms what we are really talking about is batch size. So many times you may be working on a single item–a single story, a single epic, a single feature, whatever–but what you really are working on are several small items masquerading as one large item (strategies for breaking work down will be explored in more detail in Chapter 3). The best signal you have that something may be too big and therefore may need to be broken down is its age. Ignore age at your peril.

Conclusion

If you are not paying attention to aging, you are missing the only real reason to do Kanban.

In other words, if Kanban is all about the optimal delivery of customer value, then how do you really know how optimal you are? The answer does not lie in WIP limits, CFDs, Flow Efficiency, change management, or any of the other B.S. you may have been fed until now. The answer lies in your ability to know whether your items are aging unnecessarily or not. Everything else in Kanban should be subordinate to that one aim.

However, an item that is aging in and of itself is not necessarily a bad thing. The reality is that all items must age to some extent before they can be delivered. The question we must ask, therefore, is how much age is too much age?

The answer is the Service Level Expectation or SLE. SLEs are another one of those topics that you have probably not heard much about. The SLE is so fundamental, in fact, that it deserves its own

chapter, and that chapter immediately follows…

Endnotes

1. Daniel Vacanti, "Actionable Agile Metrics for Predictability" (ActionableAgile Press, 2014)
2. Little, J. D. C., and S. C. Graves. "Little's Law." D. Chhajed, T. J. Lowe, eds. Building Intuition: Insights from Basic Operations Management Models and Principles (Springer Science + Business Media LLC, New York, 2008)
3. Little, J. D. C., and S. C. Graves. "Little's Law." D. Chhajed, T. J. Lowe, eds. Building Intuition: Insights from Basic Operations Management Models and Principles (Springer Science + Business Media LLC, New York, 2008)

Chapter 2 - The Service Level Expectation

Maybe you've heard a myth about Kanban that goes something like this:

"Because Kanban has no timeboxes, items are allowed to take as much time as they need to finish."

Or maybe you've heard something like this:

"We can't do Scrum because we can't finish items in two weeks. So we do Kanban because Kanban has no requirement to get items done in two weeks."

Putting aside the misconceptions about Scrum in the second statement for a second, both of the above quotes are incorrect in their assessment of Kanban. Work Items in Kanban do not get to sit in progress forever and finish whenever we get around to working on them. That is the antithesis of flow. Flow implies movement or progress. And if items are just sitting and aging then there is no flow. No flow, no Kanban.

So, no, in Kanban items don't get to take as long as they want to finish. But what is Kanban's solution to this problem? Well, as always, it's helpful to look at things from the perspective of our customers. What's the first question our customers will ask us once we start to work on something for them? If you answered "When will it be done?" then you win a prize. Whether you agree or not, that is a reasonable question for our customers to ask. And we need a way to provide them an answer.

If you think about it, what our customers are really asking us to do is to predict the future. Therefore, any answer we give them is tantamount to a forecast. A funny thing about the future, however,

is that it has this nasty habit of being full of uncertainty. Despite what some people might tell you, no one can predict the future with 100% certainty. The second uncertainty is involved in any endeavour, a probabilistic approach is warranted.

For example, before I flip this coin, tell me with 100% certainty that it will come up exactly heads. Obviously you can't give 100% certainty before the flip, but what you can say is you have a 50% chance of it being heads (and a 50% chance of it being tails). As another example, before I roll this 6-sided die tell me with 100% certainly that I will roll exactly a 3. Again, 100% certainty doesn't exist, but I do know I have about a 17% chance of rolling a 3.

The same principle applies in our work. Once I start to work on an item, it is impossible for me to say with 100% certainty exactly how long it will take for that item to finish. But what I can do is look at historical data to come up with a probabilistic statement about how long it should take (e.g., "85% chance of finishing in 12 days or less"). By the way, another word for "probabilistic statement about the future" is "forecast".

Putting this all together, when our customers ask "when will it be done?" we need to answer them with a forecast. In Kanban, the probabilistic statement about how long it will take for individual items to finish once started is known as the Service Level Expectation or SLE.

From the Kanban Guide: "The SLE is a forecast of how long it should take a single work item to flow from started to finished. The SLE itself has two parts: a period of elapsed time and a probability associated with that period (e.g., "85% of work items will be finished in eight days or less"). The SLE should be based on historical cycle time, and once calculated, should be visualized on the Kanban board. If historical cycle time data does not exist, a best guess will do until there is enough historical data for a proper SLE calculation."

The SLE serves two functions in Kanban. First, it provides a completion forecast for work items once they have started. Second,

the SLE helps us to answer the question we posed at the end of the last chapter, namely, "how much age is too much age?".

How to calculate an SLE for your process is covered ad nauseam in AAMFP(Actionable Agile Metrics for Predictability) and WWIBD(When Will it be Done?). If you are not familiar with the derivation of an SLE we would ask you to familiarize yourself with that concept before proceeding, as our attention here will be focused on the practicalities of how to use the SLE once calculated–especially in relation to age.

Percentiles as Intervention Triggers

As items age, we gain information about them. The percentiles on our Scatterplot work as perfect checkpoints to examine our newfound information. We will use these checkpoints to be as proactive as possible to insure that work gets completed in a timely and predictable manner.

How does this work? Let's talk about the 50th percentile first. And let's assume for this discussion that our team is using an 85th percentile SLE. Once an item remains in progress to a point such that its age is the same as the Cycle Time of the 50th percentile line, we can say a couple of things. First, we can say that, by definition, this item is now larger than half the work items we have seen before. That might give us reason to pause. What have we found out about this item that might require us to take action on it? Do we need to swarm on it? Do we need to break it up? Do we need to escalate the removal of a blocker? The urgency of these questions is due to the second thing we can say when an item's age reaches the 50th percentile. When we first pulled the work item into our process it had a 15% chance of violating its SLE (that is the very definition of using the 85th percentile as an SLE). Now that the item has hit the 50th percentile, the chance of it violating its SLE has doubled from 15% to 30%. Remember, the older an item gets

the larger the probability that it will get older. Even if that does not cause concern, it should at least cause conversation. This is what actionable predictability is all about.

When an item has aged to the 70th percentile line, we know it is older than more than two-thirds of the other items we have seen before. And now its chance of missing its SLE has jumped to 50%. Flip a coin. The conversations we were having earlier (e.g., pair, swarm, break the item up) should now become all the more urgent.

And they should continue to be urgent as that work item's age gets closer and closer to the 85th percentile. The last thing we want is for that item to violate its SLE—even though in this example we know it is going to happen 15% of the time. We want to make sure that we have done everything we can to prevent a violation occurring. The reason for this is just because an item has breached its SLE does not mean that we all of a sudden take our foot off the gas. We still need to finish that work. Some customer somewhere is waiting for their value to be delivered.

Right Sizing

We lied to you earlier when we said the SLE serves two functions in Kanban. It actually serves three. The third function is to assist in a practice known as right sizing. There is a pervasive Kanban myth that all items that flow through your process have to be of the same size. After all, that's the only way that WIP Limits make sense, right? Wrong. There is nothing in flow theory that demands that all items that flow through a WIP-controlled system be exactly the same size. In fact, there is a whole theory of variation that acknowledges that not only do items not have to be of exactly the same size but that there is also nothing you could ever do to make them all exactly the same size–even if you wanted to. That is, variation in work item size *will always exist.*

Therefore, the consequence of variation is that we have to design a system that is able to gracefully handle the varying size of items that will ultimately enter our system. But there are limits to the amount of variation that we can handle. To illustrate this idea, Frank Vega loves to use the example of a wood chipper (anyone who has seen the movie *Fargo* knows exactly what we are talking about). Think about what happens when you try to shove a tree branch that is too big into the wood chipper (a la *Fargo*). At the very least that branch will get stuck. At worst, that branch will break your chipper. Likewise, what happens if you were to pick a bunch of sawdust and throw all that sawdust into the chipper? That would clog things up too. But those are extreme cases. The wood chipper would be able to reasonably handle anything sized between sawdust and a small tree trunk. Any branch that the wood chipper can handle without struggling is said to be right sized.

The same is true of your process. For your process the right size will be the range of possible outcomes as dictated by the percentile confidence that you have chosen for your SLE. For example, if you are using an 85th percentile as your SLE, and in your process the 85th percentile is 12 days or less, then the right size for items to flow into your system is 12 days or less.

Conclusion

Once you get the hang of managing work by age, then you are about 80% of the way to being able to optimize flow. There are a few odds and ends that we still need to cover (i.e., the rest of this book), but none of that will mean anything unless you grasp the concept of work item age!

Remember, however, that it is impossible for us to know how big an item is *before* we start to work on it. As work progresses, we need to continually compare that item's age to our SLE, using percentile lines as triggers as previously discussed. Just because you thought

something was right sized when you started it doesn't meant that it actually is. The only way to truly know is to monitor aging for each and every item that flows through your system.

But monitoring is only half the battle–and it's not even the most important part. The other half of the battle is to take action once you have the information that an item is taking too long to complete. The best information in the world is useless unless action is taken. Hence, Kanban practice #2, "Active Management of Items in Progress". Luckily, we'll talk about that topic next.

Chapter 3 - Actively Managing Items in a Workflow

I once worked with a team that had designed a Kanban board which had a global Work In Progress limit of 9. They had also added an expedite lane on their board (a big no-no as anyone who has followed my work knows) but at least had set a WIP Limit of 1 for that expedite lane. I walked up to the board one morning before our daily standup and saw that there were 32 items in the expedite lane. This is a classic example of not actively managing items in a workflow.

What's the point of defining a workflow, setting WIP limits, agreeing pull policies, etc. if you are just going to ignore them? The second practice of Kanban, "Actively Manage Items In a Workflow", is where the flow rubber meets the praxis road. Kanban is about flow and flow is about movement and movement doesn't happen on its own. As Deming says, a system needs management [1], and systems based on flow are no different.

Active management of items in a workflow can take several forms, including but not limited to the following:

- Controlling WIP
- Avoiding work items piling up in any part of the workflow
- Ensuring work items do not age unnecessarily, using the SLE as a reference
- Unblocking blocked work

As discussed in Chapter 1, your primary tool for active management of WIP is to monitor age. That is, the only way we can know

with any reasonable amount of certainty whether items are not flowing the way we expect them to is look at age. But as we also saw in Chapter 2, the best information in the world is no good unless you do something with it.

What follows is a deeper dive into some actions you can take when you see that items are taking too long to complete. This is by no means an exhaustive list, but these are very good places to get started.

Pairing, Swarming and Mobbing

Work items often reveal previously unknown complexity as they move through the workflow. This complexity can cause them to age more than other items. An item that has aged to an extent that it stands out in context of the team's flow, deserves some special attention. This might come in the form of having multiple team members jump in to help on this item (beware Brooks's law). This will often mean lowering WIP to help the aging item make progress. Folks finishing the next item should be asked to help out with the aging work item instead of picking up new ones. The act of lowering WIP to below the number of team members is known by multiple names - Pairing, Swarming and Mobbing to name a few. For ease of reference, we will refer to all these as Ensemble work.

There are three major ways in which ensemble work can help control age.

