ActionableAgile® Press

ACTIONABLE AGILE METRICS FOR PREDICTABILITY

10th Anniversary Edition

With both new and revised content

Daniel S. Vacanti

Foreword by Colleen Johnson

Actionable Agile Metrics For Predictability: Tenth Anniversary Edition

Daniel S. Vacanti

This book is available at https://leanpub.com/aamfp-10th

This version was published on 2025-04-02 ISBN 979-8-9867724-8-6

For Coco. And for Skye, Sicily, and Manhattan always.

Also By Daniel S. Vacanti

Actionable Agile Metrics Volume II
Flow Metrics for Scrum Teams
The Kanban Pocket Guide
When Will It Be Done?

Contents

Foreword

I don't know if I would have ever believed you if you told me that I'd carry around a book about metrics full of notes, dog eared pages and post it notes. Long before Daniel changed the trajectory of my career with ProKanban, he did it with the first edition of this book. Like many of us who entered into the tech field as scrum masters, I floundered to figure out if I was really adding value to the team or to their work. I most often felt more like the team mom, reminding everyone to eat their vegetables and pick up their socks (and to get it all done before the sprint demo). Even though the industry—then flush with cash—was a fun place to be, I wasn't sure it was the right place for me. I toyed with starting a kids after school program and collected piles of masters degree pamphlets though I wasn't entirely sure what I wanted to pursue. Then I read AAMFP.

It unlocked everything that I wanted to be able to show the teams about how they were working with concrete data. I was able to have a daily standup that focused on organizing around the work that was stuck or delayed without half the team tuning out. I was able to have a sprint planning session that more accurately accounted for how much work we could get done without resorting to a spreadsheet of available dev hours. It gave me the tools to effectively steer retrospectives towards measurable improvements that really had an impact on how the team was working without another round of "what went well/not so well.". And it gave me ways to have a conversation about actions the team needed to take next without feeling like their very overpaid babysitter.

Don't let the maths scare you. Everything in this book is data that is easily accessible, easily digestible, and easily actionable. You likely have all of the data you will need to get started without making any changes to your process or tooling- and you won't need a Masters degree in math or statistics. Daniel's ability to weave in stories (and a surprising number of sports analogies for a man who is always in a blazer) will help you make sense of each metric outside of a software context and make it easy to recall how best to use the data in front of you.

Daniel's book came into my life and my career when I was looking for a way to provide more value to my team but even more so, more value to myself and my career…ten years later it continues to do that. Every tool, practice, and metric used in the book is as important now as it was back then. As we watch more organizations slash spending, move towards AI, and attempt to do more with less, the things Daniel offers in this book are the ways we can realistically do that without complete and utter disrespect for the brilliant people doing the work.

I hope this book changes your career next. Whether that is how you manage a dev team, how you set project due dates,or how you communicate risk to your stakeholders, I know that you will finish this book with a new perspective on how to do what you do more effectively and with the data you need to prove that its working. I believe ten years from now we will keep coming back to AAMFP, digitally dogeared and full of notes, to remind us how a few simple metrics can transform the way we work and help us all deliver value more efficiently.

Colleen Johnson
March 2025

Preface

What were you doing 10 years ago? I, myself, had just met Prateek Singh. But that's a story for another time.

Think about how much the world has changed in the past 10 years. Thankfully, flow principles have not. So why a new edition to this book?

10th Anniversary Edition

While it is true that flow principles themselves have not changed, my understanding of them has. Especially my understanding of how to teach them. If I am being completely honest, my motivation for writing the first edition of this book was anger. At the time I was angry that:

1. No one was talking about flow metrics, and,
2. Instead of flow metrics, the Kanban community was only talking about Cumulative Flow Diagrams (CFDs)—and they were even talking about CFDs incorrectly.

That is why the first book was so CFD heavy. Don't get me wrong, a CFD is a powerful chart, but its power lies in teaching and not in action. By the time you get your CFD to a point where it is usable (by following the advice in this book), then it will be extremely boring. Seriously. It simply won't tell you anything useful—certainly not anything useful that you couldn't more easily get off of another chart.

Which brings me to my next point. When I was first starting with this stuff, Work In Progress was my hammer. And I used to love to beat teams over the head with it. The problem is that if you approach a team that has never limited WIP (and I mean never because most Agile teams never have) then it is damn near impossible to get them to understand why they should. They have been taught—and therefore adhere to—two false maxims:

1. The sooner you start something, the sooner you get it done.

2. The more you work on, the more you can get done.

As you will see in the coming text, neither of these statements is true. In fact, the exact opposite is much more likely to be true.

My Work In Progress fixation caused me to completely miss the most important metric for getting teams to adopt flow: Work Item Age. Technically, Age was there the whole time buried deep inside of Little's Law's assumptions. But I certainly didn't make it as explicit as I should have. This version fixes that. Age is now front and center as it should be. If you finish reading this book and do not understand the importance of Age when it comes to managing flow for predictability, then I have completely and utterly failed.

Some other changes to this edition that you might note:

- I've removed the chapter on forecasting. Without realizing it, in that chapter, I had started writing my second book, "When Will It Be Done?". If you want to learn about forecasting, read that book (you can find the full reference in the bibliography). Further, the whole second half of my third book (and second book in this series): "Actionable Agile Metrics Volume II" is on forecasting (also listed in the bibliography). As those two texts are much better treatments, I saw no reason to include that discussion here.
- I've moved the chapter on Histograms to an appendix (Appendix C). Histograms—and the dangers of them—have been given a much richer explanation in "Actionable Agile Metrics Volume II". While Appendix C is a decent start, it doesn't do a great job of explaining how treacherous Histograms can be. Still it may be useful to have a cursory understanding of those charts for some parts of this text.
- I've made small tweaks to the "started" and "finished" language throughout the text so that it is more in line with the current Kanban Guide. No, this is not a book about Kanban, but the Kanban Guide is currently the simplest, and therefore best, reference on how to get started with Flow.
- I've removed the Siemens HS case study altogether. This is not meant to minimize the learnings of that story. However, that case study has now been published in several different places, and no doubt most readers of this book have come across it in

one form or another previously (it is still included in "When Will It Be Done?"). If you are not familiar with that writeup, you can still find that story in its full, originally published form here: https://www.infoq.com/articles/kanban-siemens-health-services/.

- For the print edition only: the most frequent complaint that I got about the first book was that the images were very hard to see in print. Therefore, I've completely redone every single image in the text and now offer a full colour version of the print edition. I think this makes a big difference to readability and should address all of those quality issues.

The above changes notwithstanding, the importance of the messaging in the original Preface to the first edition remains just as relevant now as it was 10 years ago. As such, it follows in largely unedited condition.

Preface To All Editions

Your process is unpredictable. What you may not realize, though, is that you are the one responsible for making it that way. But that is not necessarily your fault. You have been taught to collect the wrong metrics, implement the wrong policies, and make the wrong decisions. Together, we can do better.

Until now you've probably assumed that the reason your process is unpredictable is due to circumstances completely outside of your control. However, you have much more control over the way you work than you think you do. Whether explicit or not, you have put policies in place that specifically prevent you from being predictable. Amongst other things, you start new work at a faster rate than you finish old work, you work on too many items at the same time, you ignore systemic dependencies and impediments, and you expedite requests that do not need to be expedited. You, in effect, initiate a denial of service attack on yourself and then wonder why it takes so long for things to get things done.

But all of those policies are under your control.

If we, as knowledge workers, want to get to a predictable world, we must first start by controlling the policies we can control. Taking

this control will seem uncomfortable at first. It will mean saying no to customer requests to start new work immediately. It will mean placing much less emphasis on upfront estimation and planning. It will mean looking at a different set of metrics than the ones you have been trained to track. Those metrics will tell you how predictable you are and what actions to take to improve. If you choose to collect the metrics suggested by this book, you will see that the data provided by them will immediately reflect the policies you have in place. That data will in turn suggest the changes to your policies necessary to be more predictable. Those policy changes will themselves be reflected in the new data you collect after the change. And so on and so on.

Your process is unpredictable. You know it. Your customers know it. Now it is time to do something about it.

Why Write this Book?

Because our customers demand predictability. Because you need someone on your side who has been asked tough questions and has found a way to give meaningful answers. Because most organizations that I visit are either uninformed or have been misinformed about what metrics and analytics they need to track to be predictable.

To get you where you need to be, however, I am going to ask you provocative questions. I am going to challenge your assumptions about what true Agility is. I may make you uncomfortable with some of the conclusions that I draw. I hope you will forgive me for all of these as my only intention is to make your process better. After all, as I just said, I am on your side.

Who Should Read this Book

Anyone who has ever been asked to give an estimate should read this book. Likewise, anyone who has ever asked for an estimate should read this book.

Analysts, developers, and testers need to know how to stop giving estimates and how to start making accurate predictions.

Product owners, project managers, and executives need to know

what makes for a meaningful prediction and how to hold teams accountable to make those predictions.

Conventions Used

All metrics and analytics will be capitalized. For example: Work In Progress, Cycle Time, Cumulative Flow Diagram, Scatterplot, etc.

I am also going to capitalize all methodology names. For example: Agile, Scrum, Kanban, etc.

ActionableAgile.com

Finally, and unless otherwise noted, all of the images of the analytics charts and graphs that are presented in this book were built using the ActionableAgile® Analytics tool. This tool is one that my company developed and can be found at:

https://www.actionableagile.com

In addition to the tool, accompanying blog posts, book updates and errata, videos, etc. can also be found on the ActionableAgile® website.

PART I: Foundations of Predictability

Chapter 1 - The Most Important Part of Predictability

Once upon a time (1960-ish), in a faraway land (Cleveland), there lived a young professor who was tasked with teaching a class on queuing theory. One day he reached the part of the class where he had covered most of the specific types of queues required by the course curriculum. It was at this point that he off-handedly remarked to his students that an oft-reappearing formula, $L = \lambda W$, seemed to apply to every queue that they had just studied. Even more remarkably, while this generality was widely believed to be true, it had never been definitively proven. Several hand-wavy heuristic proofs were available, but nothing that would stand up to serious mathematical scrutiny.

Our hero recounts what happened next: "After class, I was talking to a number of students [about the relationship $L = \lambda W$] and one of them (Sid Hess) asked, 'How hard would it be to prove it in general?' On the spur of the moment, I obligingly said, 'I guess it shouldn't be too hard.' Famous last words. Sid replied, 'Then you should do it!' "[1]

The name of the professor of course is Dr. John Little, and the rest is, as they say, history.

Background

Not to get too math-y this early in the book, but for our story to have a happy ending, I need to take a step back and provide some background.

The equation that Dr. Little first proved in 1961 was[2]:

Equation (1): L = λ * W

Where:

1. **L** = the average number of items in the queuing system.
2. **λ** = the average number of items arriving per unit time.
3. **W** = the average wait time in the system for an item.

Soon after its publication, Equation (1) forever came to be known as Little's Law (LL).

> The problem with starting a book off with an equation—especially when there are engineers in the audience—is everyone wants to run off and start to plug in numbers. However, LL does not work that way. LL will not "predict" how your process will behave in the future. So you know, I'll only scratch the surface of the tip of the iceberg of Little's Law here. Appendix A provides a much deeper dive into LL. I highly recommend reading Appendix A if you want to understand why LL forms the basis for much of the discussion around predictability. For now, however, I'll just distill LL down to its most important part. Thanks for tuning in and back to our regularly scheduled programming…

LL—as I just stated—applies regardless of the type of queuing system under investigation. In case you are wondering, yes, your Agile process is a queuing system (more on this in the next chapter). So yes, LL is applicable to your Agile process—regardless of what that actual process is.

Little's Law is applicable to your Agile process—regardless of what type of "Agile" you are doing.

But there's a catch. There's always a catch. With LL, you can't immediately assume that Equation (1) will "predict" how your process will behave. That is because Equation (1) requires a certain set of assumptions for it to work and chances are your process does not meet all of these required assumptions. All mathematical formulas work this way. Pick any well-known mathematical theorem that you may be familiar with and there will always be assumptions baked in. Newton's third law of motion, and Einstein's special relativity both

require certain assumptions to be met before they can be applied. I'm reminded here of a joke that is told on one episode of the TV show The Big Bang Theory[3]:

"There's this farmer who has a chicken that won't lay any eggs. So he calls a physicist to help. The physicist does some calculations and tells the farmer, 'I have a solution, but it only works on spherical chickens in a vacuum.' "

Trust me, if you are a physicist, you are rolling in the aisles right now.

Not to belabour the point, but Equation (1)'s spherical chickens require:

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).

Don't worry if you do not know what "stochastic" or "stationary" means. You don't need to (for this book anyway). All you need to know here is that for Equation (1) to hold there is a set of assumptions that need to be fulfilled and those assumptions are stated above*.

The Moral of the Story

If you are not asleep yet, I'll try to get to the point now. Most probably if you have come across Little's Law in the past, you did not see it as was defined in Equation (1). Rather, you probably first saw it as:

Equation (2): WIP = CT * TH

Where:

1. **WIP** = the average total inventory in the system.

2. **CT** = the average amount of time it takes for an item to flow through the system.
3. **TH** = the average Throughput of the system.

[Note: Exact definitions for all of these terms will be given in Chapter 3.]

Confusingly, Equation (2) is also called Little's Law. Maybe that doesn't confuse you because, at first glance, Equation (1) and Equation (2) might appear the same save for the names being changed. If you look closely, however, you'll notice the subtle difference that Equation (1) is stated in terms of a system's input (i.e., Arrival Rate) while Equation (2) is stated in terms of a system's output (i.e., Throughput). Though indeed subtle, this change makes all the difference in the world. That's because—you guessed it—a different equation requires a different set of assumptions.

The assumptions required to make Equation (2) valid are[4]:

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.

As you can see, that simple change of perspective from the arrival rate of (1) to the departure rate of (2) requires a whole host of new assumptions. Those are some seriously vacuum-sealed, spherical chickens.

To explain why this is important, let's pause for a second and try an experiment. Take a look at the process that you are operating right now. Hopefully, you have been tracking (or have access to) the metrics as have been defined in Equation (2) (setting aside for a moment, that, as acknowledged earlier, I haven't *exactly* defined what those metrics are). If you do in fact know the average WIP, CT, and TH of your process (for the last month, for example), I'd like you to plug those numbers into the

Little's Law (LL) equation now. Try several different permutations of the equation. 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? LL is supposed to be an equation. As in equal. But the calculations you just performed for your process are not equal. Well, the LL formula is indeed exact, but it is only exact in contexts when that specific set of assumptions is fulfilled (see above).

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 an LL perspective) is so important because an unstable process is by definition unpredictable. Your experience tells you this. How easy is it to forecast a process where the number of things you are working on increases every day? How easy is it to forecast a process where all the things you work on get blocked by dependencies on other teams? How easy is it to forecast a process where you are constantly interrupted by "one-offs" that show up?

The upshot is this: if what you care about is predictability—which is hopefully why you are reading this book—then LL's assumptions act as a powerful guide to policies that we should implement to help prevent our process from destabilizing.

LL's assumptions act as a powerful guide to policies that we should implement to help prevent our process from destabilizing.

And of those five, when it comes to predictability, there is one assumption that rules them all.

All Assumptions Are Equal. But Some Assumptions Are More Equal Than Others.

As we'll see throughout the rest of this book, a thorough understanding of what it means to violate each of LL's assumptions is key to understanding predictability. However, there is one assumption in particular that is the most important. A quick examination of each will be helpful to illustrate this point.

The first thing to observe about Equation (2)'s assumptions is that #1 and #3 are logically equivalent. Honestly, 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. More importantly, however, you'll notice what Little is **not** saying 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 nor of more WIP being worse. Little couldn't care less. All he cares about is that WIP itself 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 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 an unpredictable workflow.

Assumption #5 is included with the others simply so that the math works out. 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. Again, if anyone on your team struggles with this concept then you have bigger problems than how to best apply LL. Assumption #5 is trivial and, therefore not the most important.

This leaves us with assumption #4. Allowing items to age arbitrarily is the single greatest factor as to why you are not predictable at delivering customer value. Stated a different way,

The single most important aspect that you should be paying attention to in order to drive predictability in your process is to not let work items age unnecessarily.

Predictability

If you have been keeping score then you know I've used the word "predictability" about a half-dozen times by now, but I've never really defined it. Well here goes.

"When will it be done?"

That is the first question your customers ask you once you start work for them. And, for the most part, it is the only thing they are interested in until you deliver. Whether your process is predictable or not is judged by the accuracy of your answer.

Think about that dynamic for a second. By answering "When will it be done?" you are setting an expectation with your customers. In your customers' eyes, if your process meets that expectation, then you are predictable. If it doesn't, then you are not. It is as simple as that.

Predictable Process: A process that behaves the way it is expected to.

This is the reason I've been talking so much about Little's Law. I said before, I don't care about the equation. Throw that equation out of the window. What I care about are the things that make that equation work: the assumptions. That is, how often does your current process violate one or more of the assumptions of LL?

For example, in your current situation:

- Are you constantly asked to start new work before you have had a chance to finish old work—for example, one-off requests, business-as-usual work, keep-the-lights-on items, etc. (Assumption #1)?
- Are you constantly asked to expedite new requests in addition to being expected to get all of your other current work done (Assumption #1)?
- How many features do you start but do not finish because they get canceled while you are working on them? How likely is it that the new items that replace the canceled work will themselves get canceled (Assumption #2)?
- When something that you are working on gets blocked (for whatever reason), do you simply put that blocked work aside and start to work on something new (Assumption #2 and Assumption #4)?
- Do you ignore the order in which you work on items currently in progress because, for example, you think, "It all has to get done anyway, so who cares what order we work on this stuff?" (Assumption #4)
- Do you constantly add new scope or acceptance criteria to items in progress because of phrases like "While we are at it…" or "Since we are in there already, it would be easy to…". When new work is discovered, is it just easier to modify an existing item in progress rather than create a new one (Assumption #4—potentially Assumption #2)?
- Do you constantly add scope to items that have started because "It's not valuable unless we do all of these things" (Assumption #4—potentially Assumption #2)
- When an item is taking too long to complete, have you ever said or heard someone say "It is just bigger than we thought it was" and/or "It will get done when it gets done"? (Assumption #4)
- When things take too long to complete, is management's first response always to have the team work overtime? Or to cancel vacations/holidays? Or try to add more people to the team who have no idea what they are doing? (possibly all assumptions!)

I think you get the picture. The problem with the above is it may feel like you are doing the right things. You might be thinking, "The

sooner we start stuff, the sooner it gets done." Or, "The more we work on, the more we'll get done." Or, "It doesn't really matter what else we are currently working on because we must start this new thing right now." Or, "It absolutely is easier just to add scope to this item than start a new one." No doubt all of these actions are well-intentioned as you truly do believe that you are adding/delivering more value.

The problem is, that the reverse is much more likely to be true. Each time you violate an assumption of Little's Law you jeopardize your ability to deliver value and you jeopardize your ability to deliver that value *predictably*. I'm not saying that in your context you are not delivering any value at all. But I am saying that you are probably not delivering as much or as fast as your customers want.

If violating LL's assumptions is a force of evil, this book is a force for good. In the coming text, I'll talk about how to identify these problems and fix your process without the need for divine intervention. In this first section, we'll focus on the most important assumption (Aging). However, by the time we are through, we will have covered all assumptions. More importantly, by the end of this book you will understand how to refactor your whole way of working for optimal predictability.

Your process is no fairy tale—you have all the control you need to get the ending you want. Take that control.

Conclusion

More than limiting WIP, more than estimation, more than fixed-length timeboxes, more than prioritization, the only question to ask of your Agile (flow) system is are you letting items age needlessly?

But I fear I've gotten way too ahead of myself in an attempt not to bury the lede. Before we can really talk about Aging, we need to talk about SLEs. And before we can talk about SLEs we need to talk about Cycle Time Scatterplots. And before Scatterplots, we need Cycle Time (CT). And before CT, we need flow data, start/finish points, and flow modeling. A journey of a thousand miles starts with the first step, but we'll start ours with those last two.

Key Learnings and Takeaways

- Little's Law relates the basic metrics of flow in an elegant, fundamental equation.
- Little's Law is a relationship of averages.
- Do not get distracted by the math of Little's Law—the significance of the law does not necessarily come from plugging numbers into the equation—the importance of LL is understanding its assumptions.
- When stating LL in terms of Equation (2), for contexts with continuous WIP, there are five assumptions necessary for Little's Law to work, they are:

    1. The average input or Arrival Rate (λ) should equal the average Throughput (Departure Rate).
    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.

- Although not explicitly stated as part of the formula itself, Work Item Age is the most important metric for predictability.

*It should also be quickly noted what you don't need to know for LL to work. You'll observe that the assumptions say nothing about size/complexity (that's right, you don't need to estimate in order to apply LL), they say nothing about queuing discipline (e.g., first-in-first-out, last-in-first-out, etc.), they say nothing about the distribution of arrival times, or the number of servers, or the number of queues feeding the servers, etc. So, yes, you can still be predictable if your work items are all of different size. Yes, you can still be predictable if sometimes you have one person working on an item and sometimes you have many people working on an item. And, yes, you can still be predictable if you have multiple, unordered backlogs feeding your process. But more on all of that in the coming chapters.

Chapter 2 - Setting Up For Predictability

The whole reason for your team's existence is to deliver value to your customer(s). Value, however, doesn't just magically appear. Constant work must be done to turn potential product improvement ideas into tangible customer value. The steps needed to turn an idea into something concrete that our customers find valuable is called a process.

"Agile" is not a process. "Scrum" is not a process[1]. "Kanban" is not a process[2]. You may have used these principles and techniques to build a process, but they, in and of themselves are not a process. Whether you know it or not, you and your team have built a value delivery process that goes way beyond anything Agile. That process may be explicit or implicit, but it exists.

Why should you care about what your process actually is? Because an understanding of your process is fundamental to an understanding of something that we will call flow:

Flow: the movement of potential value through a given process.

Maybe you've heard of the other name for process, known as workflow. There is a reason it is called workFLOW. Because for any process what really matters is flow.

[**Note**: I will often use the words "process", "workflow", and "system" interchangeably. I will try my best to indicate a difference between these when a difference is warranted. For most of the contexts in this book, however, any difference among these words is negligible so that they can easily be used synonymously.]

The reason you should care about flow is because your ability to be predictable will ultimately come down to your ability to actively manage flow.

Your ability to be predictable will ultimately come down to your ability to actively manage flow.

Since flow is all about the movement of value, it stands to reason that our whole way of working should be oriented around optimizing flow. By this line of reasoning, if you feel your current process is unpredictable, the first thing to investigate is poor flow.

The problem is, that investigating poor flow is not as easy as it sounds. Most Agile dogma is centered around the antithesis of flow (fixed-length timeboxes, complexity estimation, upfront planning—to name a few). As such, very few of us really understand what it means to achieve good flow, much less measure it.

Luckily for you, I'm here to help.

Modeling Flow

I believe that all Agile processes can be modeled as a simple queuing system as shown below[3]:

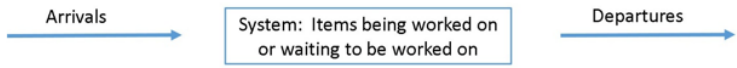

Figure 2.1: A Simple Queuing System

Dr. Little himself provides the best explanation of what is going on in Figure 2.1: "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, 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 the number in queue and total number in queue plus service, the latter being called number in system."[4]

The point of Figure 2.1 is to establish the need for clearly set system boundaries before any discussion of predictability can take place. To illustrate this point, consider that you are traveling for your next holiday or vacation. At what point would you consider your holiday to have started? When you leave your front door? When your train departs? When you get to your hotel? Or maybe you get into holiday mode when you first booked your tickets?

What about when you consider your holiday to have ended? When you check out of your hotel? When you physically return home? Or when you purchase your tickets for your next trip?

You can see how your answer to "How long?" is extremely sensitive to what you choose as your start point and what you choose as your endpoint (what happens between those start and end points is important too, but you'll have to hang on a bit for that discussion).

The same is true for our Agile processes. Before any conversation about predictability can take place, we first need to agree on what it means for an item to have arrived to our system (started) and what it means for an item to have departed our system (finished).

Defining Arrivals and Departures

Let's consider arrivals first. For this, we need to establish an explicit and obvious entry point such that once work has crossed that point, that work is considered started. If you are visualizing your workflow, then that start point can be any step of your choosing—but the choice must be made. To illustrate, for many teams this entry point usually takes the form of a Work In Progress (WIP)-limited column on the front of your process (we'll talk about WIP in the next chapter), and you will often see this column labeled as "Input" or "Ready" or "To Do" or the like (more on how to set the WIP limit on this column below). An example of what this column might look like is shown in Figure 2.1:

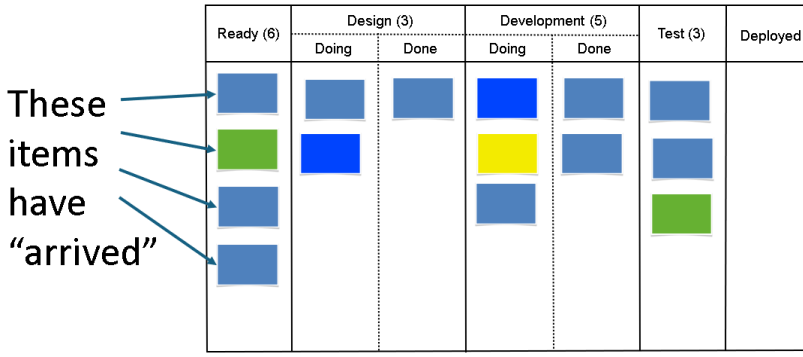

Figure 2.1: Arrivals into a Kanban System

In Figure 2.1, items that have been placed in the “Ready” column are said to have arrived to the process. This column represents a clear, unambiguous signal to the world that the team has accepted work.

Please note that in this example, the arrivals column is very different from a more traditional backlog. It is not meant to be an ever-expanding repository for all candidate customer requests. The WIP Limit on this column represents the real-time capacity of the system to take on new work and serves to force us to only pull in new work in a just-in-time manner. Again, more on that a little later.

Similarly, we are going to need to establish a clear, unambiguous departure point for our system. Items that pass this point do not necessarily have to be visualized as part of your agile process—though many teams do choose to do so; however—whether visualized or not—the departure point must be chosen. Typically speaking, if a team chooses to represent the departures column on their board, then it will not have a WIP Limit on it. Regardless of the visualization employed, it is important to define the exact point of the system where work departs, (hopefully) never to return. For example, this could be the point where we deploy code to production or the point at which we hand an item off to a downstream team (see Figure 2.2).