- Completing downstream tasks earlier - As an item ages in an earlier stage, we can enable faster flow through later stages. We can perform steps in the later stages earlier so that the task does not continue to age unnecessarily once it is past the current stage.
- Dividing work item tasks amongst team members - If the work item itself cannot be broken down into deliverable

chunks, it is possible to identify sub-tasks of the item. Different team members can take on the varying subtasks in parallel to help the item move forward.
- Removing Sticking Points - Often getting fresh perspectives on a problem a single team member has been facing helps in coming up with creative solutions. Whether these are just a result of "rubber ducking" or cross-functional pairing to get new perspectives, they help an aging item make progress.

Unblocking Blocked Work

By definition, any work that is blocked or on-hold is not flowing. These work items age, usually due to internal or external dependencies. If the cause is an internal dependency, we need to examine our process policies and look for improvement. If the cause is external, we need to figure out how to reduce the likelihood of this external dependency for future items or reduce the impact of this dependency on age. In other words, how do we get closer to eliminating the dependency or making the resolution time insignificant. Bringing external expertise in house, improving partner/vendor relationships or completely removing the dependency are all options we can exercise here. Whether the dependency is internal or external, we need to establish some policies around how we treat blocked work. There are at least three levels of blocked that need to be established -

- When to mark an item as blocked - How much time needs to pass before we mark an item whose progress is stopped as blocked? Is this in the order of hours, days or weeks?
- Blocked Items and WIP Limits - How long should a blocked item count towards our WIP limits and stop us from picking up other work? Does including it in WIP increase our focus on resolving it?

- Removing Blocked Items from the system - At what point do we say that the item is going to be blocked for so long that it might not be relevant to track it? Should we cancel the item or move it back to the backlog?

Right Sizing Redux

Read Don Reinertsen's "Principles of Product Development Flow" book and you will quickly realize that one of the biggest detriments to flow is working on items that are too big. In flow terms, that means controlling batch size. We saw earlier that usually when an item is stuck in your process it is because it is too big–it hasn't been right sized.

Right sizing is the art of enabling work to flow in small batches of value at every level. This means breaking things down into small, manageable chunks.

For sizing, all you need to do before you pull an item into your process is have a quick conversation about whether you can—based on what you know right now—get the item done in 12 days or less. If the answer is yes, then the conversation is over and you pull the item in and start working on it. If the answer is no, then you talk about how you can redefine the work item such that it is of the right size. Maybe you need to break it up. Maybe you need to tweak the acceptance criteria (more on breaking items up in the next section). Whatever the case, take the action you need beforehand and only pull the item in once you are 85% confident you can finish it in 12 days or less.

Prateek Singh communicated this guidance on right sizing when working with one of his teams:

"We have a general idea of how large work items at every level have been in the past. We can use this to 'size' upcoming items. Currently we have a very good idea of sizing at the story level due to the data

available. We are also going to lay out guidelines at the Epic level based on historical understanding of flow.

"In the Cycle Time scatterplot for our team, we noticed that 85% of the stories that we work on get done in 11 days or less. This is a guide for right sizing. Whenever the team picks up the next story, they should be able to ask themselves the question, 'is this the smallest bit of value and can it get done in 11 days or less?' If the answer to those questions is yes, great, no more estimation needed, start work on it. If the answer is no, let us try to break this story down. This is the essence of right sizing. Each team will figure out their right-size stories from their own data.

"For Epics - since we do not have great data, but a decent general idea in this regard, we are issuing some guidance. Epics should be 10 stories or less, 90% of the time. 10 is a soft number, it is something to aim for. The reality is that there will be Epics that will become 11/12 stories big. 90% of our Epics should be 10 stories or less.

This does not mean that we try to make all Epics close to 10 stories. The 'or less' part is important. If an Epic can be delivered to a customer in 3 stories, great, let us leave it that way. 10 is a soft upper bound, not a target."

This is all well and good, you may wonder, but how do we go about breaking items down once we recognized they haven't been right sized? I'm glad you asked.

Breaking Down Work Items

When you discover that an item is too large for your process, your first hypothesis should be that this large work item is probably composed of smaller, individually deliverable bits of value. Packaging multiple items into a single work item can negate many of the benefits that limiting Work in Progress provides. For example, if we are operating with a WIP limit of 1, but that one item could

potentially have been 5 separate items, our WIP is actually five times larger than what is visible. WIP often hides in work items. In order to expose our actual WIP, we should be breaking down work into individual 'feedback-able' pieces early and often.

Let's now walk through some simple strategies that can be used to break down work. It is a list that I and others have successfully used in the past to break work down. These strategies are effective at every level of work - stories, features or initiatives. Multiple of these strategies can be applied to the same work item as well. The end goal is to create smaller units of work that can help us get faster feedback from our customers.

Acceptance Criteria

The practice of adding user Acceptance Criteria (AC) helps us understand how a customer would expect to benefit from the work item. If the team works on initiatives that break down into features which are in turn composed of stories, each of those levels should have an acceptance criteria. At every level, we should be able to break work down by getting closer and closer to a 1:1 ratio of acceptance criteria and the work item. This does not mean that every story, feature or initiative should have only one acceptance criteria. Instead, we should look for each work item to have the minimum number of ACs that help us get feedback.

For example, consider the following work item.

As a reviewer I want to see the relevant sections of a submitted paper broken out so that I can easily assess them

- AC 1 - Reviewers should be able to see the title of the paper separate from the body
- AC 2 - Reviewers should be able to see the word count of the description
- AC 3 - Reviewers can grade each section separately

- AC 4 - Reviewers are able to view the main hypothesis in a separate section
- AC 5 - Reviewers can optionally separate out experimentation details to be assessed separately

In this case each of these ACs can be a separate work item. They can all independently be delivered to customers (internal or external) for us to get feedback. Each of these ACs starts solving a customer problem and delivers value without being held up for the others to finish.

Conjunctions and Connectors

Very often WIP hides in the titles of work items. Any time we see conjunctions, particularly ands and ors, in a work item title, it is an indication that the item can be broken down. Sometimes these are replaced by commas, dashes and slashes as well. These conjunctions and connectors are usually trying to lump multiple actions or actors into a single work item. Separating these can help us deliver on individual actions earlier.
For example the work item shown below can be broken into multiple items

- Patrons should be able to give restaurants overall, food quality, service and ambiance ratings

This work item can easily become four different work items as listed below. The delivery of the overall rating might be a part of the minimal viable product, while others can be later enhancements.

- Patrons should be able to give restaurants an overall rating
- Patrons should be able to give restaurants a food quality rating
- Patrons should be able to give restaurants a service rating
- Patrons should be able to give restaurants an ambiance rating

Generic Terms or Plurals

Generic terms are often used in work item titles to represent what should otherwise be multiple requirements. Getting a greater degree of specificity can help tease out the smaller work items that are hiding within the large one. Greater specificity also leads to better test plans, with fewer missing test cases. In fact, talking about how we would test an item is often a great way to move from generic to specific work items.

- The system should flag items with appropriate colours based on severity.

In the above item the generic terms ‘colours’ and ‘severity’ can be made more specific to create smaller work items which can then be prioritized. These items can be as follows -

- The system should flag events with greater than 90% impact as red.
- The system should flag events with between 50% and 90% impact as yellow.
- The system should flag events with less than 50% impact as green.

Optimize Now vs Later

This strategy focuses on a minimal viable increment approach at every level. What is the simplest, least optimized solution that we can deliver to start getting feedback from internal or external customers? The customers might want a lot more than the first deliverable, but we would get increasingly more aligned and deliver value with each increment. The example below shows how this can be done.

- Provide customers the ability to order pizzas listed in our menu online.

The above work item can be broken down into a few smaller items.

- Provide customers a button to order margherita pizza online.
- Provide customers the ability to change the size of pizza ordered.
- Provide customers the ability to pay for the pizza online.
- Provide customers the ability to select from a list of pizzas online.
- Allow customers to add/remove toppings from pizza orders.
- Allow customers to build their own pizza by adding toppings to a basic cheese pizza.

There are multiple such strategies that can be used to break work down. These techniques can be used in conjunction with each other to enable faster and more frequent delivery of value. They should be used at every level of work definition.
The following diagram is a cheat sheet that Becky McKneeley put together while working at Ultimate Software. It represents an excellent getting started guide for breaking down work items.

Strategy	Use When	Questions to ask
Acceptance Criteria	Individual Acceptance Criteria fits the INVEST acronym principles and can be split out	Do any of the ACs fit the INVEST acronym principles individually? Are all of the ACs required in order to receive some feedback (from anybody)?
Conjunctions and Connectors	Look for connector words (and, or, if, etc.), dashes, slashes, commas	Are there any conjunctions or connectors in the story title? Are there any conjunctions or connectors in any of the ACs? Can these conjunctions/connectors be split apart in order to receive feedback sooner?
Generic Terms & Plurals	Look for non-proper nouns or other generic words that could be replaced with something more specific. Look for plurals that could be split (pages, fields, etc.)	Can any of the terms listed be further or more clearly identified? Are plurals being used anywhere where they could be split up and processed separately?
User Role or Persona	More than one role is impacted, and the functionality is handled differently for each role	Which roles are involved in this story? Are different roles impacted differently by this functionality?
Business Rules	Can be hard to discover – but think about test cases. Test cases often imply or hint at business rules that could be broken out into individual stories	Which business rules apply to this story? Can simpler rules suffice in order to receive some feedback on the functionality? Which test scenarios are used to verify this story?
Platform Options	Provide for support of different platforms. For example, mobile vs tablet, IOS vs Android. Also used for browser compatibility	Which platforms/browsers need to be supported? Are all of these platforms/browsers required for a first pass?
Exception Processing	Happy vs Unhappy Path.	What is the happy path? Are there exceptions and/or edge cases identified in the story?
Operations	Based on different operations performed. Typically for the management of entities such as customers, etc. (CRUD)	What operations are involved in this story? Is it necessary that all operations are performed at once in order to receive some feedback?
Workflow Steps	Identify the steps in the workflow and implement it in workflow stages	What are the workflow steps in this story? Can the workflow steps be split and still provide some feedback on the functionality?
Optimize Now vs Later	**Functional Optimization**. Also referred to Simple to Complex. Implement simple functionality that provides enough value. More complex functionality added later	Can the functionality be simplified and still provide some value/feedback? What are other ways to handle this functionality that would allow us to get feedback sooner?
Major Effort	A significant effort is required for the first story. New technical aspects, etc.	Is there some new technology we are implementing? Is there one part of the story that we don't know how to handle technically or is new to the **TEAM**?

Figure 3.1: Sample Work Item Breakdown Strategies

Conclusion

The world is going to conspire against you to make your process as unpredictable as possible. Diligence is required to spot when items are taking too long, but more importantly diligence is required for prompt action. The world is indeed a nasty place out there and you should thank your favourite deity that you have Kanban on your

side.

Wouldn't it be nice if aging alone allowed us to see all of our process ills? It does not. Unfortunately, aging itself is dependent on how we have defined our workflow, so it is the nuance of a defined workflow that we will discuss next.

Endnotes

1. W. Edwards Deming, "Out of the Crisis" (The MIT Press, 2000)

Chapter 4 - Defining and Visualizing a Workflow

Pre-covid19 pandemic definition of Kanban: "We have Post-Its up on a whiteboard".

Post-covid19 pandemic definition of Kanban: "We have a Jira board set up".