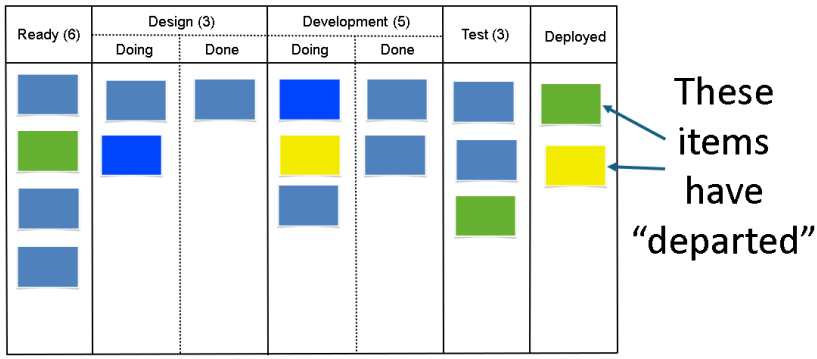

Figure 2.2: Items that have departed from the system

In the Figure 2.2 example, the demarcation line between "in our process" and "not in our process" is the line that separates the "Test" column from the "Deployed" column. More importantly, the expectation here is that the team has put in place a set of policies for what it means for items to move from Test to Deployed, and that once those items are in Deployed, they no longer count as being part of our process.

Why Start and Finish Matter

As in the holiday analogy above, you can see how choosing a different start point for Figure 2.2 could potentially radically change how predictable our system is. For example what if we chose Development Doing as our start point? Or Design Doing? Or some point earlier in our process such as "Discovery"?

The same is true of departures. What if we chose Development Done as our finish point? Or Test? Or some point later in our process such as "Customer Validation"?

The task of choosing a start and end point might seem trivial, but I assure you it is not. If you were to ask your team members to point to the unambiguous point at which work is started in your current context, what would they say? What about what they think about when work is finished? If you have ten people on your team, would you get 10 different answers?

You might think that if you are employing a well-known Agile

framework, such as Scrum, these considerations are already taken care of for you. Not so fast. When, actually, is work considered "started" in Scrum? Is it when a Product Backlog Item (PBI) is selected for a Sprint? Is it when work on the PBI actually begins during the Sprint? Is it when refinement starts on the PBI even before it is selected for a Sprint? Is it when the PBI is first placed on the Product backlog? Believe or or not, all of these are legitimate choices for a start point depending on context and depending on what you want to measure. But you have to make the choice.

Likewise, what does it mean for a PBI to be finished in Scrum? Is it when it meets the Definition of Done (DoD)? That might be your obvious choice, but is that what happens in practice? Maybe your team considers all PBIs finished when the Sprint ends—whether the DoD has been met or not? Maybe PBIs finish at some point *after* the DoD has been met (e.g., maybe your DoD does not include deployment to production, but you don't count things as finished until they are deployed).

Long story longer, your first step toward predictability is to sit down and decide what you want the boundaries of your agile process to be. And there are at least two pieces of good news here:

1. If you are not sure what start point and end points are, then feel free to choose several. Going back to the Scrum example, there is nothing wrong with saying we want to look at when a PBI starts refinement AND when that same PBI starts development in the Sprint. To clarify, you can have as many start points and as many finish points as you want in your process—but you have to have at least one of each (and my recommendation is usually just to start with one of each).
2. Those start and end points can change as your system matures. There's nothing to say that your boundaries have to be set in stone. Quite the opposite. As you learn about your context, you will almost want to change what you are measuring. And that's generally a good sign as that means you are improving. Usually over time, the boundaries of your process will widen to the point that they cover your whole end-to-end workflow. Don Reinertsen is famous for saying that your process begins when the first dollar is spent[5]. Likewise, (I'm assuming) your process ends when the

last dollar is spent. Granted, this is a fairly radical and idealized view, but one that I believe that all teams should aspire to.

Push Vs. Pull In a pull-based system, an entry (or boundary) point is fairly easy to define. That is because, in a pull system, a team only starts work when it has capacity to do so. Thus, a work item can only count as started if it has been voluntarily pulled into the process by the individual, team, or organization responsible for operating that process. The "arrival point" of the system, therefore, could easily be defined as the point at which the team performs its first pull transaction on the work.

For push-based systems, an entry point is much harder to define. That is because there is no consideration for a team's capacity when deciding when work should be started. In a push system work can be considered started when any stakeholder has a reasonable expectation that work has been committed to (whether the team responsible for performing the work knows about that work or agrees to it or not). This expectation could be set for such arbitrary reasons as the work has been requested, the project has been funded, or some manager somewhere thinks it is a good idea to start—regardless of whether there is any capacity to do so.

Obviously I have a bias for pull systems over push systems, but the concept of started and finished applies regardless of push vs. pull. If you find yourself operating within a push system, then the best, first predictability exercise you might want to undertake is to define the boundaries of your process. Getting a handle on that is a necessary (but unfortunately not sufficient) step down the road to predictability. Just know that at some point, optimal predictability may require you to at some point to move to a pull-based system.

Conclusion

Modeling flow isn't difficult. Just start by deciding the boundaries of your process in the form of a well-defined start point and a well-defined finish point for all value items that are flowing through your process.

Having modeled flow, measuring it will simply be a matter of taking a timestamp of when a potential value item crosses your newly defined start and finish points. That will be all the data you need to get started.

So now that you've got your flow modeled, and you've started collecting data, how do you turn that data into meaningful insights? I'm glad you asked as that is what we will cover next.

Key Learnings and Takeaways

- All Agile processes can be modeled as a simple queuing system.
- Any predictable process needs a clear, unambiguous point at which it considers items to have "arrived" (started).
- Any predictable process needs a clear, unambiguous point at which it considers items to have "departed" (finished).
- You can choose several started points and several finish points for your process, but you must have at least one of each.
- The started and finished points can change over time.
- Once started and finished points have been defined, then the minimum data to collect is a timestamp for when an item starts and a timestamp for when an item finishes.

Chapter 3 - The Basic Metrics of Flow

Understanding flow and managing it requires a different paradigm than that espoused by traditional processes and frameworks. The answers to the essential questions of predictable process execution are not found in project plans, resource utilization charts, or team members' estimates. The answers will come from the monitoring, measurement, and management of a specific set of metrics. This chapter is all about defining the first three of these metrics: Work In Progress (WIP), Cycle Time, Throughput, and Work Item Age.

We've already spent a bit of time on Age, so let's talk about the others first. The good news is that these other three flow metrics are exactly the ones we need to track in order to answer the questions that our customers are asking. The customer question "How long to complete?" is best answered by the flow metric known as Cycle Time. The customer question "How many new features am I going to get in the next release?" is a question best answered by the flow metric known as Throughput. The last of the three, Work In Progress (WIP), does not directly answer any particular customer question, but it is the metric that will most greatly influence the other two (as demonstrated by Little's Law). For that reason, I will start this discussion with it.

Work In Progress

Work In Progress is the best flow metric to begin with for two reasons. First, as we have seen, WIP is the best predictor of overall system performance. Second, the other metrics of flow will themselves both be defined in terms of WIP.

Even so, WIP is probably the hardest metric to define. That is because the definition of WIP is two-dimensional: it must cover both the notion of "work" and the notion of "in progress".

Let's look at the idea of *work* first. For this book, I regard any direct

or indirect discrete unit of customer value as a candidate for *work*. The generic term I will use for these candidate units of customer value is "work item". A work item might be a user story, an epic, a feature, or a project. It might be a requirement, use case, or enhancement. How you capture work as work items and how you name your work items is entirely up to you.

Secondly, to define *in progress* we must first consider the boundaries of your process. Luckily for us, we just did that in the previous chapter. So if you skipped that part, you may want to revisit that now before proceeding.

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

If defining WIP is the hard part, then measuring it is the easy part. 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.

I will spend more time on this topic a little bit later, so I am going to ask you to just suspend disbelief here and accept that when it comes to WIP and predictability, there is no requirement to have all of your work items be of the same size or complexity. There is not going to be a need for any further complexity to be added to the calculation such as estimating your WIP 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. Trust me when I say you need to abandon that type of thinking if you truly want to build predictable processes.

For those of you who do not want to wait, an explanation of why size does not matter (said the actress to the bishop) is outlined in

Appendix A (the deeper dive into LL). For now, all you need to know is that WIP is calculated by counting individual work items.

Nor is there any restriction on the level at which you track work items. You can track WIP at the portfolio, project, feature, epic, or story level—just to name a few. All of these types of decisions will be completely up to you.

For you Kanban practitioners out there, you will also want to note 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 it does not. This result should be obvious as most Kanban boards do not always have columns that are at their full WIP limit. A more common situation is to have a Kanban board with WIP limit violations in multiple columns. In either of those cases simply adding up WIP limits will not give you an accurate WIP calculation. Even in a Kanban world, you still have to actively track the total number of work items in your process.

An implication of all of this is that most often items located in a backlog do not meet the definition of being included in a WIP calculation. There is a subtlety here that is going to require further discussion as it refers to the "point of commitment" that I mentioned a little earlier (for this deeper discussion, please see Chapter 8). Just know that—for the most part—when I talk about WIP, I do not include backlog items in that discussion.

As an interesting aside, you should know that you will have the option to segment and report on your WIP as you see fit. In some contexts, it may be beneficial to lump all of your WIP together and examine it from a holistic system's view. Or it may be beneficial to segment that WIP into types or categories and examine each one of those subgroups on its own.

For example, let's say your team performs work for the sales department, the marketing department, and the finance department. Let's also say that your team is responsible for the maintenance of a variety of existing applications. When looking at WIP you may want to combine all of those requests into one big group. Or your team may choose to just look at the part of your WIP that pertains to sales. Or your team may choose to look at the part of your WIP that pertains to marketing. Or you may just want to look at how your maintenance

items are doing. From a metrics perspective, performing that type of segmentation is not only going to be perfectly okay, but also, as mentioned earlier, in some instances is going to be desirable. If your team does segment WIP into different categories, then it is also going to be valid to talk about the Cycle Time and Throughput of those different types of segments. Segmenting (or filtering) WIP into different types may also be important from a reporting and analytics perspective which is why I will revisit this topic in the flow analytics chapters to come.

To sum up, if you are not currently tracking WIP, then you are going to want to start. Sooner is better than later.

Cycle Time

As I mentioned at the outset, the first question our customers ask when we start work for them is "When will it be done?" Answering that question will require us to measure the flow metric of Cycle Time. Measuring Cycle Time becomes much easier now that you have a basic understanding of 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[1] and, I 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: that 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.

Not only does defining Cycle Time in terms of Work In Progress make it more concrete and easier for people to understand, but it also

brings some needed consistency when talking about Cycle Time with respect to Little's Law and with respect to how Cycle Time is (or is not!) visualized on a Cumulative Flow Diagram.

Lastly, notice 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). I happen to think this guidance is wrong. I 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 would expect 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. I am 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 predictability 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 the most substantial predictability improvements. Remember, delay is the enemy of flow!

Lead Time vs. Cycle Time

If you have been exposed to Lean or Kanban concepts before reading this book, then what I have just defined as Cycle Time may sound a lot like what you have come to recognize as Lead Time. I understand that most people in the Kanban community prefer the term Lead Time to Cycle Time, but I am not one of them. My intention here is not to dive headlong into an academic (and ultimately useless) debate about which nomenclature is better, but I feel that I should at least present my thoughts on why I have chosen the terms that I have. You may agree or disagree with

my reasoning, but I hope you understand my intention here is not to be provocative or antagonistic (yet). I am going to talk about nomenclature in general a little later, but these specific terms require some special attention.

So why choose the term Cycle Time over Lead Time? My first argument is that regardless of whether you are talking about Cycle Time or Lead Time, you still have to qualify the boundaries of your time calculation. That is to say, both terms are very dependent on one's perspective: one person's Lead Time is another person's Cycle Time, and vice versa. For example, the development team's Lead Time is just the Product Manager's Cycle Time through the development phase. Given that with both terms boundaries must be qualified, I see no clear advantage of the term Lead Time over the term Cycle Time. Further, defining Cycle Time in terms of when something is counted as WIP clears up a lot of this ambiguity. (And this is not even to mention that as you get out of software and manufacturing, Lead Time is defined very differently.)

Secondly, I do not buy the argument that we, the Lean-Agile community, should shy away from using the term Cycle Time because the manufacturing industry has already defined it in a different way that may or may not be in agreement with how we use the name. I do not subscribe to the thinking that the "Lean" we are talking about here is just manufacturing theory blindly applied to knowledge work. I fully reject this thesis. The fact that manufacturing has its own definition of Cycle Time should be neither influential nor consequential to how we in knowledge work choose to define the term.

Lastly, and, I must stress, most importantly, the authors that I quote most—Reinertsen[2] and Little—both favor the use of the term Cycle Time. If it is good enough for them, then it is good enough for me. By the way, Hopp and Spearman also sometimes refer to Cycle Time as "Flow Time"[3]. I would suggest that the term "Flow Time" might be a better way for us to communicate what we precisely mean by Cycle Time in our context anyway. Even so, for the rest of the book, I will use the more common term, Cycle Time, and I will use it in the way that I have defined it here.

Maybe you are already persuaded by these reasons to track Cycle

Time. But there are more.

The first supporting reason to track Cycle Time is that it can be a rather good predictor of cost. Very generally speaking, the longer something takes to complete the more it is going to cost. Project, feature, or even user story cost can be one of the biggest determiners of whether a company chooses to invest in development or not. Like it or not, we are going to need Cycle Time data to figure out development costs.

The final and most important reason to understand Cycle Time is that it 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.

Throughput

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 get done per day, per week, per iteration, etc. For example, you might state that the Throughput of your system is "three stories per day" (for a given day) or "five features per month" (for a given month).

A further thing to know about Throughput, however, is that this metric as I have defined it here is potentially very different from the traditional Agile metric of "Velocity". Velocity, as you may know, can be measured in terms of Story Points per sprint or iteration. However, for Throughput, I am talking about actual counts of work items (e.g., the

actual number of discrete Stories and *not* Story Points) per unit of time. As I have just mentioned, the unit of time you choose for Throughput is completely up to you. The implication being that your choice of a time period need not necessarily coincide with an iteration boundary. I say all of this because many agile coaches and consultants use the words "Velocity" and "Throughput" interchangeably. Just know that in most contexts these two terms are definitely *not synonymous*.

If Throughput is how fast items depart from a process, then Arrival Rate is how fast items arrive to a process. I 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, as I alluded to in the previous chapter, 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 (also see Chapter 11). However, my more immediate reason for 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.

Work Item Age

I have saved, as stated at the outset, the most important metric for last. Work Item Age is the amount of time that has elapsed from when an item crossed the well-defined start point of your process until the present time.

Work Item Age: The amount of time that has elapsed from when an item crossed the well-defined start point of your process until the present time.

Stated differently, Work Item Age is the amount of elapsed time that

a started but unfinished work item has spent as Work In Progress.

For example, let's say that an item started on January 1 and today it is January 6. Let's further say that the item still is not finished. The Age of that item is 6 days (as with Cycle Time, the Age calculation is inclusive—which means we count the first day that the item started as part of its Age). Note that Age only applies to items that have started but not finished. Once an item finishes then what we calculate is Cycle Time and not Age. Also, note the emphasis on elapsed time. As was the case with Cycle Time, the calculation for Age is based on the total time that has *elapsed* since the item started. There is no subtracting out nights, weekends, holidays, etc. Remember that we want our flow metrics to reflect how our customers see the world and Age is no exception.

Before we get into why you should care about Age, we need to revisit Cycle Time (CT). At the risk of repeating myself, most people think that the reason flow 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 flow 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 be 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 a longer age represents a 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.

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.

Maybe you've heard the Kanban saying "Stop Starting. Start Finishing." Now you know why they say that.

We can take this logic a step further and assert that **all** flow 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. We will revisit all of this in Chapter 7.

Conclusion

What I have shown here are just the basic metrics of flow to get you started: WIP, Cycle Time, Throughput, and Work Item Age. 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 goal is predictability, then these are the metrics that you are going to want to track.

I would also like to say one final word on vocabulary. No doubt if you have done any reading on this topic that you have come across different names for the concepts that I have defined in this chapter (I discussed the most contentious example of this in the "Lead Time vs. Cycle Time" section above). As I mentioned earlier, the point of this discussion is not to spark any religious wars over nomenclature. The point of this chapter is only to get you thinking about the basic concepts that are communicated by these metrics.

For example, for us to have a conversation about predictability, we are going to need some notion of the total amount of items in a system. I am choosing to call that notion Work In Progress. If you prefer the term Work in Process or something else, then, by all means, use that name. We are also going to need some notion of the amount of time that items spend in the system. I am choosing to call that Cycle Time. If you prefer Lead Time, Flow Time, Time In Process, or something else, then, please do not let me stand in your way. Lastly, we need some notion of the amount of items that leave the system per unit of time. I am choosing to call that Throughput. But please feel free to use the terms Completion Rate, Departure Rate, or anything else that may make you comfortable (so long as you do not use the term Velocity!).

Just know that it is the definitions of these concepts that are important—not the names. However, to be clear, the rest of this book will utilize the names and definitions of these metrics as I have outlined in this chapter.

Having spent an inordinate amount of time defining these metrics, it is now possible to get on the really important piece: how to calculate them.

Key Learnings and Takeaways

- Any work item can be counted as WIP when it is between the defined entry point of a process and the defined exit point of a process.
- The choice of what work items you count as WIP when between those two points is completely up to you.
- WIP can be segmented into several different types.
- If WIP is segmented into several types, then it is also valid to talk about the Cycle Time and Throughput of those type segments.
- Cycle Time and Throughput are always defined in terms of WIP.
- Cycle Time is the amount of elapsed time that an item spends as Work In Progress.
- Throughput is the amount of Work In Progress completed during some arbitrary interval of time.
- Work Item Age is the amount of elapsed time that an unfinished item has spent as work in progress.
- The real reason to limit WIP is to prevent an item from aging unnecessarily.
- All flow principles for predictability can be derived from aging.
- The names of metrics are not as important as their definitions. Use whatever names you want for these metrics, but make sure you define them as they are defined here.
- Track these metrics because they have predictive power, are inexpensive to gather, and answer the important questions that your customers are asking.

Chapter 3a - Flow Metrics Data and Calculation

Calculating flow metrics will first depend on collecting the right data[1]. Collecting the right data will depend on modeling your flow. From Chapter 2, we know that to model flow, we need start and finish points.

What Data To Collect

Once you have the start and finish points of your process defined, then the only data you need to collect to calculate flow metrics is a timestamp for when an item crosses your start point and a timestamp for when an item crosses your finish point. That's it. All of the metrics that I will discuss in this book need only those data points. In the interest of clarity, there may be other data you want and/or need to collect for your context, but for the basic metrics of flow, those two data points are all you need.

For example, in your context, your timestamp tracking might look like this:

Work Item ID	Arrived	Departed
1	01/01/2016	01/03/2016
2	02/02/2016	03/03/2016
3	01/02/2016	03/04/2016

Figure 3a.1 – Sample Timestamp Data

[It is impossible to tell from these data in Figure 2.4, but I'm using American-style dates. Not only are American dates represented in Figure 2.4, but I will use that style for the rest of the book (unless otherwise noted). It doesn't necessarily matter here, but it might matter later.]

We'll discuss what to do with these data in the coming chapters, but just feel secure in the knowledge that the data won't get much more complicated than this. Channeling my inner broken record, these data cannot be collected unless start and end points have been established.

If you are using a tool (like Jira) to help track work, then when it comes to data, there is again good and bad news. The good news is that most Agile tooling tracks timestamps as required above. The bad news is that those tools usually do not make this data easily accessible. You might be able to query a database or code against some API, but almost certainly there will be some nontrivial amount of work required to get the data you need. Thankfully, there are plugins out there to help. The one that I recommend is ActionableAgile® Analytics. But as I said before, I am biased.

Flow Metrics Calculation

The definitions from the previous chapter are meaningless if you don't know how to calculate each metric from the data you've collected. Now that we understand what data we need, let's explore how to properly calculate each metric:

WIP

WIP is the count of all work items that have a started timestamp but do not have a finished timestamp for a given interval of time. That last part is a bit difficult for people to grasp. Although technically WIP is an instantaneous metric—that is, at any time you could count all of the work items in your process to calculate WIP—it is usually more helpful to talk about WIP over some time unit: days, weeks, sprints, etc. Our strong recommendation—and this is going to be our strong recommendation for all of these metrics—is that you track WIP per day. Thus, if we wanted to know what our WIP was for a given day, we would just count all the work items that had started but not finished by that date. For Figure 2.1, our WIP on January 5th is 3 (work items 3, 4, and 5 have all started before January 5th but have not finished by that day).

Cycle Time

Cycle Time equals the finished date minus the started date plus one (CT = FD - SD + 1).

If you are wondering where the "+ 1" comes from in the calculation, it is because we count every day in which the item is worked as part of the total. For example, when a PBI starts and finishes on the same day, we would never say that it took zero time to complete. So we add one, effectively rounding the partial day up to a full day. What about items that don't start and finish on the same day? For example, let's say an item starts on January 1st and finishes on January 2nd. The above Cycle Time definition would give an answer of two days (2 – 1 + 1 = 2). We think this is a reasonable, realistic outcome. Again, from the customers' perspective, if we communicate a Cycle Time of one day, then they could have a realistic expectation that they will receive their item on the same day. If we tell them two days, they have a realistic expectation that they will receive their item on the next day, etc.

You might be concerned that the above Cycle Time calculation is biased toward measuring Cycle Time in terms of days. In reality, you can substitute whatever notion of "time" that is relevant for your context (that is why up until now we have kept saying track a "timestamp" and not a "date"). Maybe weeks are more relevant for your specific situation. Or hours. Or even Sprints. If you are on a Scrum team and you wanted to measure Cycle Time in terms of Sprints, then the calculation would just be Finished Sprint – Start Sprint + 1 (assuming work items cross Sprint boundaries in your context). The point here is that this calculation applies in all contexts. However, as with WIP, my very strong recommendation is to calculate Cycle Time in terms of days. The reasons are too numerous to get into here, so when starting out, calculate Cycle Time in terms of days and then experiment with other time units later should you feel you need them.

Work Item Age

Work Item Age equals the current date minus the started date plus one (Age = CD - SD + 1).

The "plus one" argument is the same as for Cycle Time above. You will never have a work item that has an Age of zero days. Again, start

by tracking Age in days. All the reasons why are the same as what I have outlined for CT above.

Throughput

Let's take a look at a different set of data to make our Throughput calculation example a bit clearer:

Work Item Id	Arrived	Departed
1	01/01/2022	03/01/2022
2	01/02/2022	03/03/2022
3	02/02/2022	03/03/2022
4	01/02/2022	03/04/2022
5	03/02/2022	03/04/2022

Figure 3a.2 - Sample Process Data

To calculate Throughput, begin by noting the earliest date that any item was completed, and the latest date that any item was completed. Then enumerate those dates. In this example, those dates in sequence are:

Completed Date
03/01/2022
03/02/2022
03/03/2022
03/04/2022

Figure 3a.3 - Consecutive Calendar Days Between First and Last Finished Items

Now for each enumerated date, simply count the number of items that finished on that exact date. For this data, those counts look like this:

Completed Date	Throughput
03/01/2022	1
03/02/2022	0
03/03/2022	2
03/04/2022	2

Figure 3a.4 - Calculated Throughput

From Figure 3.3 we can see that we had a Throughput of 1 item on 03/01/2016, 0 items the next day, 2 items the third day, and 2 items the last day. Note the Throughput of zero on 03/02/2016—nothing finished that day.

As stated above, you can choose whatever time units you want to calculate Throughput. I advise very strongly that you measure Throughput in terms of days. Again, it would be a book in itself to explain why, but let's consider two quick justifications: (1) using days will provide you much better flexibility and granularity when we start doing things like Monte Carlo simulation for planning activities (explained in detail in my second book "When Will It Be Done?"); and, (2) using consistent units across all of your metrics will save you a lot of headaches. So if you are tracking WIP, Cycle Time, and Age all in days, then you will make your life a whole lot simpler if you track Throughput in days too (not to mention we now know this is a requirement of LL).

You still with me? I hope so because we've saved the most difficult parts for last. These next couple of topics might seem out of place here, but, trust me, you must understand these concepts before we can go any further with metrics and data.

Randomness

From above, you now know how to calculate the four basic metrics of flow at the individual work item level. Further, we now know that all of these calculations are deterministic. That is, if we start a work item on Monday and finish it a few days later on Thursday, then we know that the work item had a Cycle Time of *exactly* four days.

But what if someone asks us what our overall process Cycle Time is? What if someone asks us what our team's Throughput is? How do we answer those questions?

You should immediately see the problem here. If, say, we look at our team's Cycle Time for the past six months, we will see that we had work items finish in a wide range of times. Some in one day, some in five days, some in more than 14 days, etc. In short, there is no single deterministic answer to the question "What is our process Cycle Time?". Stated slightly differently, your process Cycle Time is not a unique number, rather it is a distribution of possible values. That's because your process Cycle Time is really what's known as a random variable. [By the way, we've only been talking about Cycle Time in this section for illustrative purposes, but all of the basic metrics of flow (WIP, Cycle Time, Age, Throughput) are all random variables.]

What random variables are and why you should care is one of those topics that is way beyond the scope of this book. But what you do need to know is that your process is dominated by uncertainty and risk, which means all flow metrics that you track will reflect that uncertainty and risk, and further, that uncertainty and risk will show up as randomness in all of your Flow Metric calculations. The broader implication is that once randomness shows up, you can throw determinism out the window. Once you know you are dealing with a random process, you are required to take a probabilistic approach. We'll talk about what that means starting in the next chapter.

The Flaw of Averages

The lazy and innumerate love to shortcut probabilistic thinking by using the term "average". To illustrate, allow me to ask: do you still commute to work? If so, how long does it take you? Maybe your answer is something like "On average it takes me 25 minutes to get to work". If that was your answer, you have fallen victim to the Flaw of Averages (FoA).

The Flaw of Averages (FoA) is a concept detailed in the book of the same name by Dr. Sam Savage[2]. Simply put, the Flaw of Averages can be stated as "plans based on average fail on average".

To explain FoA, I'd like to use the same example that Dr. Savage uses in many of his talks. Let's assume there is a 9:00 AM business meeting with 10 people invited and that all attendees must be present before the meeting can begin. Let's further assume that, on average,

all participants have a history of arriving to meetings on time (for this example we'll say that average means a 50% on-time record). What are the chances that the meeting will start on time?

Again, you might think the answer to this question is easy. If, on average, everyone has a history of arriving on time, then it is reasonable to assume that there is an average chance that the meeting will start on time. Unfortunately, again, this answer is wrong. If everyone has the same chance of arriving on time as arriving late, then there is actually only a 0.1% chance that the meeting will start on time. Think of it this way: since every invitee has a 50% chance of arriving on time, then you could use the flip of a coin to model if a given attendee will arrive punctually—heads she/he does and tails she/he doesn't. Remember the meeting can only start when all participants arrive. Therefore, the case where the meeting starts on time is the equivalent of flipping 10 heads in a row—flipping only one tails means that the participant is late and the meeting itself starts late. The chance of flipping 10 heads in a row is 0.1% ($1/2^{10}$)—or about 1 in 1,000. There is virtually no chance the meeting starts on time—which is significantly worse than average.

This point can be illustrated by a joke you've probably heard, "If [insert name of world's richest person here] walks into a bar (pub), then, on average, everyone in the bar is a billionaire". Averages don't mean much outside of some very specific use cases. Predictability isn't one of those exceptions.