I'd love to know where the misconception that Kanban is only about visualizing work came from. Saying that Kanban is only about visualization is like saying Scotland is only about whisky or Nadal is only a clay-court player. If you think that the first practice of Kanban "Define and Visualize a Workflow" means "have a Jira board" then you are sorely mistaken. Hopefully by the end of this chapter we will have convinced you otherwise.

The fundamental purpose of a Kanban board is not just to visualize work, it's to visualize a much broader concept known as a Definition of Workflow (DoW). A DoW is meant to encompass all aspects of how potential value flows through your system to be converted into tangible value once delivered to your customers. Specifically, at a minimum, a DoW must include:

- A definition of the individual units of value that are moving through the workflow. These units of value are referred to as work items (or items)
- A definition for when work items are started and finished within the workflow. Your workflow may have more than one started or finished points depending on the work item
- One or more defined states that the work items flow through from started to finished. Any work items between a started

point and a finished point are considered Work in Progress (WIP)
- A definition of how WIP will be controlled from started to finished
- Explicit policies about how work items can flow through each state from started to finished
- A service level expectation (SLE), which is a forecast of how long it should take a work item to flow from started to finished

A quick note about value: in this book whenever we say "value" I most likely mean "potential value". In most but not all cases the ideal state of your Kanban implementation is that you deliver items to your customers, validate that what you have delivered is valuable, and then make any improvements to your process based on what has or has not been validated. It would be a bit unwieldy for me to replace "value" with "potential value" in every instance, so please understand that unless we are talking about a context where value has been explicitly validated, what we really mean is potential value.

Work Items

Since the whole reason of Kanban is to optimize value delivery, it is reasonable to begin our discussion with how that value is encapsulated in our system. Most Lean-Agile implementations already have some notion of how value is captured to be worked on. Tools such as user stories, epics, features, etc. are all meant to make explicit what the notion of an individual unit of value is in our context. Kanban does not care what tool you use to define value in your system, all Kanban cares about is that you have some way of describing value in discrete units. These individual units of value that flow through your system are called work items (or items for short).

There is no other requirement about work items in Kanban other than they exist somewhere in your system. Kanban doesn't care if you use stories, or epics, or features, or fubars. Kanban doesn't care if they have been defined by a product owner, or ordered in a backlog somewhere, or refined by a team (although you may choose to do any of these things and more if you wish). From a Kanban perspective, all you need is a way to encapsulate what you think is value into an item that can flow through your process, be delivered to your customer, and be validated by them.

Workflow

Speaking of flow, once items have been defined, they need a set of value-add work steps that they can move through to be turned into something tangible.

Started and Finished

The first step in identifying your workflow is to draw boundaries around it. This means having at least one clearly defined point at which you consider work to have started and having at least one clearly defined point at which you consider work to have finished. So let's say you have a workflow that is Options -> Discovery -> Building -> Validating -> Done. We might define our started point to be when an item moves into the Discovery phase and we might define finished to be when an item moves into the Done stage. Or we might define started as Building and finished as Validating. The right started and finished points are totally dependent on your context and Kanban doesn't care how you set them as long as you do.

It gets a little more complicated than that because it is perfectly allowed in Kanban to have more than one started point and more

than one finished point. In the above example maybe we want to measure started from both Discovery and Building and finished from both Validating and Done. Why we might want to do that will depend on what questions you want to answer based on the metrics you collect. For example, maybe we want to know both how long it takes items to get done from when we first start working on them (Discovery to Done) as well as how long our Building step takes (Building to Validating).

It is encouraged that you experiment with different started and finished points in your process to better understand your context. The only thing to remember is that Kanban requires you have one of each defined. The rest of the DoW elements as well as the basic metrics of flow that you will track will be defined in terms of your decision around started/finished (a detailed discussion of flow metrics will be covered in a later chapter).

Workflow States

Once you have your started/finished decided, the next step is to map out the discrete value-added states between those two points. The number of states you choose will be context specific, but you must have at least one between started and finished. Contrary to popular belief, To Do -> Doing -> Done, where started is the Doing state and Done is the finished state, is a perfectly valid workflow in Kanban.

Exactly how to select what states should be in your workflow is way beyond the scope of this book. However, there a few things that you may want to keep in mind.

First, there are some very basic guidelines around how to identify different workflow states (but keep in mind there are no hard and fast rules):

1. Any value-added activity, like “Discovery”.

2. When you want to model an explicit hand-off between two people or groups, like separating a "Design" state into "Design" and "Design Review" (assuming the people doing the review are different than the people who did the design).
3. When you want to track metrics for an activity, like in the example above, maybe the review is not done by a separate group but you want to know how long it takes for a review to happen.
4. You generally want to bring more transparency to a particular activity.

It is a typical practice (though not a requirement) that each state you choose for your workflow will become a column on your Kanban board. More on that below.

Second, I'm a big believer in the KISS principle (Keep It Simple, Stupid). Especially when engineers are involved, it becomes very easy to design an overly complicated workflow. There is no rule of thumb around the exact number of states you should have in your workflow, but I'd like you to be skeptical whenever anyone wants to add a state or column. Not only do more states make it harder to understand what is actually going on in a workflow, but worse, more states are usually an excuse to increase Work In Progress (that's a bad thing, by the way).

Third, how you name your columns can either hurt or help collaboration. Generally it is poor form to name states or columns after specific roles in your organization–"Analysis", for example. One reason being that if a column is named after a role, then there may be a tendency for people to silo themselves into a particular part of the workflow and be reluctant to help out in other areas. When it comes to naming columns specifically, what you will quickly realize is that the name itself matters much less than the organization's shared understanding of what activity is being undertaken in that step. One of the best ways to facilitate

that understanding is to make policies explicit for each workflow stage. Which brings us to our next DoW element

Explicit Policies

Policies are the rules by which you play your process game. Think about how drastically different the game of baseball would be if each batter were allowed 1 strike instead of 3 or if games finished in 3 innings instead of nine. For you cricket fans out there, think about how different a test match is from a one day international is different from a twenty20 match. What leads to these massive differences and a change in outcomes are our policies or process rules.

Some examples of policies that exist within your process (whether you realize them or not):

1. Rules around how to handle blocked items.
2. The order in which you pull items through your board.
3. What it means for an item to be completed in a particular column.
4. When or if to break an item up into smaller items.
5. Whether items can flow backward or not (by the way, there is no rule in Kanban that items cannot flow backward–but backward flow may not be ideal in many contexts).

Making these policies explicit can go a long way to clearing up confusion around how work should flow within a given system. In fact, if you get these policies right then things you thought may have been important–like column names, or number of columns, or whatever–may become irrelevant.

There is one policy that we have skipped over but that may be the most important of all. For those of you keeping score at home, you'll know that of course we are referring to the policy around how WIP is controlled.

Controlling WIP

What do you call a highway that is operating at 100% capacity? What do you call a network that is operating at 100% capacity? What do you call a Starbucks queue that is operating at over 100% capacity?

We experience problems with flow every single day of our lives and those problems are usually directly attributable to not effectively controlling WIP. Knowledge work is no different. Generally speaking a person, or team, or organization is much more efficient at working on one or two things at a time than working on 100 or 200 things at a time. In other words, if you want to enable flow, at some point you will have to consider controlling WIP.

As with each of these sections, full details around how to control WIP is way beyond the scope of this book, but we do have space for a few key points.

First, Kanban does not care how you control WIP. It only cares that you do. Don't let anyone tell you that Kanban requires every column on your board to have a WIP limit. That is simply not true. You can control WIP however you want. You can have one big limit for the whole board, you can have a limit per person, you can group limits across several columns, and/or you can have a limit on every column. And that is just to name a few examples. Your DoW requires that you have an explicit policy around how WIP is to be controlled, but the implementation of that control is completely up to you.

Second, controlling WIP is just the first step. You also need to decide how you are going to respond when you are over your WIP control or under your WIP control. You are going to need to decide things like "should blocked items count against our WIP limit?".

Third, when starting out, keep in mind that the WIP control you choose is less about a hard and fast rule and more about a trigger for a conversation. Don't get too hung up on whether you are over

or under a limit but do pay attention to are those limits causing us to ask the right questions sooner? If your conversations are happening too late or too soon, those are good indications that a WIP limit may need to change.

The Service Level Expectation

You should have read all about SLEs in Chapter 2, so there is nothing more to say here other than SLE's are a first class member of your DoW. If you aren't using SLE's, then you aren't doing Kanban.

The Kanban Board

The collective visual implementation of each element of your DoW is called a Kanban Board. Same song, different verse: how you choose to visualize your DoW is completely up to you. You can use a whiteboard, you can use a virtual tool, you can use a spreadsheet– Kanban doesn't care. You are also only limited by your imagination in terms of how you choose to visualize your DoW. You can show flow from left to right, right to left, top to bottom, bottom to top. There's nothing that even says you have to use columns for states. You could use circles or spirals or whatever.

A very interesting topic for me of late is how do we make Kanban boards more inclusive and accessible for those of us who are visually challenged? We think more "traditional" visualization techniques may need to be challenged or scrapped altogether. But we also think that any solution that brings more than one of our senses to bear on solving a problem would be in keeping with the spirit of Kanban. That sense need not be limited to vision alone.

Conclusion

Once you've designed your workflow and created your board, it is time to use it. Yes, you heard me correctly, you actually need to use the board that you have just created. Review the last chapter for more information on how to do that.

But remember that pesky requirement from the definition of Kanban that we actually have to "optimize" the flow of value through our process? Optimization does not come from standing still. It comes from taking the original system that we have designed and performing experiments to find out how to make it work better. In short, to optimize our process we are going to need to continually improve it…

Chapter 5 - Improving a Workflow to Optimize for Flow

In the Kanban Guide, we purposefully gave preference to the word "optimizing" over the word "maximizing". This is because to "maximize" value is a very dangerous sentiment. Its fairly straightforward to maximize value over the short term. Just burn your people out, make decisions where you favour long term risk for short term gain (instead of short term risk for long term gain), ignore costs, etc. The trick is how do you position yourself to deliver value to your customers year in and year out. For example, Toyota has been doing this lean thing for over 80 years and my guess is they would tell you they are not done yet.

Optimizing also acknowledges that you are operating within certain constraints. Trying to maximize one means that you may minimize the others. Optimization implies finding the right balance across all organizational constraints. We believe the Kanban Guide says it best:

"Rather, value optimization means striving to find the right balance of effectiveness, efficiency, and predictability in how work gets done:

- An effective workflow is one that delivers what customers want when they want it.
- An efficient workflow allocates available economic resources as optimally as possible to deliver value.
- A predictable workflow means being able to accurately forecast value delivery within an acceptable degree of uncer-

tainty.

The strategy of Kanban is to get members to ask the right questions sooner as part of a continuous improvement effort in pursuit of these goals. Only by finding a sustainable balance among these three elements can value optimization be achieved."

You will note from the above statement that at the heart of optimization is this notion of continual improvement–which is the purpose of our discussion here.

Options for Improvement

The third practice of Kanban is to improve a workflow. Any element of your Definition of Workflow (DoW) is a candidate for experimental improvement. We say "experiment" because not all process changes will work. If the experiment makes things better, great. If it makes things worse, go back to what you were doing before. The point is to try.