The lesson here is that any time you hear someone say "on average…" your ears should perk up because anything after the "on average…" statement will contain little to no informational value. For example, I currently live in South Florida, and as you may know, Florida is fairly prone to being hit by big storms known as hurricanes. Before every hurricane season, forecasters go through their song and dance to try and predict the severity of the upcoming season. You will hear statements like, "The 2022 hurricane season will be more active than average." It should be immediately obvious to you now how that statement is a classic case of the FoA and contains no informational value whatsoever. By comparing a single value (the 2022 hurricane season) to an average (the average of all hurricane seasons in the past) then you would expect that about 50% of the time the upcoming season will be more active than average and about 50% of the time the

upcoming season will be less active than average. So saying the 2022 season will be more active than average doesn't really tell us anything because that forecast has just as much chance of being right as being wrong.

One final thought on the FoA before we get to more pressing matters. English speakers are very lazy and many times they will use the word "average" when what they really mean is "typical". In our commute example, if someone were to say "On average it takes me 25 minutes to get to work" what they probably really mean is "typically it takes me 25 minutes to get to work". The problem with this laziness is that when it comes to probability (and the FoA) average is most often very far from typical—and in many cases average may not even be a possible outcome.

To demonstrate, if you were to roll a single, fair, six-sided die, the average outcome would be 3.5. But we know that it is impossible to roll a 3.5, so once again saying that "on average we will roll a 3.5" provides no informational value.

Even when the average is in the realm of possibilities, the average outcome usually isn't very likely. Let's say we want to roll two, fair, six-sided dice. The average outcome in this scenario is (did you have to look it up?) about 16.7%. This is a quirk of averages that most people have a hard time coming to grips with. Even though rolling seven is the average outcome and even though in this case rolling seven is the most likely outcome, the chances of actually rolling a seven are pretty low. To quote my friend and colleague Frank Vega, "The most likely outcome is not very likely". In other words, would you bet on something if ***on average*** you only had a 16.7% chance of succeeding? In short, don't use averages when talking about flow metrics.

Conclusion

Admittedly, the past three and a half chapters have been a bit of a slog. Unfortunately, that was all information that you needed to have before we could get to the heavy-lifting part of this book. You may want to stretch your legs or get a cuppa (or something stronger) because now is when we get into where the predictability rubber meets the road.

Key Learnings and Takeaways

- Once started and finished points have been defined, then the minimum data to collect is a timestamp for when an item starts and a timestamp for when an item finishes.
- WIP is the count of all work items that have a started timestamp but do not have a finished timestamp for a given interval of time.
- Cycle Time equals the finished date minus the started date plus one (CT = FD - SD + 1).
- Work Item Age equals the current date minus the started date plus one (Age = CD - SD + 1).
- Throughput is a count of items that are finished on a given date.
- The flow metrics for your overall process as random variables. As such, when working with them, we will need to take a probabilistic approach.
- When using probability, beware the Flaw of Averages. There are few things more dangerous than the word "average" when it comes to probabilistic thinking. If you ever hear someone discussing averages, then you can be sure they know nothing about predictability.

Chapter 4 - Introduction to Cycle Time Scatterplots

When I begin my analysis of any Agile team that I am working with I almost always start by placing the historical Cycle Time Data into a Scatterplot. I can think of only one other chart (Chapter 6) that even comes close to giving you the analytic insights you need at a glance.

But before I get into the explanation about how to do basic quantitative and qualitative analysis using Scatterplots, I need to make one thing clear about how to read this chapter. For this discussion, I am going to focus only on how to chart the flow metric of Cycle Time on a Scatterplot. In reality, you can put pretty much any metric that you want to in a Scatterplot. You can put things like Throughput, bugs per feature, work items per epic, etc. For this chapter, however, whenever I say the word "Scatterplot" without any qualifier, what I really mean is "Cycle Time Scatterplot" (if you would like a refresher on how I am choosing to define Cycle Time, then please revisit Chapter 2).

What is a Cycle Time Scatterplot?

It will first be beneficial to get a basic understanding of a Scatterplot's anatomy before diving into what these charts can tell us.

If you have never seen a Cycle Time Scatterplot before, then one is displayed in Figure 4.1 for your reference:

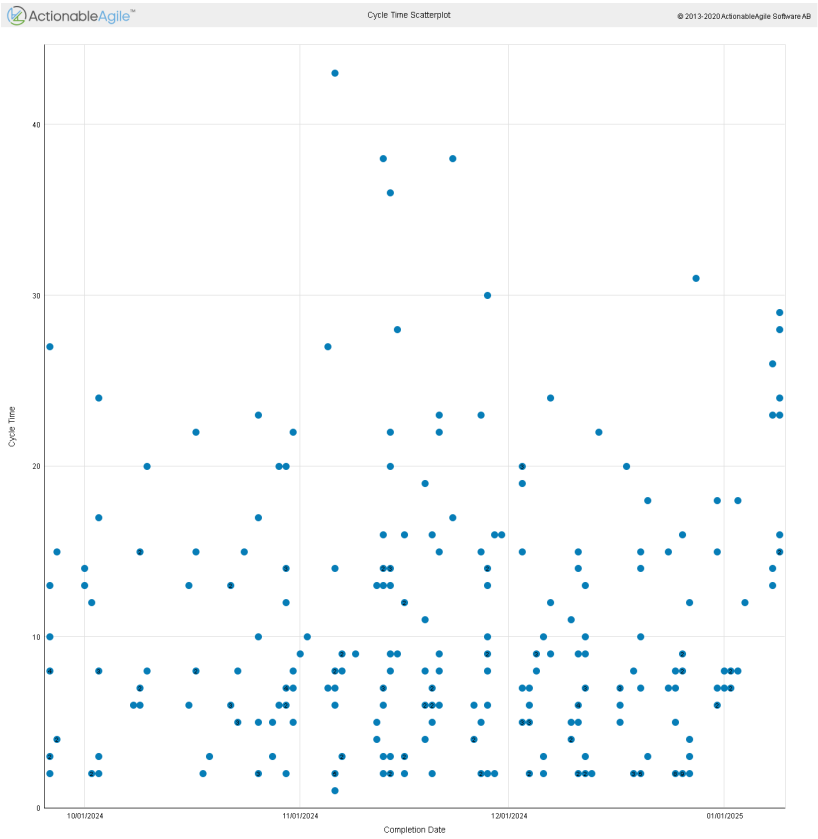

Figure 4.1: A Basic Cycle Time Scatterplot

As you can see from Figure 4.1, across the bottom (the X-axis) is some representation of the progression of time. Like CFDs, the X-axis essentially represents a timeline for our process. The tick marks on the X-axis represent our choice of labels for that timeline. When labeling the X-axis, you can choose whatever frequency of labels you want. In this particular Scatterplot, we have chosen to label every month. However, you can choose whatever label is best for your specific needs. You can choose to label every two weeks, every month, every day, etc.

I should point out that in Figure 4.1 I have chosen to show the timeline progression from left to right. This is not a requirement, it is only a preference. I could have easily shown time progression from

right to left. I personally have never seen a Cycle Time Scatterplot that shows time progression from right to left, but there is no reason why one could not be constructed that way. However, for the rest of this chapter (and this book), I will show all Scatterplot time progressions from left to right.

Up the side (the Y-axis) of your chart is going to be some representation of Cycle Time. Again, you can choose whatever units of Cycle Time that you want for this axis. For example, you can measure Cycle Time in days, weeks, months, etc.

To generate a Scatterplot, any time a work item is completed, you find the date that it was completed across the bottom and plot a dot on the chart area according to its Cycle Time. For example, let's say a work item took seven days to complete and it finished January 1, 2013. On the Scatterplot you would go across the bottom to find January 1, 2013, and then go up and put it a dot at seven days. Recall that for CFDs you could choose whatever time reporting interval you wanted to plot your data. In a Scatterplot, however, there is no concept of a reporting interval. A dot is always plotted on the day a given work item finishes.

Note that you could have several items that finish on the same day with the same Cycle Time. In that case, you would simply plot the several dots on top of one another. Hopefully whatever tool you are using to plot your Scatterplot can handle this case, and, further, can alert you to the instances where you have several dots on top of each other. In the ActionableAgile® Analytics tool, we signify this situation by putting a little number on the dot to show there is more than one work item located at that point (as also shown in Figure 4.1).

Over time as you plot more and more work item completions, a random set of dots will emerge on your chart. The original diagram I showed you in Figure 4.1 is a good example of what I am talking about. So how do we get useful information off of a chart that just looks like a bunch of random dots?

Percentile Lines

The first thing that we can do to gain a better understanding of our process's Cycle Time performance is to draw what I would call "standard percentile lines" on our Scatterplot. I should stress upfront

that this standard percentile approach is only a starting point—you will have every opportunity to change these percentiles as you get a better understanding of your context. I would argue, however, that these standard percentiles represent a good enough place to start for most teams.

The best way to explain how to use standard percentiles on a Scatterplot is by example. I want to refer you again to the chart shown in Figure 4.1. Looking at this graph the first line that we could draw would be at the 50th percentile of Cycle Times. The 50th percentile line is going to represent the value for a Cycle Time such that if we draw a line completely across the chart at that Cycle Time, 50% of the dots on the chart fall below that line and 50% of the dots are above that line. This calculation is shown in Figure 4.2 below.

In Figure 4.2, the 50th percentile line occurs at seven days. That means that 50% of the work items that have flowed through our process took seven days or less to complete. Another way of saying that is that when a work item enters our process it has a 50% chance of finishing in seven days or less (more on this concept a little later).

The next line that might be of interest to us is the 85th percentile. Again this line represents the amount of time it took for 85% of our work items to finish. In Figure 4.2 you can see that the 85th percentile line occurs at 16 days. That means that 85% of the dots on our chart fall below that line and 15% of the dots on our chart fall above that line. This percentile line tells us that when a work item enters our process it has an 85% chance of finishing in 16 days or less.

Another line we might want to draw is the 95th percentile line. As before, this line represents the amount of time at which 95% of our work items are complete. In Figure 4.2 the 95th percentile line occurs at 23 days and tells us that our work items have a 95% chance of finishing in 23 days or less. This calculation is shown in Figure 4.2 below.

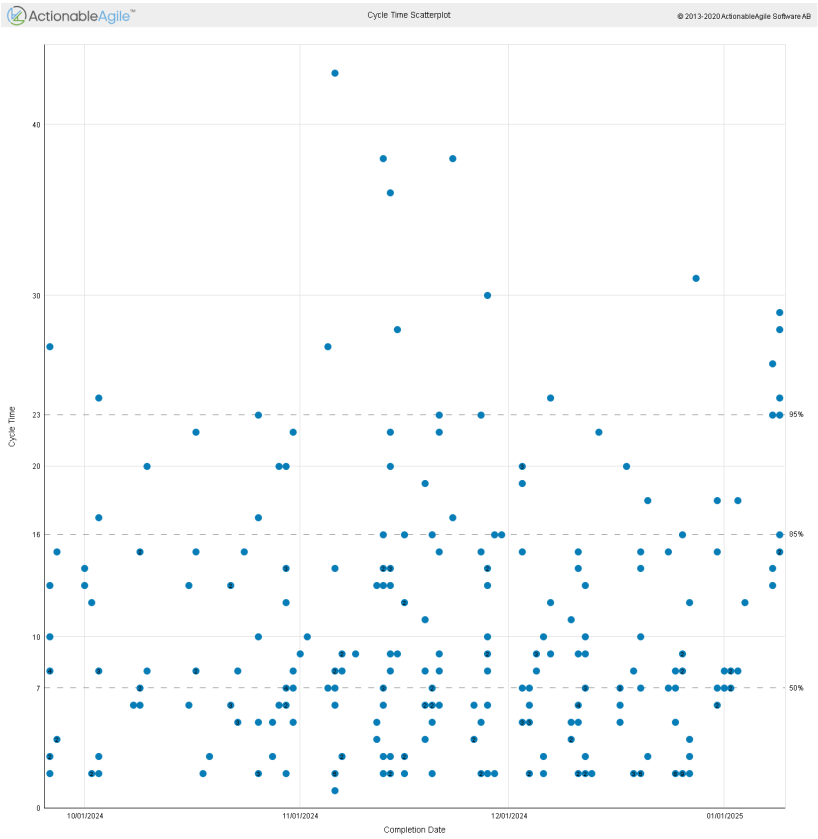

Figure 4.2: 50, 85, and 95 Percentile Lines all shown on a Scatterplot

The 50th, 85th, and 95th percentiles are probably the most popular "standard" percentiles to draw. Other percentiles that you will see, though, could include the 30th and 70th. Calculating those percentiles is exactly the same as I have just demonstrated with the others.

I am sure you have noticed that as we increase our level of confidence we have to increase the amount of time it takes for work items to complete. This is due to the variability inherent in our process. We will spend a little bit of time talking about variability later in this chapter. What we will see in that discussion is that no matter how hard we try to drive it out, variability will always be present in our system. But that is okay. It turns out that we do need a little variability to protect

flow. However, what we are going to want to understand is how much of that variability is self-imposed, and how much of that variability is outside of our control. The good news is that I will give you ways to identify each of these cases and strategies with which to handle them.

As I mentioned earlier, drawing these standard percentile lines is a good start, but you can see that you can easily add or subtract other percentile lines to your chart as you see fit. Which lines to draw is mostly going to be a function of what you want to learn from your data.

Your Data is Not Normal

Many electronic tools will draw arithmetic mean and standard deviation lines on their Scatterplots instead of drawing the standard percentile lines as described above. That is to say, these tools will figure out the arithmetic mean of all of their Cycle Time data and then first draw that horizontal line on the chart. They will then compute a standard deviation for that data and draw horizontal lines corresponding to the mean plus one standard deviation and the mean minus one standard deviation.

They might go further and draw the +2 standard deviation and -2 standard deviation lines as well as the +3 standard deviation and -3 standard deviation lines. They will call the top standard deviation line the "Upper Control Limit" (UCL) and they will call the bottom standard deviation line the "Lower Control Limit" (LCL). They will then call the resulting graph a "Control Chart". If you are using an electronic tool to track your process maybe you have seen an example of a Control Chart.