Work Items

Updates to Work Items could range anywhere from what types to track, to what information is tracked on the work item, to how the work item is sized. For example let's say User Stories are one type of work item that you track on your Kanban board. Maybe an improvement is (as mentioned in Chapter 3) to put in place a policy that a given story can only have 1 or 2 acceptance criteria. Maybe you want to introduce color to segment the different types of work items. Maybe you want to eliminate the assignee field from your work items (assignee is probably not good information to be tracking anyway). Maybe you want to stop estimating work items in points (easily the best thing you can do to a work item). Whatever

the case, the point is you have plenty of options when it comes to experiments with work items in your workflow. Which brings us too…

Workflow

Equally you should experiment with different aspects of the workflow itself. Maybe it is adding or removing columns, maybe it is renaming columns, maybe it is removing swimlanes (better) or adding swimlanes (worse). Maybe you want to start to visualize steps upstream and downstream of your started and endpoints to get a more end-to-end view of your process (more on this in just a bit). The thing is, if you are using a virtual Kanban tool, adjusting your workflow is one place where you may be handcuffed. I've made no secret of my love affair with physical boards, but we understand that in our increasingly remote/distributed world physical boards may not be practical (not impossible, but not practical either). The advantage of a physical board is that you are only limited by your imagination in terms of how you want to visualize your workflow. With an electronic tool you are now locked into whatever functionality the tool supports. This is not fatal, however. I once worked with a team that used an electronic tool that did not natively support WIP controls. That didn't stop them as they put Post-It Notes with numbers on them to signify the WIP limits at the top of the monitor that broadcast the board in the team room. Which brings us to…

WIP Controls

In 2016, Steve Reid, Prateek Singh, and Daniel Vacanti published a case study from Ultimate Software detailing our success with Kanban up to that time[1]. In that case study we explored in some detail some of our efforts with a group known as the "Aces Team":

"After taking note of the Scatterplot, the team began to dive into the reasons why stories were taking so long to complete. What they discovered was that most long-lived stories were sitting in the "Ready for QA" column for extended periods of time. That was a problem because 'Ready for QA' is a queuing column where stories just sit and are not actively worked on. These "waiting" columns are the low hanging fruit of process improvement and so it was "Ready for QA" that the team decided to attack first by putting a WIP limit of 5 on that column. This decision meant that developers could not pull in new work if there were 5 or more things waiting for QA. They would instead have to go help with the testing of the product. This implication was discussed and accepted by the team as the appropriate behavior to ensure the flow of work."

What is not mentioned in this case study is the Aces continually reviewed and adjusted the WIP Limit on their "Ready for QA" column and ultimately (pun intended) got it down to two. You can read from the case study that they had evidence that this change worked because what they saw was a commensurate decrease in Cycle Time and increase in Throughput through their process.

You'll notice that hidden in this case study is a policy tweak that the Aces team put in place. Whenever the "Ready for QA" column became full, they had a policy that team members should go help out on another item in progress somewhere on the board. The policy was not to start something new, or not to go home, it was to go look for opportunities to pair or swarm. Which brings us to…

Policies

The most target-rich environment when it comes to improvement is most likely your process policies. You have policies that you may not even know are policies.

For example, how do you handle blockers? What does it even mean for something to be blocked? Should blockers count against the

WIP Limit? What's the order in which you pull items through your process? What does it mean for something to be finished in a particular workflow stage? Should we pair by default? If you step back and think about it, you probably have dozens of policies–either implicit or explicit–that affect your ability to get work done everyday. One of the biggest levers you can pull to change your process is to change these policies.

Making a policy change will, by definition, change how your process performs. Which brings us to…

The Service Level Expectation

In the Aces example earlier, their process Cycle Time went from 33 days at the 85th percentile (before WIP controls) to 14 days at the 85th percentile (after WIP controls). If, when they started with Kanban, they set an SLE of 33 days or less at the 85th percentile, then certainly that SLE was no longer valid after weeks or months of improvement.

A common question we get is "how do I know when I should adjust my SLE?" As with almost everything flow, there is no straightforward answer. We can say this: a change in SLE is probably not warranted unless there has been some other change in the process (i.e., a change in one of the DoW elements discussed up until now). When you make a change–especially what might be considered a major change (e.g., change WIP limits, remove columns, change blocker policy, etc.)–what you typically need to do is empty the board of whatever items were in progress when you made the change and start tracking Cycle Time for new items that enter the process after the change. Most likely, if the change is big enough, when the change was made will be obvious on your Scatterplot and you can accordingly adjust the historical data that you use to calculate your SLE. If it is not so obvious on your Scatterplot, then, as a rule of thumb, once you have 10-ish "new" items that have completed after the change you should have a enough data to

calculate a new SLE. Note that not all process changes will result in SLE changes. This is where the discretion of the team comes in. Use your understanding of context to decide whether an SLE change is appropriate or not.

One final thing about SLEs: you may be wondering what you do if you don't have enough historical data to calculate an SLE. While this is a very uncommon case (usually the problem is you have too much data), if you do find yourself in this situation, the answer is easy. Guess. Seriously, guess. Make a judgement about what an appropriate SLE might be and then (do you see a pattern here yet?) adjust the SLE once you have data.

One case where you might not have enough data to calculate an SLE might be if you have changed the started and end points of your process. Which brings us to…

Started and Finished Points

Let's say you've run through all the examples here and more and have run out of ideas of what process changes to make for improvement. For me, that is a pretty clear indication that it may be time to expand the boundaries of your Kanban system. Maybe you want to expand more upstream to cover the work that happens before items enter your process. Or maybe you want to expand more downstream to cover the work that happens after items leave your process. Either way, once you expand your boundaries you will almost always have all kinds of improvement questions to answer: are our WIP limits set properly? What policies do we have in place to handle upstream/downstream work? What should our new SLE be? By continually expanding your process boundaries outward, sooner or later you should get to that utopia known as true end-to-end business agility. We have yet to see that out in the wild, but we believe it exists (or at least it can).

By the way, for you skeptics out there who might say something like

we do continuous deployment/delivery so there is no room for us to expand downstream, let me ask you this: once you have delivered to your customer, how are you validating that what you delivered is valuable? Let's even give you the benefit of the doubt and say that you are validating what is/isn't valuable, then how are you tracking what changes you are making based on that feedback? My point is, if you get creative enough, you will almost certainly think of opportunities to expand the scope of your overall workstream.

Conclusion

There is no way that we could enumerate in this book all of the possible ways you might improve your Kanban system. The best we can do is give you some examples of improvement and let you experiment on your own process from there.

Further, if you haven't made a meaningful change to your process in months, then you are not doing Kanban. The essence of professionalism is showing up to work every day in the belief that you can get better. We began this chapter talking about Toyota. Toyota sees improvement as a journey and not a destination. And so should you.

Running experiments on your process is all very well and good, but how do you know if those experiments made any difference? The missing–and final piece of our Kanban puzzle–is going to be a set of basic flow metrics that will give us additional insight into the health and performance of our process. Let's discuss those next.

Endnotes

1. Steve Reid, "Ultimate Kanban: Scaling Agile without Frameworks at Ultimate Software" https://www.infoq.com/articles/kanban-scaling-agile-ultimate/

Chapter 6 - The Basic Metrics of Flow

The four flow measures to track in Kanban are:

- **WIP**: The number of work items started but not finished.
- **Cycle Time**: The amount of elapsed time between when a work item started and when a work item finished.
- **Work Item Age**: The amount of elapsed time between when a work item started and the current time.
- **Throughput**: The number of work items finished per unit of time. Note the measurement of throughput is the exact count of work items.

In Chapter 4, we talked about how important it is to have a well defined point at which work is started and a well defined point at which work is finished. For our purposes here, the reason for this importance is that all basic metrics of flow are defined in terms of those started and finished points. For a fuller discussion of this concept, I'll point you to Chapter 4. Just know that for the rest of this chapter we will assume you have a process with well-defined boundaries.

In and of themselves, the above metrics are meaningless unless they can inform one or more of the three Kanban practices. Further, these four flow measures represent only the minimum required for the operation of a Kanban system. Teams and organizations may and often should use additional context-specific measures that assist in making data-informed decisions[1].

Work In Progress

WIP: All discrete units of potential customer value that have entered a given process but have not exited.

To calculate WIP you simply count the discrete number of work items within your process boundaries as defined above. That's it: just count.
Your natural objection might be, "doesn't that mean you have to make all of your work items the same size?" After all, the work items that come through your process are of different durations, are of disparate complexities, and may require a wide mix of resources to work on them. How can you possibly account for all of that variability and come up with a predictable system by just counting work items? While that is a reasonable question, it is not something to get hung up on.

As we saw in Chapters 2 and 3, when it comes to WIP, there is no requirement that all of your work items be of the same size. There is not going to be a need for any further complexity to be added to the WIP calculation such as estimating in Story Points or assigning ideal hours to each work item. This concept is probably very uncomfortable to those of you who are used to thinking about work in terms of relative complexity or level of effort, but there is no requirement to do any kind of upfront estimation when practicing flow.

Nor is there any restriction on the level at which you track work item WIP. You can track WIP at the portfolio, project, team, individual level—just to name a few. All of these types of decisions will be completely up to you and should be made given the constraints of your context.

It should also be noted that there is a difference between WIP and WIP limits. You cannot calculate WIP simply by adding up all the WIP limits on your board. It should work that way, but in reality it does not. This result should be obvious as most Kanban boards do

not always have columns or boards that are at their full WIP limit. A more common situation is to have a Kanban board with WIP limit violations in multiple columns–or across the whole board. In either of those cases simply adding up WIP limits will not give you an accurate WIP calculation. The sad truth is there is no getting around actually counting up the physical number of items in progress to come up with your total WIP.

Bottom line, if you want to use Kanban but are not currently tracking WIP, then you are going to want to start. Sooner is better than later.

Cycle Time

In the previous section we stated that a process has specific arrival and departure boundaries and that any item of customer value between those two boundaries can reasonably be counted as WIP. Once your team determines the points of delineation that define Work In Progress, the definition of Cycle Time becomes very easy:

Cycle Time: The amount of elapsed time that a work item spends as Work In Progress.

This definition is based on one offered by Hopp and Spearman in their Factory Physics book and, we believe, holds up well in most knowledge work contexts. Defining Cycle Time in terms of WIP removes much—if not all—of the arbitrariness of some of the other explanations of Cycle Time that you may have seen (and been confused by) and gives us a tighter definition to start measuring this metric. The moral of this story is: you essentially have control over when something is counted as Work In Progress in your process. Take some time to define those policies around what it means for an item to be "Work In Progress" in your system and start and stop your Cycle Time clock accordingly.

You will also not want to overlook the emphasis on "elapsed time".

The use of elapsed time is probably very different from the guidance you have previously been given. Most other methodologies ask you to measure only the actual amount of time spent actively working on a given item (if they ask you to measure time at all). We happen to think this guidance is wrong. We have a couple of reasons why.

First, and most importantly, your customers probably think about the world in terms of elapsed time. For example, let's say that on March 1, you communicate to your customers that something will be done in 30 days. My guess would be that your customer's expectation would be that they would get their item on or before March 31. However, if you meant 30 "business days" then your expectation is the customer would get something sometime around the middle of April. We are sure you can see where that difference in expectations might be a problem.

Second, if you only measure active time, you are ignoring a large part of your flow problem. It is the time that an item spends waiting or delayed (i.e., not actively being worked) that is usually where most of your unpredictability lies. It is precisely that area that we are going to look at for most substantial predictability improvements. Remember, delay is the enemy of flow!

There is still a more important reason to understand Cycle Time. Cycle Time represents the amount of time it takes to get customer feedback. Customer feedback is of vital importance in our knowledge work world. Value itself is ultimately determined by the customer, which means your team is going to want to make sure it gets that value feedback as quickly as possible. The last thing you want is to develop something that the customer does not need—especially if it takes you forever to do so. Shortening Cycle Time will shorten the customer feedback loop. And to shorten Cycle Time, you are going to first need to measure it.

Work Item Age

Age is by far the most important of all the flow metrics to track. The reason why is covered in great detail in Chapter 1 of this book (which may be why you started reading this chapter in the first place).

The definition of WIP Age is:
Work Item Age: the total amount of time that has elapsed since an item entered a workflow.

Because Work Item Age is a measure of current time in progress for all of your current work in progress it, by definition, only applies to items that have entered but not exited the workflow. Once an item exits the workflow, then all the age that has accumulated up to that point immediately becomes converted to Cycle Time.

Throughput

I have saved the easiest metric to define for last. Simply put, Throughput is defined as:

Throughput: the amount of WIP (number of work items) completed per unit of time.

Stated a slightly different way, Throughput is a measure of how fast items depart a process. The unit of time that your team chooses for your Throughput measurement is completely up to you. Your team can choose to measure the number of items that it gets done per day, per week, per iteration, etc. For example, you might state the Throughput of your system as "three stories per day" (for a given day) or "five features per month" (for a given month).

A further thing to know about Throughput is that many agile coaches and consultants use the words "Velocity" and "Throughput" interchangeably. While Velocity can be defined in terms that are the

same as Throughput, most of the time when a coach says "Velocity" he or she means "story points per sprint". When defined in terms of story points, you should know that Throughput and Velocity are anything but synonymous.

If Throughput is how fast items depart from a process, then Arrival Rate is how fast items arrive to a process. We mention this fact here because depending on your perspective, Arrival Rate can be thought of as an analog to Throughput. For example, let's say that the "Development" step and "Test" step are adjacent in your workflow. Then the Throughput from the "Development" step could also be thought of as the Arrival Rate to the "Test" step.

Even more importantly, though, comparing the Arrival Rate of one step in your process to the Throughput in another, different step, may give you some much needed insight into predictability problems. We will be going into much more detail about this comparison in the coming chapters. However, my more immediate reason in discussing Arrival Rate is simply to point out that how fast things arrive to your process could be just as important as how fast things depart.

The Throughput metric answers the very important question of "How many features am I going to get in the next release?" At some point you are going to need to answer that question, so track Throughput and be prepared.

Conclusion

What we have shown here are just the basic metrics of flow to get you started: WIP, Cycle Time, WIP Age, and Throughput. There are most certainly other metrics that you will want to track in your own environment, but these represent the metrics common to all flow implementations. If your goals are improvement and predictability, then these are the metrics that you are going to want to track.

Endnotes

1. John Coleman and Daniel Vacanti, “The Kanban Guide” https://kanbanguides.org/

Chapter 7 - Unleashing the True Power of Kanban

Having gotten to this point in the book, you could be forgiven for thinking that Kanban is only applicable to development teams. However, we would argue that the true power of Kanban is unleashed only when we start scaling our implementation along the dimensions described in this chapter. While the implementation and optimization of a Kanban system at the team level will help you walk, the scaling of these practices will help you run. The good news is any scaling of Kanban does not require any new principles or practices. Scaling Kanban is simply the application of the same practices that you have already read about in this book, along new dimensions.

Dimensions of Scaling

There are multiple dimensions of scaling along which we can apply the practices of Kanban. Here are some ways that Kanban can be applied to achieve great results beyond the confines of a single development team.

Multiple Teams Working on the Same Product

Products are often too large for a single team to develop and maintain. Teams would often pick up features of the product to

develop by themselves. Soon, we realize that there are multiple dependencies across these teams as they are contributing to the same product. This leads to blocked work and often more features getting started. We end up in a case where the individual teams might be able to maintain their work item WIPs, but the overall Feature WIP for this set of teams keeps growing. A very common pattern that leads to this problem is teams that are split by specialization, for example teams focused on Front End, Back End, Database, QA, etc. This problem can also exist with a set of cross functional teams working on the same product.

The observant reader would probably already have guessed the authors' answer to this problem, based on earlier chapters. This is the perfect opportunity to look at these teams as a single system. We can do this by having a single representative model for the work on the product, probably a Kanban board with the features for the product laid out. The Kanban board forces two very important practices in this context - The limiting of WIP, at the feature level across these teams and the establishment of an SLE for these features.

If we actively manage the work on this board, we can ensure that none of the features age too long. We can also facilitate teams helping each other by unblocking work or even working on some features together. This usually leads to a single product backlog, instead of a backlog for each team. As a result, new work and collaboration patterns might start to emerge. Teams might start picking up work that was initially slated for another team, allowing for greater degrees of flexibility and knowledge transfer. The boundaries between the teams might start to blur. We might even realize that it might benefit the overall flow, to combine some teams and create larger teams that can better harness the newfound flexibility. Having a Feature or Epic level Kanban board across teams working on the same product can ensure the success of all teams and as a consequence, the product.

Multiple Teams and Multiple Products

It is easy to see the benefits of a single board for teams working on the same product. Their work is probably related and they have the ability to help each other finish work items. What if we have multiple teams but sets of them are focussed on multiple products? Should we still attempt to implement Kanban at a higher level? The answer is still 'Yes'.

A portfolio level Kanban board can ensure that our execution is in line with our current strategy. If we lay out all the initiatives/features/epics (whichever level makes sense here) that the organization is currently working on, what information might we gain? We will very quickly find out what is our current WIP on these high level items across the board. We will also find out if the focus for the teams matches the strategy for the organization. When each product has its own backlog it is very likely that the highest priority item in one product's backlog is not only lower priority for the org but also runs counter to the overall strategy. Furthermore, if the team working on this product is very efficient, the org will produce more of what it does not want than what it does consider strategically important.

Having a portfolio-wide Kanban implementation helps us become both efficient and effective at the org level. We are able to make the flow of work efficient by monitoring flow metrics and improving them over time. We are also able to produce more of the right things, by making sure that the distribution of that active and upcoming work aligns with our overall strategy. The frequency at which a board at this level is managed might not be the same as a team level board. It might be less often, but often enough so we can make adjustments in time.

Higher Levels of The Organization

Similar to how multiple teams can get out of sync with the overall strategic direction, entire departments can fall into the same trap. The strategic goals and priorities of the organization can often be mismatched from the priorities of individual departments. Scaling Kanban to the organization level where high level objectives are represented on a board can help avoid this. A board at every level can help tie the high-level organization objectives to department level work items, which can in turn be tied to product and team level features and consequently to work items on the team boards. The first and immediate result of this is the understanding of whether our current activities align with our organizational strategy and goals.

A Kanban board is not complete without a Service Level Expectation and explicit policies.This is where the true benefits of having boards at higher levels start coming in. The consistent problem with organizational objectives is how long running and vague they can be. Applying explicit exit criteria and identifying a 'finished' stage can help make sure that the overall objectives are not vague. An SLE can also ensure that we do not have long running objectives that are outdated, yet still active. Actively managing items on a board at any level of the organization requires the same discipline as team level boards. The higher the level of the board, the greater impact it would have on efficiency, effectiveness and predictability of the organization.

Active management of work items on higher levels can extend the benefits of Kanban across the board. Aging of the work items on these boards can reveal WIP, sizing or dependency issues. At higher levels of the organization, we often have the right people involved to solve these issues. We are able to bring teams and departments together to work on our collective priorities and ensure delivery of value to customers.

Extending Upstream and Downstream

So far we have talked about scaling up to include more teams, products and departments in our Kanban system. Another dimension is scaling out to include more activities. Many teams would often start implementation of Kanban from the point someone starts implementing a solution to the point at which the solution is ready to be delivered. This is a great place to start, but if left alone, this system might end up being suboptimal. We are likely to create a highly optimized sub-system which produces output faster than we can deliver it to customers. It might also encourage upstream processes to try and front load the system and create waste.

When we have stabilized Kanban between the lines, it is time to scale the system out to include upstream and downstream activities. We need to redefine the start and finish points within which our system runs. Once the process of creating the solution for customer problems is optimized, we should include the processes of understanding the problem (upstream) and delivering the solution (downstream) as well as getting feedback on the delivered solution (downstream which informs upstream). As we move the start and finish lines out, we apply the same practices as described in this book - Identifying work items, defining stages, limiting WIP, explicit policies and having an SLE. Over time we are able to extend our Kanban practice from the point anyone in the org starts to understand the problem to the point we get feedback from customers that the problem has or has not been solved.

Other Departments in the Organization

Very often within an organization only one department will start with Kanban. Another mode of scaling Kanban is to cross-pollinate the learnings to other departments. If the IT department has seen great benefits from the implementation of Kanban systems, some of the departments they regularly interact with are great places to

spread the benefits of Kanban. We have seen firsthand, departments like customer support, client activations and product strategy utilize the methods that were first adopted by development teams. These adoptions have led to greater efficiency, effectiveness and predictability for these departments.

Scaling in this manner often coincides with the other scaling dimensions already mentioned. Having an organization level Kanban system can act as a vehicle for spreading the benefits of Kanban to the various departments.

Conclusion

When it comes to scaling Kanban, there are multiple dimensions to consider. We have described a few of these here. We can unleash the true power of Kanban by scaling along these dimensions. Often one of these dimensions leads to the need to work along a different one. We recommend that the organization pick the one that seems easiest to make progress with and observe some success before embarking down a different dimension. Kanban is a strategy that is applicable across multiple types of teams and levels of an organization. Do not be afraid to play outside the lines.

An example of scaling is the Ultimate Software Scaling Kanban case study. This excerpt describes how the organization combined some of the scaling approaches described here -

'Adopting Agile techniques has provided the benefits of increased productivity and predictability. For an overall perspective though, Ultimate Software is in a waterfall sandwich. The Agile development organization sits in the middle of traditional sales and support organizations and traditional deployment and activation organizations. As a part of the next evolution of Agile and flow based thinking at Ultimate Software, we are expanding out to organizations that flank development. Ultimate's culture that en-

courages managers and employees to experiment and make the right decisions for Ultimate, has aided greatly in spreading the principles outside of core development. Departments within Ultimate Software have started pulling the services of the Agile coaches within development to help them with the same principles.

Our closer engagement with Product Strategy and the ability to give them a higher degree of predictability has vastly improved Development's ability to assist with support issues without interrupting active work. Tier 3 support has also adopted Kanban practices in order to improve their ability to support our customers. Product Strategy is able to utilize the predictability and productivity gains of Development to provide better guidance to Sales on upcoming products and features. As we continue to improve the predictability that we can provide Sales, we can start creating feature requests and priorities in conjunction with Sales. Features can then be pulled all the way through the value stream and tracking of Cycle Time and Throughput can allow us to make and keep more accurate commitments to our customers.