You might have further heard several claims about these charts. First, you may have heard that on a Control Chart (as described above) about 68% of the dots fall between the plus-one standard deviation line and the minus-one standard deviation line. They might further go on to say that over 99% of the dots fall between the +3 standard deviation in the -3 standard deviation line. You might have further heard that the reason you want to segment your data this way is because this type of visualization will be able to tell you if your process is in control or not (hence the name Control Chart). Any dots that fall above the UCL or below the LCL, it is argued, signify the points in your process that are

out of control.

What is being called a Control Chart here is supposedly inspired by the work of Walter A. Shewhart while employed at Bell Labs in the 1920s[1]. Shewhart's work was later picked up by W. Edwards Deming[2] who became one of the biggest proponents of the Control Chart visualization.

There is only one problem. By using the method outlined above what they have created is most certainly not a Control Chart—at least not in the Shewhart tradition. What Shewhart Control Charts are and how to construct them are the topic of Volume II of this book series, but just know that you should be skeptical whenever you see someone show you something that is labeled "Control Chart"—as it most certainly is not.

As these things usually go, the problem is much worse than you might think. That tools vendors' charts are most assuredly not Control Charts notwithstanding, there remains one (at least) fatal flaw with a pseudo-Control Chart approach. These charts—especially the calculations for the UCLs and LCLs—assume that your data is normally distributed (to be clear, Shewhart *never* made this assumption). I can all but guarantee you that your Cycle Time data is not and will not be normally distributed. So know that for now the conclusions based on the standard deviation calculations above when applied to your non-normally distributed data will be incorrect.

The use of this normal distribution method is so pervasive because that is the type of statistics that most of us are familiar with. One very important consequence of working in the knowledge work domain is that you pretty much have to forget any statistics training that you may have had up until this point (for a great book on why we need to forget the statistics that we have been taught read "The Flaw of Averages"). We do not live in a world of normal distributions. But as we are about to see with Scatterplots, that is not going to be a problem at all.

As a quick aside, you may have also heard the name "Run Chart" in association with these diagrams. Again, the Scatterplots I am talking about here are not Run Charts. A deep discussion of Run Charts is also covered in Volume II of this book series. I am not saying that Run Charts are not useful, by the way—far from it. I am just trying to be clear that this chapter's Cycle Time Scatterplots are most certainly not

Run Charts.

Getting back to standard percentiles, there are at least three reasons why I like those lines better than the dubious Control Chart tactic mentioned above. First, notice that when I described how to draw the standard percentile lines on a Scatterplot I never made one mention of how the underlying Cycle Time data might be distributed. And that is the beauty of it. To draw those lines I don't need to know how your data is distributed. In fact, I don't care. These percentile line calculations work regardless of the underlying distribution.

Second, note how simple the calculations are. You just count up all the dots and multiply by percentages. Simple. You are not required to have an advanced degree in statistics in order to draw these lines.

Third, percentiles are not skewed by outliers. One of the great disadvantages of a mean and standard deviation approach (other than the false assumption of normally distributed data) is that both of those statistics are heavily influenced by outliers. You have probably heard the saying, "If Bill Gates walks into a bar, then on average everyone in the bar is a millionaire". Obviously, in the Bill Gates example, the average is no longer a useful statistic. The same type of phenomenon happens in our world. However, when you do get those extreme Cycle Time outliers, your percentile lines will not budge all that much. It is this robustness in the face of outliers that is why percentile lines are generally better statistics for the analysis of Cycle Time.

As I mentioned at the beginning of this section, chances are if you are using an electronic tool for metrics it will not show you a Scatterplot view with percentile lines overlaid. So what are you to do? You can use a tool like Excel and generate the charts yourself. Or you can use the ActionableAgile® Analytics tool as it takes care of everything for you.

Conclusion

Randomness exists in all processes. One of the best ways to visualize the randomness in your process is to put your Cycle Time data into a Scatterplot. A Cycle Time Scatterplot can yield vast amounts of quantitative information and qualitative information.

I mentioned at the beginning of this chapter that Cycle Time Scatterplots are a great way to visualize Cycle Time data that goes far

beyond simple analysis by average. I hope that you are convinced of that now.

There is so much more to Scatterplots, but this brief introduction should be more than enough to get you started. For a guide on how to interpret some common patterns on Cycle Time Scatterplots, please see Appendix B. You'll want to study Appendix B closely because those patterns can give you an initial idea if your process is straying from predictability.

Better insight, however, comes from understanding why we made such a big fuss about those percentile lines in the first place. That explanation happens now.

Key Learnings and Takeaways

- Scatterplots are one of the best analytics for visualizing Cycle Time data.
- This type of visualization communicates a lot of quantitative and qualitative information at a glance.
- The anatomy of a Scatterplot is:
  - The X-axis represents the process timeline.
  - The Y-axis represents the Cycle Time for an item to complete.
  - The labels and reporting intervals on the chart are at the sole discretion of the graph's creator.
- A Cycle Time Scatterplot is not a Control Chart. It is not a Run Chart, either.
- One of the best ways to put some structure around Cycle Time Scatterplot data is to draw percentile lines. Consider starting with the 50^{th}, 70^{th}, 85^{th}, and 95^{th} percentiles.
- Percentiles have the advantages of being easy to calculate, being agnostic of the underlying data distribution, and not being skewed by outliers.

Chapter 5 - Service Level Expectations

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

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

Work Items in a flow system 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 predictability.

So, no, in a flow system items don't get to take as long as they want to finish. But what is the forcing function to make sure that items do indeed finish? 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 with 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 that 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 being heads (and a 50% chance of it being tails). As another example, before I roll this 6-sided die tell me with 100% certainty 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 to 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".

A forecast is a statement about the future that contains a range of possible outcomes and a probability of an outcome occurring within that range.

Putting this all together, when our customers ask "When will it be done?" we need to answer them with a forecast. With flow, 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.

A Service Level Expectation is a probabilistic statement about how long it should take a work item to finish once started.

While this is not a book specifically about Kanban, one of the best statements about an SLE is from the Kanban Guide: "The SLE is a forecast of how long it should take a single work item to flow from start to finish. 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."[1]

The SLE serves two functions. First, it provides a completion forecast for work items once they have started. Second, the SLE helps us to answer the question, "How much age is too much age?".

Calculating an SLE

The way we determine what date range and confidence level that we can reasonably commit to is by looking at the percentile lines on our Scatterplot.

To explain, I want to refer you back to Figure 4.2. You can see in this diagram that the 50th percentile for the Cycle Times is 7 days, the 85th percentile is 15 days, and the 95th percentile is 23 days. That means that any item that enters our process has a 50% chance of finishing in 7 days or less, an 85% chance of finishing in 15 days or less, or a 95% chance of finishing in 23 days or less. Armed with this information we can sit down with our customers and ask them what kind of confidence level they would be most comfortable with. If they are ok with us missing our commitments 50% of the time, then the team would choose 7 days at 50% as its SLE. If, however, they want greater confidence in terms of the team meeting its commitments, then the team may choose to go with an SLE of 23 days at 85%. To reiterate, the choice of a team's SLE should be made in close collaboration with their customers.

While there is no hard and fast rule on this, it is been my experience that most teams start at the 85th percentile as their SLE. The goal of the team then should be to first meet that SLE at least 85% of the time (true predictability) but then also to bring down the total number of days that the 85th percentile represents over time. Part of process improvement is going to be to shift all the percentile lines down as much as possible (but no further!). A wider spread in those lines means not only a higher number of days that we must communicate for our SLE, but it also means that our process is suffering from more variability. Both of those things decrease our overall predictability.

Take the following example of Figure 5.1:

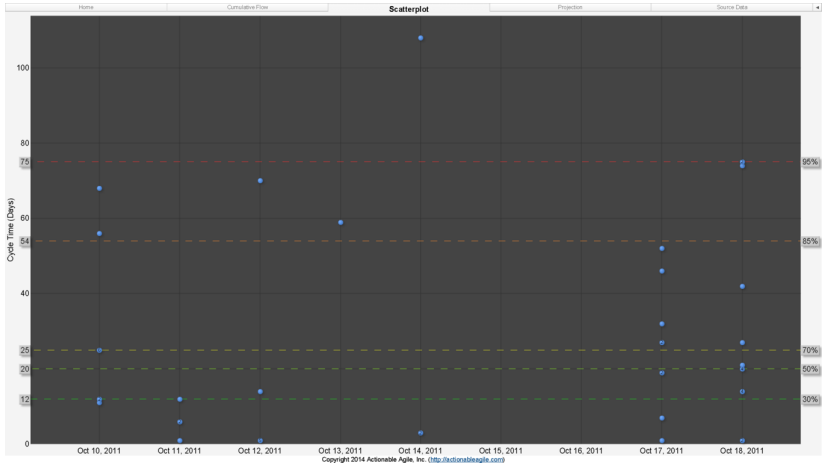

Figure 5.1: A Wider Spread in Percentiles

In Figure 5.1 the 50th percentile for the chart is 20 days, the 70th percentile is 25 days, the 85th percentile is 54 days, and the 95th percentile is 75 days. Think for a second about what an interesting conversation this would be when we present this data to our customers. At 70% confidence, the team would require a 25-day or less SLE. But to go to an 85% confidence—that is just a 15% increase in confidence—the team would have to *more than double* their SLE from 25 days to 54 days! This particular example is taken from a real-world client of mine and, in this instance, the customer chose the 70% SLE to start. Interestingly enough, though, the team, by implementing the strategies outlined in this book, was able to shift all of those percentile lines down over the course of the project such that by the end, the 85% percentile was now 25 days—exactly what the 70% percentile had been just months before. The team removed unnecessary variability, and, by definition, became more predictable.

I have just explained how to use standard percentiles to establish an SLE, but you might question, "How do I know if these standard percentiles are the right ones to use for my context?" Great question. The answer is that if you are just starting, then those standard percentiles are most likely good enough. How you might detect if you need to move to another percentile more suitable for your specific situation is a more advanced topic that will need to wait for my next book. The

point is that there is no hard and fast rule in terms of what percentile numbers to use. All I can say is to begin with these standard ones and experiment from there.

Another question you might ask is, "How many data points do I need before I can establish an SLE?" The answer to that is—as always—dependent on your specific context. But I can tell you it is probably less than you think. As few as maybe 11 or 12. Probably no more than 30. The bigger question is in terms of quality not quantity. Instead of considering the number of dots, one question you may ask yourself is how well your process is obeying the assumptions of Little's Law in producing those Cycle Times. The better you are at adhering to those assumptions, the fewer data points you will need. If you consistently violate some or all of the assumptions, then almost no amount of data is going to provide you with a confidence level that you can be comfortable with.

The last thing I want to say about SLEs is that there are generally three mistakes I see when they are set. Those mistakes are:

1. To set an SLE independent of analyzing your Cycle Time data.
2. To allow an SLE to be set by an external manager or external management group.
3. Set an SLE without collaborating with customers and/or other stakeholders.

For the first point, I want to say that there is nothing (necessarily) wrong with choosing an SLE that is not supported by the data. For example, let's say your data communicates that 85th percentile is 45 days. It would technically be ok to publish an SLE of 35 days at 85%. But at least make that decision in context after having reviewed what your Scatterplot is telling you.

The second mistake should be obvious, but it is worth reiterating. The whole point of an SLE is not to beat a team into submission or to punish them when they miss their commitments. Since it is the team that is making the commitment, it should be the team that chooses what that commitment point is. The only other party that should be involved in the decision to set an SLE should be a customer and/or other stakeholder.

Which brings me to the last point. We are nothing without our customers. As stated in Chapter 1, they are the whole reason for our

existence. It is our professional obligation to design a process that works for them. Therefore, our customers should have a seat at the table when discussing what commitment confidence level is acceptable. They may surprise you. They may opt for a shorter Cycle Time SLE with a higher uncertainty. They may be fine with a longer Cycle Time SLE if that means greater certainty. Our customers and stakeholders almost certainly have contextual information that we do not that will have some bearing on our choice of an SLE. Listen to them.

SLEs for Different Work Item Types

In Appendix A, I talked about the strategy of filtering different work item types to generate different views of your data. The same approach is available for us to use on Cycle Time Scatterplots. Let's say we had a dataset that included the work item types of user stories, defects, and maintenance requests. With this data, we could generate a Scatterplot and corresponding percentile lines for the data that included all three work items. Or we could generate a Scatterplot that included data for just the user stories. Or one that included just the defects, or one for just the maintenance requests, or one for some combination thereof. As with CFDs, any one of these data segmentations—and their corresponding analysis—is perfectly valid.

But why might we want to segment our data in this way? There are at least two answers to this question. The first might be that you have tagged the items that did not finish "normally" (e.g., were abandoned) and want to filter your data to show only those. Displaying only the abandoned items would give you a good visualization of the time wasted on those activities. That might give rise to questions and conversations about how to minimize those occurrences.

The second reason for segmenting is that the Cycle Time percentiles for a Scatterplot consisting of data for only the work item type of "story" is probably going to be much different from the Cycle Time percentiles for a Scatterplot consisting of data for only the work item type of "defect". Segmenting our data this way would allow us—if we wanted—to offer different SLEs for different work item types. For example, our SLE for user stories might be 14 days at 85% but for defects, it might be five days at 85%.

I am reluctant to discuss this SLE segmentation now because you have to be very careful here. Remember that all the assumptions of Little's Law still apply. If you are going to offer different SLEs for different work item types, then you have to ensure that all the assumptions for Little's Law for each and every subtype are adhered to.

Offering different SLEs for different work item types is a fairly advanced behavior. If you are just starting out with flow principles, I would highly recommend just setting one global SLE for all your work item types and getting predictable that way first. Ignore "conventional wisdom" that you have to design in things like Classes of Service up front and offer different SLEs for those different Classes of Service immediately. To put it delicately, I believe this type of advice is misguided (a fuller treatment of Class of Service and its dangers is presented in Chapter 13). If you are new to these metrics, begin by applying the principles presented in this book and then measure and observe. Get predictable at an overall system level first. You may find that is good enough. Only optimize for subtypes later if you absolutely need to.

Percentiles as Intervention Triggers

There is still another reason to look at our Cycle Time data percentiles as they pertain to SLEs. And to understand this other reason, we need to first talk about life expectancy.

According to a life expectancy calculator at WorldLifeExpectancy.com (at the time of this writing), a female born in the United States has a life expectancy of 85.8 years at the time of her birth. If she lives to be 5 years old, her life expectancy goes up to 86.1 years. If she lives to be 50, her life expectancy becomes 87.3 years. And if she lives to be 85 (her life expectancy at the time of her birth), her new life expectancy jumps to 93! This data is summarized in the following table:

Age	Life Expectancy (in years) at that Age
Birth	85.8
5	86.1
50	87.3
85	93

Figure 5.2: Life Expectancies at Different Ages

It is a little-known fact that the older you get, the longer your life expectancy is. That is because the older you get the more things you have survived that should have killed you.

The same phenomenon happens with Cycle Time. Generally speaking, the older a work item gets, the greater chance it has of aging still more. That is bad. Remember, delay is the enemy of flow!

This is why it is so important to study the aging of work items in progress. As items age (as items remain in progress without completing), we gain information about them. We need to use this information to our advantage because, as I have said many times before, the true definition of Agile is the ability to respond quickly to new information. To paraphrase Don Reinertsen, this new information should cause our tactics to change[1]. 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 ensure 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 a 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? (More on these actions later.) 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 bigger 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 (i.e. when the item hit the 50th percentile line) 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 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 from 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.

However, once we breach our SLE we are squarely in unpredictable land because now we cannot communicate to our customers when this particular item will complete. For example, take a look at the figure below (Figure 5.3):

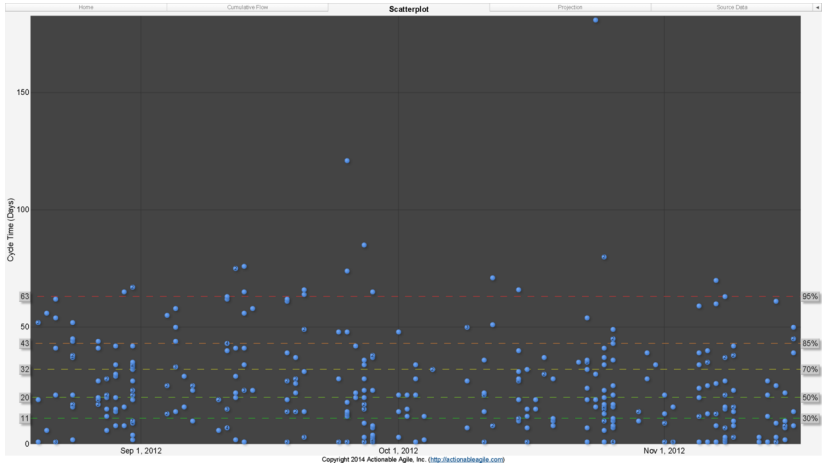

Figure 5.3: The Danger of Breaching an SLE

You can see in this chart that the 85th percentile is 43 days. But there is an item in late October that took 181 days to finish (do you see that isolated dot right at the top of the chart?). That no man's land between 43 days and 181 days (and potentially beyond) is a scary place to be in. We want to do whatever we can not to have items fall in there.

Right-Sizing

One last thing about percentiles and SLEs. More than being able to forecast when an item will complete, the biggest advantage of calculating an SLE is it allows us to perform a "right-size" check on the item before we pull that item into progress.

Before you ask, right-sizing does not mean you do a lot of upfront estimation and planning. Remember, this book emphasizes measurement and observation over estimation and planning. The SLE we have chosen is the measurement we are looking for. In other words, the SLE will act as the litmus test for whether an item is of the right-size to flow through the system. For example, let's say we have chosen an SLE of fourteen days or less at 85%. Before a team pulls an item into the process, a quick question should be asked if the team believes that this particular item can be finished in fourteen days or less. The length of

this conversation should be measured in seconds. Seriously, seconds. Remember, at this point, we do not care if we think this item is going to take exactly five days, or exactly nine days, or exactly 8.247 days. We are not interested in that type of precision as it is impossible to attain that upfront. We also do not care what the particular relative complexity is compared to the other items. The only thing we do care about is we think we can get it done in 14 days or less. If the answer to that question is yes, then the conversation is over and the item is pulled. If the answer is no, then maybe the team goes off and thinks about how to break it up, change the fidelity (e.g., tweak acceptance criteria), or spike it to get more information.

Some of you out there may be arguing that right-sizing is a form of estimation. I would say that you are probably right. I never said that all estimation goes away. All I said was that the amount and frequency with which you do estimation will change. Think about all the time you have wasted in your life doing estimation. Think about all the time wasted in "pointless" debates of whether a story is two points or three points. Using these percentiles is a means to get rid of all of that. Measuring to get an SLE allows us to adopt a much lighter approach to estimation and planning. To me, this is one of the biggest reasons to gather the data in the first place.

Conclusion

SLEs are one of the most important and yet least talked about topics in all of Lean-Agile. SLEs not only allow teams to make commitments at the individual work item level, but they also give us extremely useful information about when teams need to intervene to ensure the timely completion of those items. Further, if a team follows all of the principles presented in this book, then the SLE can be used as a substitute for many upfront planning and estimation activities.

Even so, there is a much more effective tactical way to use SLEs—especially when talking about percentiles as intervention triggers. This is where the predictability rubber meets the road, so you are going to really want to pay attention to what comes next.

Key Learnings and Takeaways

- Use your Scatterplot's percentiles to collaborate with your customers in choosing a Service Level Expectation for your process.
- It is possible to segment your data by type. You might choose to do this to offer different SLEs for different work item types in your process.
- SLEs allow for commitment (and estimation) at the work item level.
- SLEs provide a sense of urgency to items that have been committed to.
- You can also use Cycle Time data percentiles as a guide for "right-sizing" items that come into your process. Use this right-sizing as a shortcut for estimation.
- Comparing an item's age to its SLE can provide useful information about when to intervene to ensure timely completion.

Chapter 6 - The Work Item Aging Chart

The problem with the Cycle Time Scatterplot—if there is a problem with that chart—is that by definition, a dot does not show up on a Scatterplot until an item has finished. However, if something is taking too long to complete, waiting until it is finished to get the signal that it is taking too long to complete is too late. Ideally what we want is a much, much earlier signal that maybe something is wrong so that we can do something about it. Enter the Aging Work In Progress (or Work Item Aging Chart or simply "Aging Chart" for short).

What Is a Work Item Aging Chart?

To understand an Aging chart, let's consider a process workflow that looks like this:

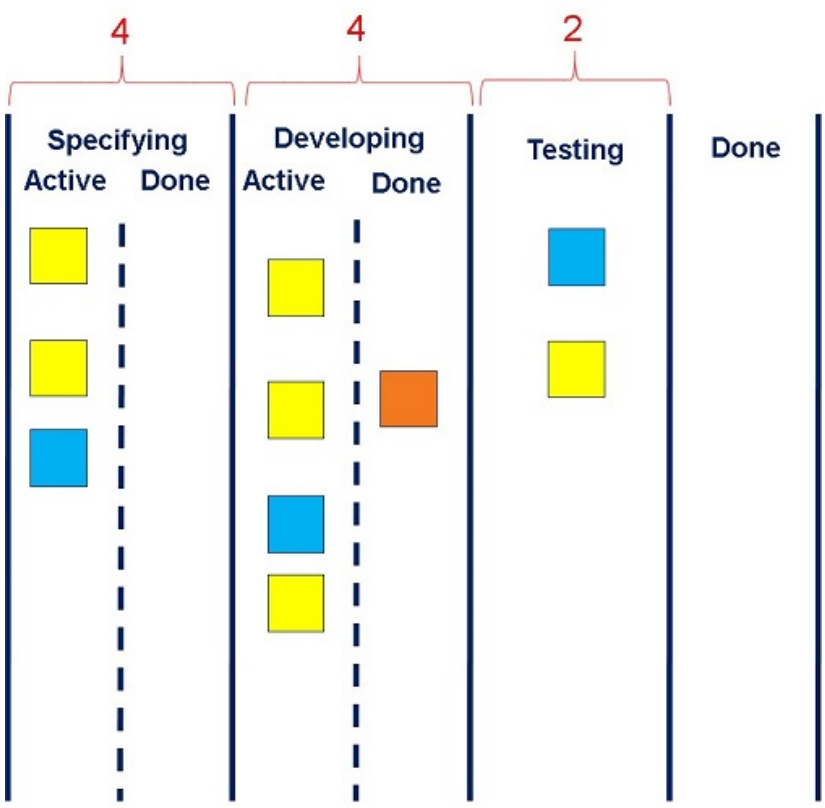

Figure 6.1 - Sample Workflow

An example Aging Work In Progress chart for this particular workflow might look like Figure 6.2:

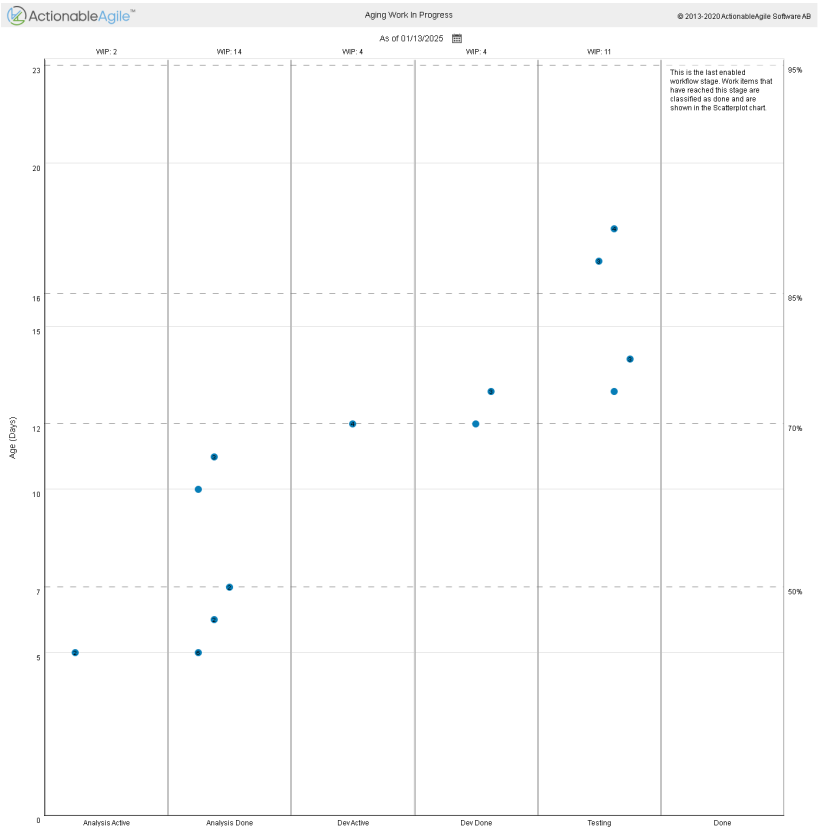

Figure 6.2 - Aging Work In Progress Chart for Sample Workflow

Before I get into how this chart should be used, let me quickly go over the anatomy of the chart so you know what you are looking at. Unlike the Scatterplot, you can see that across the bottom all of the states of your workflow have been mapped out in the same order as they appear in Figure 6.1. In fact, the whole chart itself has been segmented into columns to match your process's workflow (much like a Kanban board). Up the side of the chart is the Age of work items. Recall that Age is defined as the amount of elapsed time that an item has spent inside the workflow.

Obviously, Age in this context is very different from Cycle Time on the Scatterplot. What we are representing on this chart (Figure

6.2) is the total elapsed time that an item has spent started but not completed (Age) as opposed to the total elapsed time it took for an item to complete (Cycle Time). However, just like Cycle Time, for Age you can use whatever time units you want: days, weeks, months, sprints, etc. Thus, every dot on the chart represents an item that has entered the process but has not exited the process. To plot a dot, you simply find the workflow stage that it is currently in and then subtract today's date from the item's start date (remember you should have tracked the timestamp for when the item entered your process!).

At the risk of belabouring the subject, I would like to re-emphasize that the "Age" of each item is the **total** elapsed time from your chosen start point. Therefore, don't be confused in thinking that the height of each dot represents the amount of time that an item has spent in that particular column. For example, in Figure 6.2 you'll see a dot in the Analysis Done column at 10 days. That does not mean that that work item has been in Analysis Done for 10 days. Rather, it means that 10 days have elapsed since the item crossed our well-defined start point—in this case that is the Analysis Active column. I emphasize this point as some tooling only tracks Age per workflow state. As we will see in the next chapter, while time per state may be of interest, what we really care about for predictability is the total amount of elapsed time that an unfinished item has spent as Work In Progress.