While the upstream expansion helps us get better at the creation of value, expanding downstream to deployment and activations is where we can improve the delivery of value to our customers. As Ultimate Software has started working on new products, we have pulled deployment activities onto the teams. For our older products, we have always done a handoff to our Sass deployment group. We broke the 'over the wall' mentality by embedding deployment engineers on the development teams for new products and helping them educate the rest of the team on maintaining their own deployment pipelines. The teams were initially concerned about taking on the additional responsibility. Those fears have abated as the teams have realized the support that is available to them from the rest of the organization. This practice has also greatly reduced the occurrences of production environment surprises. Since the teams help build the environments that they deploy code to, the code does not behave unexpectedly when pushed to production. These teams are

supported by three groups outside of Product Engineering. Groups that manage the Build and Deployment infrastructure for the products being developed have also adopted Kanban principles and started measuring Cycle Times for making infrastructure available to teams. They have established SLAs for different types of requests and have become predictable with these metrics.

We can now see a feature make its journey all the way from a request generated in Sales to Product Strategy, to Development and finally to Production. Once we are able to track the progress of a feature in this manner, we can start identifying opportunities for improvement in the inception-to-delivery cycle. The organization as a whole can identify where features get stuck and apply our understanding of flow to eliminate the time features have to wait in queues across the entire organization.

Another aspect that is downstream from the development and even the deployment group is activations. Activations is the group that helps a new customer go live with Ultimate Software's products. The activation process can take up to a year and can involve multiple teams. Every day that a customer is in the activation phase, Ultimate Software is investing time, but not receiving full revenue. This is an area that can use the benefits that the Development Organization has gained from flow and Agile practices. Development has started working with Activations to share the principles and practices that have made a positive difference in the predictability and speed of completion for deliverables.

Moving Kanban outside the lines is the next large step for Ultimate Software. We have already started moving in this direction through our work with support and deployment teams. Ultimate continues to scale out its Agile implementation without using any of the established frameworks. Setting up the right channels of communication and visualizing our work in a manner that is easily understood by all is at the crux of how Ultimate has been able to successfully adopt and evolve Agile at scale.'

Chapter 8 - Thoughts On How To Get Started

As you read this book, you are probably already forming a decent idea of what a Kanban system would look like in your context. The harder part often is just getting started. Here we describe an approach to help you get started. There are multiple ways to start with Kanban, the one described below is the one we have seen to be most successful. Elements of this approach have been described in previous chapters (especially chapter 4). We have denoted the chapters that go deeper into the concept alongside each step. Please refer back to those chapters for deeper discussion of these steps.

Starting Steps

Define representations of value (Chapter 4)

First step is to understand what are the work items that represent the individual bits of value that flow through your system. These are items we will deliver and validate with customers.

Define start and finish points (Chapter 4)

Next we need to determine when we consider these items to have started and when we are "done" with these items. These form the boundaries within which we will manage the Kanban system. These boundaries will and should change over time, but we always need to be aware of what they are so we can effectively observe and improve the system.

Determine all activities that help create value (Chapter 4)

As an item moves from the start point of a system, to the finish point of a system, we perform action on it to turn it into deliverable value. We need to determine what these activities are. For a piece of software these might be - Understanding the problem, Technical design, Creating the code, Code Review, Unit testing, Integration Testing, Build and Deployment, Customer Validation. Systems will have a varying number of activities that contribute to a work item becoming a piece of deliverable value. As we list them out we get an idea of the flow of the system.

Map determined activities and work items to process stages (Chapter 4)

So far we have work items, start and finish points and activities. We are starting to get some understanding of flow through the system. The next step is to figure out what are the major stages between our start and finish points that work items flow through. We can map the activities defined in the previous steps to those stages. These activities are part of our set of policies for the system. They become the exit criteria for each stage - A work item cannot pass a stage unless the mapped activities have been completed. We can also map currently active items in the system to these stages using the exit criteria. This gives us an idea of the current state of the system.

Decide on a method of limiting WIP (Chapters 1 and 4)

The single most important theme of this book is - All Kanban practices can be derived from the singular motivation of not wanting items to age unnecessarily. The starting point for controlling

age and as a consequence Cycle Time is limiting WIP. There are multiple ways of limiting WIP - Adding WIP limits to each stage, adding an overall WIP limit to the entire system, limiting WIP based on the number of people on the team etc. This step is to pick one of these methods and ensure that we have an understanding of how we will be limiting WIP in order to prevent aging.

Select a starting SLE (Chapter 2)

Next, we need to create a common understanding of 'How long does it take us to get work done?' via an SLE. We can do this by analyzing the Cycle Time distribution of items that have previously finished. Based on this distribution we can pick an SLE that represents the amount of time it takes the majority (70%, 85%, 95% etc.) of items to get done. This SLE becomes the yardstick for items that are active in the system.

Set expectations on inspecting and adapting policies (Chapter 5)

Doing all the work to define and set up a Kanban system can often give teams the false confidence that they have discovered the perfect way to execute and track work. This is just the beginning of the journey and we need to be explicit about that. Leaders and teams should expect that as they learn more about their work and workflow, the stages of work and policies will change. As we evolve the system it will be a better fit to our particular context.

Actively manage work to improve the system (Chapter 3)

With the system now set up, we need to focus on the operation of the workflow. We need to, on a daily basis, manage work

items with the aim of not letting them age unnecessarily. This active management can come in many forms - Pairing, Swarming, Removing Blockers, Right-Sizing and others not discussed in this book. As we do this on a daily basis, we create a feedback loop into the process. This feedback loop helps adjust our policies and workflow to improve the system overall.

Conclusion

There are many ways to get started with Kanban. This chapter provides a blueprint of the approach that we have seen work (ex post). The majority of the work to get started is around understanding your current activities and defining your workflow and policies. Once we have a common understanding of these, we can start watching the reasons why items age in our system and make the appropriate adjustments. We take the one time revolutionary, yet simple step of defining a WIP limited system so that we can take multiple evolutionary steps of improving the system.

Epilogue - Professionalism and ProKanban.org

What does the future of Kanban look like?

You've had a great view into how Kanban got started, and how it should look, so what is next for Kanban?

I don't know how long the changes the pandemic brought to the working world will last but there is one that I hope sticks- the relentless prioritization of self-care. Working from home and across time zones has made it easier to work from dusk till dawn, with no breaks, and never seeing the actual sunlight, which also means that we are burnt the fuck out. Home life and work life are so inextricably blended that they no longer have a start and stop. And this is where Kanban shines.

We need a mechanism (whether we are a team of one or a team of twenty) to organize around the work to be done. To see our priorities, to see what's stuck, and to focus our time on getting things done. A new term for this day-to-day deficit is "time poverty" and it's having a profound effect on our physical health, well-being, and productivity.

I hear developers talk about how it feels to be part of "the feature factory" as Melissa Perri describes it, where we crank out code without getting time to pause and understand what problem we are trying to solve. I almost always hear them say that they never get time to do deep thinking. Real focused time to creatively solve problems. Down time to address growing tech debt. Opportunity to learn new tech or mentor others. And the result of this is that

people quit and move on hoping to find a pace that doesn't burn them out. A pace they rarely find.

So is it possible? The place where there's 100 things in the backlog and we aren't drinking from the fire hose trying to keep up? I believe there is. It's the places where they are using Kanban.

There will always be new methods (and new names for old methods) of how to deliver knowledge work, but it's hard to argue with the simple need for focus. We spend a lot of time in the agile community talking about team health and team morale, but I have never seen a team find better balance and have better control over the volume of work in front of them every day than when they have the control to create the policies that allow them to shut off at the end of the day and return the next day with a clear understanding of what to work on next.

Teams that have control over how they work are happier and more productive. They know how to contribute when they have capacity. They know how to help when work is blocked. And they have the autonomy to adjust the things that aren't working for them as they go. This is how we engage employees and prevent burnout- with Kanban.

A few years before the pandemic, I started working with a large security organization to roll out Kanban. What I learned quickly is that the security space has one of the highest levels of turnover because they are constantly fighting fires. Everything is urgent. Burnout is high. There is rarely downtime. And ultimately people do not last long at this pace.

The teams I coached wanted Kanban as a way to balance their workload and create a predictable SLE for delivery to their "customers" but they were pleasantly surprised to find that that made their teams happier. It reduced the amount of turnover in the team, it allowed the team to balance long running internal projects with the everyday requirements of a security organization and it did all of this indefinitely. It wasn't one successful sprint or one slow week

after a week of insanity- it was a pace that created balance in a normally hectic storm of urgency.

What is important here is that this "time poverty" trickles over into everything we do outside work as well. We are out of energy, out of patience and out of the spark that makes us great at what we do. The answer here is rarely about saying no- in many of these cases that may not even be possible. This is about focus, flow, and efficiency- and having control over the process to make that possible..

I hope that what is next for our industry and more importantly- the people in it- is that we demand a different way of working. A way that allows us to bring our best selves, and our best work into what we do- and I think that is best done with Kanban.

Kanban gives us the ability to be better at what we do by enabling us to fully focus. Kanban gives us the ability to be a better team member by collaboratively organizing around the work to be done. Kanban gives us the opportunity to be better professionals by seeking endless improvement of our systems and how we deliver value. And all of this makes us better humans when we shut our laptop.

Appendix A - An Introduction to Little's Law

Note: this appendix originally appeared as chapter 3 of Daniel Vacanti's book "Actionable Agile Metrics for Predictability". It has been edited from that original form to make it more inline with goal of this book.

Chapter 6 dealt with the basic metrics of flow: WIP, Cycle Time, and Throughput. In what may be one of the most miraculous results in the history of process analysis, these three metrics are intrinsically linked by a very straightforward and very powerful relationship known as Little's Law:

Average Cycle Time = Average Work In Progress / Average Throughput

If you have ever seen Little's Law before, you have probably seen it in the form of the above equation. What few Agile practitioners realize, however, is that Little's Law was originally stated in a slightly different form:

Average Items In Queue = Average Arrival Rate * Average Wait Time

This fact is important because different assumptions need to be satisfied depending on which form of the law you are using. And understanding the assumptions behind the equation is the key to understanding the law itself. Once you understand the assumptions, then you can use those assumptions as a guide to some process policies that you can put in place to aid predictability.

The math of Little's Law is simple. But this appendix is about how we do not care about the math. What we do care about—and I cannot stress this point enough if we want to gain a greater appreciation of the law's applicability to our world—is looking far beyond the elegance of the equation to get a deeper understanding of the background assumptions needed to make the law work. That is where things get more complicated, but it is also where we will find the greatest benefit. A thorough comprehension of why Little's Law works the way it does is going to be the basis for understanding how the basic metrics of flow can become predictably actionable.

We Need a Little Help

First, some background.

Dr. John Little spent much of his early career studying queuing systems. In fact, one of the best definitions of such a queuing system comes from Dr. Little himself:

"A queuing system consists of discrete objects we shall call items, which arrive at some rate to the system. The items could be cars at a toll booth, people in a cafeteria line, aircraft on a production line, or instructions waiting to be executed inside a computer. The stream of arrivals enters the system, joins one or more queues and eventually receives service, and exits in a stream of departures. The service might be a taxi ride (travelers), a bowl of soup (lunch eaters), or auto repair (car owners). In most cases, service is the bottleneck that creates the queue, and so we usually have a service operation with a service time, but this is not required. In such a case, we assume there is nevertheless a waiting time. Sometimes a distinction is made between number in queue and total number in queue plus service, the latter being called number in system."

The diversity of domains that he mentions here is extraordinary. While he does not specifically mention software development or

knowledge work in general, I am going to suggest that these areas can also be readily modeled in this way.

In 1961, Dr. Little set out to prove what seemed to be a very general and very common property exhibited by all queuing systems. The result that he was researching was a connection between the average Arrival Rate of a queue, the average number of items in the queue, and the average amount of time an item spent in the queue (for the purpose of this appendix, when I say "average" I am really talking about "arithmetic mean"). Mathematically, the relationship between these three metrics looks like:

Equation (1): $L = \lambda * W$

Where:

L = the average number of items in the queuing system.

λ = the average number of items arriving per unit time.

W = the average wait time in the system for an item.

Notice that Equation (1) is stated strictly in terms of a queuing system's Arrival Rate. This point is going to be of special interest a little later in this chapter.

Also notice that—if it is not obvious already—Little's Law is a relationship of averages. Most knowledge work applications and discussions of the law neglect this very important detail. The fact that Little's Law is based on averages is not necessarily good or bad. It is only bad when people try to apply the law for uses that were never intended.

Dr. Little was the first to provide a rigorous proof for Equation (1) and, as such, this relationship has since been known as Little's Law. According to him, one of the reasons why the law is so important is the fact that (emphasis is mine): "L, λ, and W are three quite different and important measures of effectiveness of system performance, and Little's Law insists that they must obey the 'law.'… *Little's Law locks the three measures together in a unique and consistent way for any system in which it applies. Little's*

Law will not tell the managers how to handle trade-offs or provide innovations to improve their chosen measures, but it lays down a necessary relation. As such, it provides structure for thinking about any operation that can be cast as a queue and suggests what data might be valuable to collect."

The great advantage of Little's Law is the overall simplicity of its calculation. Specifically, if one has any two of the above three statistics, then one can easily calculate the third. This result is extremely useful as there are many situations in many different domains where the measurement of all three metrics of interest is difficult, expensive, or even impossible. Little's Law shows us that if we can measure any two attributes, then we automatically get the third.

To illustrate this point, Dr. Little used the very simple example of a wine rack. Let's say you have a wine rack that, on average, always has 100 bottles in it. Let's further say that you replenish the rack at an average rate of two bottles per week. Knowing just these two numbers (and nothing else!) allows us to determine how long, on average, a given bottle spends sitting in the rack. By applying Equation (1), we have L equal to 100 and λ equal to 2. Plugging those numbers into the formula tells us that a given wine bottle spends, on average, 50 weeks in the rack.

Before we get much further, it is worth exploring what necessary contextual conditions are required for the law to hold. When stated in the form of Equation (1) the only assumption necessary is that the system under consideration has some guarantee of being in a steady state. That's it. Really, that's it. To illustrate the things we do not need, notice that we can arrive at the wine rack result without tracking the specific arrival or departure dates for each or any individual bottle. We also do not need to know the specific order that the bottles were placed in the rack, or the specific order that the bottles were taken off the rack. We do not need to understand anything fancy like the underlying probability distributions of the Arrival and Departure Rates. Interestingly, we do not even need to

track the size of the bottles in the rack. We could have some small 20cl bottles or some large 2 litre bottles in addition to the more standard 750ml bottles. The variation in size has no impact on the basic result. (You should know that, in the interest of thoroughness, I am in the process of independently verifying this wine rack result on my own. Rest assured that no detail has been overlooked in the research of this book.)

As remarkable as all of this may be, the mathematics are not really what is important for our purposes here. What is important is that we acknowledge that the fundamental relationship exists. Understanding the inextricable link among these metrics is one of the most powerful tools at our disposal in terms of predictable process design.
But before we can get into how Little's Law can help us with predictability, it is probably helpful to first state the relationship in more familiar terms.

Little's Law from a Different Perspective

In the late 1980s (or early 1990s depending on whom you ask) Little's Law was usurped by the Operations Management (OM) community and was changed to emphasize OM's focus on Throughput. The OM crowd thus changed the terms in Little's Law to reflect their different perspective as shown by Equation (2):

Equation (2): Cycle Time = Work In Progress / Throughput

Where:

1. Cycle Time (CT) = the average amount of time it takes for an item to flow through the system.

2. Work In Progress (WIP) = the average total inventory in the system.

3. Throughput (TH) = the average Throughput of the system.

In the interest of completeness, it is ok to perform the algebra on Little's Law so that it takes the different, yet still valid forms:

Equation (3): TH = WIP / CT

and

Equation (4): WIP = CT * TH

Where CT, WIP, and TH are defined the same way as in Equation (2).

Because of its roots in Operations Management, the Lean and Kanban knowledge work community has adopted this "Throughput" form of Little's Law as their own. If you have seen Little's Law before, you have almost certainly seen it in the form of Equation (2)—even though Equation (2) does not represent the law's original format.

The upshot of Little's Law is that, in general, the more things that you work on at any given time (on average) the longer it is going to take for each of those things to finish (on average), ceteris paribus. As a case in point, managers who are ignorant of this law panic when they see that their Cycle Times are too long and perform the exact opposite intervention of what they should do: they start more work. After all, they reason, if things take so long, then they need to start new items as soon as possible so that those items finish on time—regardless of what is currently in progress. The result is that items only take longer and longer to complete. Thus, managers feel more and more pressure to start things sooner and sooner. You can see how this vicious cycle gets started and perpetuates itself. After studying Little's Law, you should realize that if Cycle Times are too long then the first thing you should consider is lowering WIP. It feels uncomfortable, but it is true.

What Dr. Little demonstrated is that the three flow metrics are all essentially three sides of the same coin (if a coin could have three sides). By changing one of them, you will affect one or both of the

other two. In other words, Little's Law reveals what levers that we can pull when undertaking process improvement. Further, as we are about to see, Little's Law will suggest the specific interventions that we should explore when our process is not performing the way we think it should.

At the risk of repeating myself, what I am talking about here is simple, incontrovertible mathematical fact. A change in one metric results in a change in the others. Most companies that I talk to that complain of poor predictability are almost always ignorant of the negative implication of too much WIP on Cycle Time or Throughput. Ignore this correlation at your own peril.

It is all about the Assumptions

This is all straightforward enough so far, right? Well, unfortunately, it is not. Remember I said at the outset that Little's Law is deceptively simple? Here is where things get more complicated.

It is easy to see from a purely mathematical perspective that Equation (1) is logically equivalent to Equation (2). But it is more important to focus on the difference between the two. As I mentioned earlier, Equation (1) is expressly stated in terms of the *Arrival Rate* to the system whereas Equation (2) is expressly stated in terms of the *Departure Rate* from the system. This emphasis on Throughput in Equation (2) probably seems more comfortable to us as it reflects the usual perspective of a knowledge work process. Typically, in our context, we (and more importantly, our customers) care about the rate at which we are finishing our work (even though, as we shall soon see, we should care just as much about the rate at which we start work). What is nice to know is that Little's Law can morph to match this required perspective.

At first glance, this change may not otherwise seem all that significant. However, this transformation from the perspective of arrivals

to the perspective of departures has a profound impact in terms of how we think about and apply the law. When we state Little's Law in terms of a system's Throughput then we must also immediately consider what underlying assumptions must be in place in order for the departure-oriented law to be valid.

Earlier when I first introduced Equation (1) I had stated that there was really only one assumption that needed to be in place for it to work. Well, in the interest of completeness, technically there are three. For Equation (1) we need:

1. A steady state (i.e., that the underlying stochastic processes are stationary)
2. An arbitrarily long period of time under observation (to guarantee the stationarity of the underlying stochastic processes)
3. That the calculation be performed using consistent units (e.g., if wait time is stated in days, then Arrival Rate must also be stated in terms of days).

By the way, the point here is to not give you an advanced degree in statistics or queuing theory. Do not worry if you do not know what "stochastic" or "stationary" means. You do not need to. As I have just said, I mention these things for completeness only.

When we shift perspective to look at Little's Law from the perspective of Throughput rather than from the perspective of Arrival Rate, however, we also need to change the assumptions necessary for the law to be valid. This point is so important, I want to place it in its own callout:

Looking at Little's Law from the perspective of Throughput rather than from the perspective of Arrival Rate necessitates a change in the assumptions required for the law to be valid.

In systems where WIP never goes to zero (please see AAMFP for a fuller discussion of the case when WIP is allowed to go to zero), then the assumptions about our process that are necessary to make Little's Law (in the form of Equation (2)) work are:

1. The average input or Arrival Rate (λ) should equal the average output or Departure Rate (Throughput).
2. All work that is started will eventually be completed and exit the system.
3. The amount of WIP should be roughly the same at the beginning and at the end of the time interval chosen for the calculation.
4. The average age of the WIP is neither increasing nor decreasing.
5. Cycle Time, WIP, and Throughput must all be measured using consistent units.

The first two assumptions (#1 and #2) comprise a notion known as Conservation of Flow. The second two assumptions (#3 and #4) speak to the notion of system stability.

The last assumption (#5) is necessary for the math (and any corresponding analysis) to come out correctly (you will notice this is the same assumption necessary when stating the law in terms of arrivals). The necessity for using consistent units when performing a Little's Law calculation should be intuitively obvious, but it is fairly easy to get tripped up over this. When we say "consistent" units what we are really saying is, for example, if we are measuring average Cycle Time using the unit of time "day", then the average Throughput must be in the form of the number of items per that same unit of time (day), and the average WIP must be the average amount of items for one unit of time (day). As another example, if you want to measure average Throughput in terms of items per week (i.e., the unit of time here is "week"), then average Cycle Time must be stated in terms of weeks, and average WIP must be the average for each week.

You might think I am wasting your time by mentioning this, but you would be surprised how many teams miss this point (one is immediately reminded of when NASA slammed an orbiter into the side of Mars because one team used metric units while another used English units—moral of the story: don't do that). For example, I saw one Scrum team that was measuring their velocity in terms of story points per sprint (as Scrum teams are wont to do, more's the

pity). For their Little's Law calculation, they proceeded to plug in their velocity number for Throughput, their WIP number as total number of user stories (actual stories—not story points) completed in the sprint, and expected to get a Cycle Time number in days. You can imagine their surprise when the numbers did not come out quite the way that they expected.

Assumptions as Process Policies

Understanding these foundational assumptions is of monumental importance. Despite what many people will tell you, the true power of Little's Law is not in performing the mathematical calculation by plugging numbers into its formula. Even though I have spent so much time on it already, I want you to forget about the arithmetic. In truth, most of us will never have a need to compute Little's Law. As I mentioned Chapter 6, the four flow metrics' data is so easy to capture that you should never have to compute one of them—just go look at the data!

Rather, the true power of Little's Law lies in understanding the assumptions necessary for the law to work in the first place. If there are three things that I want you to have taken away from this conversation about Little's Law they are:

1. It is all about the assumptions.
2. It is all about the assumptions.
3. It is all about the assumptions.

Every time you violate an assumption of Little's Law your process becomes less predictable. Every time. This increased unpredictability may manifest itself as longer Cycle Times or more process variability or or less Throughput or higher WIP or all of the above. Worse still, these violations may not even immediately show up in your data. The whole time you are violating Little's Law your data may be showing you a rosier picture of the world than is really

occurring. The danger here is that you may be basing some forecast on this overly optimistic view—only to find that things are much worse than they seemed.

Of course, we live in the real world and there are going to be times when violating these assumptions is going to be unavoidable or even necessary. But that is exactly why it is all the more important to understand the implications when these violations occur. There are always going to be things that happen to us that are outside of our control. However, the last thing we want to do is compound those uncontrollable events by allowing bad things to happen that were in our control and could have been easily prevented. Control what you can control and then try to eliminate or mitigate the things you cannot.

The above assumptions (especially the first four) are going to help us do just that. We can use these assumptions as the basis for some simple policies that will govern the operation of our process. These policies will serve to control the things that we can control. These policies will serve to make our process more predictable.

Based on the assumptions above, some process policies might include (but certainly would not be limited to):

- We will only start new work at about the same rate that we finish old work.
- We will make every reasonable effort to finish all work that is started and minimize wasted effort due to discarded work items (this will necessitate some notion of late-binding "commitment").
- If work becomes blocked, we will do everything we can do unblock that work as expeditiously as possible.
- We will closely monitor our policies around the order in which we pull items through our system so that some work items do not sit and age unnecessarily.

The design of your process is really just the sum of all the policies you have in place. How well your system performs or does not perform is directly attributable to those policies and to how well you adhere or do not adhere to them. When I talk about designing for predictability, what I am talking about is giving you some insights into appropriate policies that you can build into the day to day operation of your process. These policies will serve to normalize and stabilize your system in order to give your process the predictability that you are looking for. It is only from this stable base that we can even hope to implement real, long-lasting process improvement.

As Frank Vega so often likes to say, "your policies shape your data and your data shape your policies". The policies that I have mentioned here will in no small way influence the data that you collect off of your process. That is a good thing, by the way. It is a good thing because that data in and of itself is potentially going to further suggest where our process policies are deficient. It is this virtuous cycle that I am talking about when I say "actionable metrics for predictability".

Kanban Systems

From a WIP perspective, it may seem that running a Kanban system guarantees Little's Law's assumptions are taken care of. There are several reasons why that may not be the case:

1. It is possible that changing WIP limits may have no effect on total average WIP (e.g., decreasing or increasing a WIP limit after a clear systemic bottleneck). This may be one reason you do not get the "forecasted" behavior you might expect from Little's Law.
2. Setting a WIP limit is not necessarily the same as limiting Work In Progress. I cannot tell you how many teams I come across that set WIP Limits but then routinely violate them. And violate them egregiously.

3. Average WIP over a time period is highly dependent on pull policies in place. E.g., are as many items as possible pulled in order to satisfy WIP limits at all times?

The point here is that if you are using a Kanban system, you cannot just simply add up all the WIP Limits on your board and think that you have calculated WIP for your process. You are going to have actually track physical WIP.

Lastly, most people think that Little's Law is the single greatest reason to implement a Kanban-style Agile process. While I would not strictly disagree with that statement, I would offer a better way of stating it. I would say that Little's Law is the single greatest reason to move to a more WIP-controlled, pull-based, flow process. The thing is, once we do that, we can then start to use Little's Law as our guide for process predictability.

Size Does Not Matter

I have one last topic I want to cover before wrapping up. Notice how in the assumptions for Little's Law I made no mention a requirement for all work items to be of the same size. That is because no such requirement exists. Most people assume that an application of Little's Law specifically—and limiting WIP in general—necessitates that all work items be of the same size. That is simply not true (see Chapter 3).

The first reason that work items' size does not matter is because with Little's Law we are dealing with relationships among averages. We do not necessarily care about each item individually, we care about what all items look like on average.

Secondly, and more importantly, the variability in work item size is probably not the variability that is killing your predictability. Your bigger predictability problems are usually too much WIP, the frequency with which you violate Little's Law's assumptions, etc.

Generally, those are easier problems to fix than trying to arbitrarily make all work items the same size. Even if you were in a context where size did matter, it would be more about right-sizing your work and not same-sizing your work (Chapter 3).

Forecasting

All this time you may have been expecting me to say that once you understand Little's Law all you need to do is to plug in the numbers and out will pop the forecasting result that you are looking for (à la Newton's F = ma or Einstein's $E = mc^2$). However, nothing could be further from the truth.

In this regard, it is important to know that Little's Law is *only* concerned with looking backward over a time period that has completed. It is not about looking forward; that is, is not meant to be used to make deterministic predictions. As Dr. Little himself says about the law, "This is not all bad. It just says that we are in the measurement business, not the forecasting business".

This point requires a little more discussion as it is usually where people get hung up. The "law" part of Little's Law specifies an exact relationship between average WIP, average Cycle Time, and average Throughput, and this "law" part only applies only when you are looking back over historical data. The law is not about—and was never designed for—making deterministic forecasts about the future. For example, let's assume a team that historically has had an average WIP of 20 work items, an average Cycle Time of 5 days, and an average Throughput of 4 items per day. You cannot say that you are going to increase average WIP to 40, keep average Cycle Time constant at 5 days and magically Throughput will increase to 8 items per day—even if you add staff to the keep the WIP to staff ratio the same in the two instances. You cannot assume that Little's Law will make that prediction. It will not. All Little's Law will say is that an increase in average WIP will result in a change to one or

both of average Cycle Time and average Throughput. It will further say that those changes will manifest themselves in ways such that the relationship among all three metrics will still obey that law. But what it does not say is that you can deterministically predict what those changes will be. You have to wait until the end of the time interval you are interested in and look back to apply the law.

But that restriction is not fatal. The proper application of Little's Law in our world is to understand the assumptions of the law and to develop process policies that match those assumptions. If the process we operate conforms—or mostly conforms—to all of the assumptions of the law then we get to a world where we can start to trust the data that we are collecting off of our system. It is at this point that our process is probabilistically predictable. Once there we can start to use something like Monte Carlo simulation on our historical data to make forecasts and, more importantly, we can have some confidence in the results we get by using that method.

There are other, more fundamental reasons why you do not want to use Little's Law to make forecasts. For one thing, I have hopefully by now beaten home the point that Little's Law is a relationship of averages. I mention this again because even if you could use Little's Law as a forecasting tool (which you cannot), you would not want to as you would be producing a forecast based on averages. There are all kinds of reasons why you should not forecast based on averages—too many to go into here (see Dr. Savage's book, "The Flaw of Averages"). It turns out we can do better than averages. When collecting metrics data and there are going to be much better tools at our disposal when we are ready to do forecasting.

Having said all that, though, there is no reason why you cannot use the law for quick, back-of-the-envelope type estimations about the future. Of course you can do that. I would not, however, make any commitments, staff hiring or firing decisions, or project cost calculations based on this type of calculation alone. I would further say that it is negligent for someone to even suggest to do so. But this simple computation might be useful as a quick gut-check to decide

if something like a project is worth any further exploration.

Remember that being predictable is not completely about making forecasts. The bigger part of predictability is operating a system that behaves in a way that we expect it to. By designing and operating a system that follows the assumptions set forth by the Little's Law, we will get just that: a process that behaves the way we expect it to. That means we will have controlled the things that we can control and that the interventions that we take to make things better will result in outcomes more closely aligned with our expectations.

Conclusion

I know I have said it before, but I need to say it again: Little's Law is not about understanding the mathematics of queuing theory. It is about understanding the assumptions that need to be in place in order for the law to work. We can use those assumptions as a guide, or blueprint, or model for our own process policies. Whenever your process policies are in violation of the assumptions of Little's Law then you know that you have at least diminished—or possibly eliminated—your chance of being predictable.

As you operate your process think about the times and reasons why work flows in at a faster rate than work flows out. Think about why items age unnecessarily due to blockages or poor pull policies. Think about why work is abandoned when only partially complete (and how you account for that abandonment). Think about how these occurrences are violating the assumptions Little's Law and how they are ultimately affecting your ability to be predictable. But more importantly, think about how your understanding of Little's Law should result in behavior changes for you and your team. When violations of Little's Law occur, it is usually because of something you did or chose (intentionally or otherwise) not to do.

Remember, you have much more control over your process than

you think you do.

Bibliography

Bertsimas, D., D. Nakazato. The distributional Little's Law and its applications. Operations Research. 43(2) 298–310, 1995.

Brumelle, S. On the relation between customer and time averages in queues. J. Appl. Probab. 8 508–520, 1971.

Coleman, John and Vacanti, Daniel S. "The Kanban Guide" Kanban-Guides.org, 2020.

Deming, W. Edwards. The New Economics. 2nd Ed. The MIT Press, 1994.

Deming, W. Edwards. Out of the Crisis. The MIT Press, 2000.

Glynn, P. W., W. Whitt. Extensions of the queuing relations L = λ W and H = λ G.

Operations Research. 37(4) 634–644, 1989.
Goldratt, Eliyahu M., and Jeff Cox. The Goal.
2nd Rev. Ed. North River Press, 1992.

Heyman, D. P., S. Stidham Jr. The relation between customer and time averages in queues. Oper. Res. 28(4) 983–994, 1980.

Hopp, Wallace J., and Mark L. Spearman. Factory Physics. Irwin/McGraw-Hill, 2007.

Little, J. D. C. A proof for the queuing formula: L = λ W. Operations Research. 9(3) 383–387, 1961.

Little, J. D. C., and S. C. Graves. "Little's Law." D. Chhajed, T. J. Lowe, eds. Building Intuition: Insights from Basic Operations Management Models and Principles. Springer Science + Business Media LLC, New York, 2008.

Ripley, Ryan and Miller, Todd. Fixing Your Scrum. The Pragmatic Programmers LLC, 2020.

Reid, Steve and Singh, Prateek and Vacanti, Daniel "Ultimate Kanban: Scaling Agile without Frameworks at Ultimate Software" Infoq.com, 2016.

Reinertsen, Donald G. Managing the Design Factory. Free Press, 1997.

Reinertsen, Donald G. The Principles of Product Development Flow. Celeritas Publishing, 2009.

Savage, Sam L. The Flaw of Averages. John Wiley & Sons, Inc., 2009.

Schwaber, Ken and Sutherland, Jeff "The Scrum Guide" Scrumguides.org, 2020.

Shewhart, W. A. Economic Control of Quality of Manufactured Product, 1931.

Shewhart, W. A. Statistical Method from the Viewpoint of Quality Control, 1939.

Stidham, S., Jr. L = λ W: A discounted analogue and a new proof. Operations Research. 20(6) 1115–1126, 1972.

Stidham, S., Jr. A last word on L= λ W. Operations Research. 22(2) 417–421, 1974.

Vacanti, Daniel S. "Actionable Agile Metrics for Predictability". ActionableAgile Press. 2014.

Vacanti, Daniel S. When Will It Be Done? ActionableAgile Press, 2017.

Vacanti, Daniel S. and Bennet Vallet. "Actionable Metrics at Siemens Health Services". AgileAlliance.com. 1 Aug 2014.

Vallet, Bennet. "Kanban at Scale: A Siemens Success Story." Infoq.com. 28 Feb 2014.

Vega, Frank. "Are You Just an Average CFD User?" Vissinc.com. 21 Feb 2014.

Vega, Frank. "The Basics of Reading Cumulative Flow Diagrams". Vissinc.com. 29 Sep 2011.

Wheeler, Donald J., and David S. Chambers. Understanding Statistical Process Control. 2nd Ed. SPC Press, 1992.

Wikipedia "Monte Carlo method." Wikipedia.com 01 Aug 2014.