You'll also notice that some dots in Figure 6.2 have numbers inside of them. For example, it might be hard to see, but the top dot in the Testing column has a number four inside of it. What the "4" is telling us is that that is not a single dot. It is actually four dots right on top of each other. If you are using the ActionableAgile® Analytics tool[1], you can click on the dot to expand it further as shown in Figure 6.3:

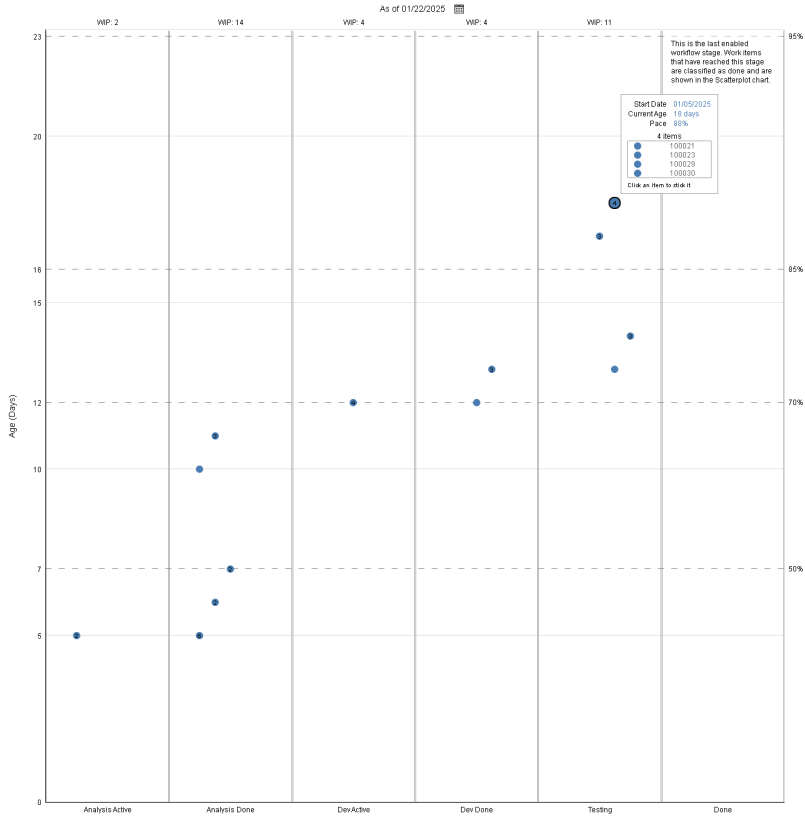

Figure 6.2 - Inspecting Multiple Items

What this is telling us is that there are four items in the Testing stage each with the same Age of 18 days. The dot right below that one (also in Testing) has a 3 in it. That means those next three items in Testing each have an Age of 17 days. And so on for any dot with a number on it in any column. So at first glance, it may only look like there are four dots in Test, you have to be careful and look closely as there may be more. Luckily, the chart itself helps us in that regard.

See the "WIP" text at the top of each column in Figure 6.2? These numbers are a point of possible confusion even though they are here to help us. Don't be fooled. Those numbers are not WIP Limits. They are simply a count of the number of items that are currently in progress and aging in that column. Continuing on the discussion from above,

the Testing column has a WIP of 11 (if you are verifying yourself, make sure to count the items that are all stacked on top of each other as one dot). That means there are currently 11 items in progress in Testing—not that your board has a WIP Limit of 11. You'll recall from Chapter 3 that we distinguished between WIP and WIP limits. The Aging Chart only communicates the former. Thus, this graph tells you nothing about whether the number of items that you currently have in progress is above or below any limit that you have established. To determine that, you'll have to look at your board—which is not a bad thing, by the way.

Next, you will see percentile lines on the Aging Chart. These percentile lines are the same lines that we calculated for our Scatterplot. Let me repeat that to be clear. The percentile lines that you see on your Aging Chart are the exact percentile lines that you calculated and placed on your Cycle Time Scatterplot. You overlay the Scatterplot percentiles lines on the Aging Chart so that you can see how items that are currently being worked on are progressing as compared to the total amount of time it took previous items to complete.

The percentile lines on the Aging Chart are the percentile lines from the Cycle Time Scatterplot. They are not percentile lines calculated from Aging data.

I started the explanation of why you would want to overlay Cycle Time percentiles on the Aging chart in the previous discussion about SLEs. I'll finish that explanation in the next chapter when we do a deep dive into how to leverage the Aging Chart. For now, simply be aware that those percentiles are Cycle Time percentiles, not Aging percentiles.

Finally, you'll note the last column on the chart. That last column represents our "finished" state. As such there are no dots in that column. If you look more closely at Figure 6.2 you will see that the text of the last column tells us as much: "…Items that have reached this stage are classified as done are shown in the Scatterplot chart." In other words, all work items age until they are finished at which point of that Age is immediately converted to Cycle Time. The dot thus disappears from the Aging Chart and reappears on the Scatterplot. Believe it or not, that is the exact property that we want from this chart as that is what is going to allow us to get the most out of it. But more on that in the next chapter.

Additional Data

To construct the chart seen in Figure 6.2 we are going to need a little more data than what I have previously mentioned. But not much. Instead of simply taking a timestamp of the start and finish points of your process, the data we'll need for the Aging Chart is a timestamp for the date that every item arrives to each stage of the workflow—as pictured below:

Story_ID	Analysis Active	Analysis Done	Development Active	Development Done	Testing	Done
1	06/25/2012	06/25/2012	06/26/2012	06/28/2012	06/29/2012	06/29/2012
2	06/25/2012	06/25/2012	06/27/2012	06/29/2012	06/29/2012	06/29/2012
3	06/21/2012	06/21/2012	06/21/2012	06/27/2012	06/27/2012	07/02/2012
4	06/21/2012	06/21/2012	06/21/2012	06/27/2012	06/27/2012	07/02/2012
5	06/21/2012	06/21/2012	06/21/2012	06/28/2012	07/02/2012	07/02/2012
6	06/21/2012	06/22/2012	06/22/2012	06/28/2012	06/28/2012	07/02/2012
7	06/25/2012	06/25/2012	06/25/2012	06/26/2012	06/29/2012	07/02/2012
8	06/25/2012	06/25/2012	06/25/2012	06/26/2012	06/29/2012	07/02/2012
9	06/21/2012	06/22/2012	06/22/2012	06/28/2012	06/28/2012	07/03/2012
10	06/25/2012	07/02/2012	07/02/2012	07/05/2012	07/06/2012	07/06/2012

Figure 6.3 - Data for the Aging Chart

The justification of why you need this type of data will be covered ad nauseam in Section II. For now, you'll just have to take it as a leap of faith that Figure 6.3 represents all the data you need for Aging.

[**Note**: There is absolutely no reason why you couldn't construct an Aging Chart with just start and end point timestamp data. In that case, your workflow would simply be To Do -> Doing -> Done—which is perfectly valid. And that might even be a decent enough place to start if it is hard to get more detailed process data. In general, though, it is preferred to be able to model each step of your workflow in an Aging Chart as in Figure 6.2. So even if you do start with To Do -> Doing -> Done, chances are you'll want to move to a more granular model as soon as is reasonable in order to exploit all of the advantages that the more detailed workflow grants you.]

Conclusion

The Aging chart will be the most important tool in your predictability arsenal—that's all you need to know about the chart itself. Remarkably, as has been shown, the anatomy of the Work Item Aging Chart is extremely simple. And as you will see, form indeed does follow function…

Key Learnings and Takeaways

- The columns on your Aging Chart represent the defined steps in your workflow.
- The height of a dot in any column represents the total Age for that item. It does not represent the amount of time that the item has spent in a particular column.
- One dot could represent multiple items of the same age.
- WIP at the top of the columns represents items currently in progress. It does not represent a WIP Limit.
- The percentiles on the Aging chart are the percentiles calculated from the Scatterplot. They are not percentiles calculated from Aging data.
- If possible, capture a timestamp for the date that every item arrives to each stage of the workflow.

Chapter 7 - Leveraging the Aging Chart for Predictability

We've talked about the most important flow metric for predictability and the most important chart for predictability, so let's now talk about potentially the most important place to use both of those.

The Daily Meeting

If you are currently practicing some flavour of Agile, then no doubt you have established some sort of daily meeting. That meeting could be called a standup, or daily scrum, or whatever. The name itself is of no consequence. What is of consequence is the purpose of that meeting—which, unfortunately, most teams get wrong.

The purpose of your daily is not to get status, nor is it to go around the room and hear from everyone. No, the purpose of your daily is for the team to come together and decide on the team's plan for the day.

The purpose of your daily meeting is for the team to come together and decide on the team's plan for the day.

By plan, I mean what collective actions is the team going to take to ensure the optimal execution of their process. In flow terms, that means taking action on the parts of the process where flow is suffering (or non-existent). Trust me when I say that there is no better indicator of problems with flow than Work Item Age.

Earlier we stated that flow implies movement; not just movement for movement's sake, but movement toward the delivery of customer value. But we know that once work items start, they begin to Age. That is not necessarily a bad thing. It is not reasonable to expect that work items will accumulate no Age as they are being worked on. So what is

reasonable to expect? It is reasonable to expect that items that are being worked on only Age to some agreed upon service level—you guessed it: a Service Level Expectation.

Percentiles As Intervention Triggers Redux

This is where we go back to our discussion in Chapter 5. The whole reason that we have Cycle Time percentile lines on the Aging chart is so that we can constantly compare current items' Age to how long it has taken us to complete work in the past. Specifically, we want to know how work is Aging relative to our SLE. Using the Aging chart, we can in near-real time determine what work items have either violated our SLE or are about to violate our SLE and, most importantly, take the necessary action.

Let's walk through an example to explain how this works. Assume for a moment that the Aging Chart in Figure 7.1 is for your team.

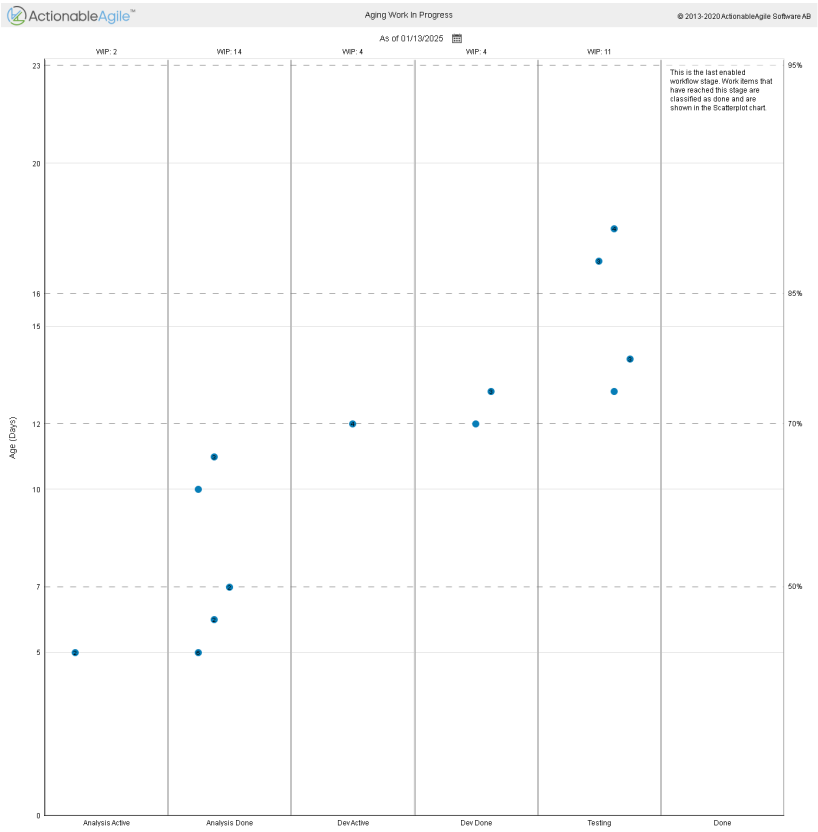

Figure 7.1: First Aging Chart

Further, assume that you are about to start your daily meeting and that your team has chosen the 85th percentile as your SLE. How might we leverage the information in Figure 7.1 during our meeting?

Before reading further, I'd like you to take a moment and consider what you might do in this situation. As a hint, remember that the thing we are looking for is the team's plan for the day. Also, remember that this plan should be based on attacking any problems that we see with flow. How might the Aging chart indicate such problems? Once you have your answer, then read on.

All other things being equal, issues with flow are indicated by items that have aged the most in our process. In Figure 7.1 specifically, if you

look closely, you will see that 7 items have aged past the team's SLE. In terms of our plan for the day, it should be all hands on deck to get those items finished (we'll talk through some more detailed optional actions below). Of even additional importance is to note how close those things are to being done. Why would anyone on the team focus on the item in Analysis Active, for example, when we have seven items that have breached their SLE and are one step away from being finished?

In case you haven't guessed it by now, the general heuristic for predictability is—assuming no other overriding contextual information—always work on the oldest items first. To be clear, I am saying work on the oldest item first regardless of what column it is in. It is really that simple. In your daily, pull up your Aging Chart and, as a team, start with the oldest item on the board and discuss what can be done to get that item moving (or even better to get that item to done). Then proceed to the next oldest item and do the same thing. Continue these steps until you have your plan for the day.

FIFO vs. FIFS
Many people mistake the "work on the oldest item first" strategy to mean "implement a First-In-First-Out (FIFO)" queuing strategy. That's not quite correct. What we are really implementing is a First-In-First-Served (FIFS) queuing strategy. I don't want to bore you with the details of FIFO vs. FIFS—I'll leave that as an exercise for you to look that up. I just want to point out here that I am advocating for the latter and not (necessarily) the former.

Note that by doing the preceding, you may not necessarily "walk the whole board". Or maybe not everyone on the team will have a chance to speak. That's ok. The goal is to come up with a plan *for the team.* The goal is not to come up with a plan for each individual. Once we have the plan, the meeting is over and let's get to work. Remember that flow (and thus predictability) is achieved by actually doing work not just talking about it.

Getting back to Figure 7.1, then, we would start our discussion with the oldest dot (which, as you can see, is the highest dot in Testing and as

we saw before is in reality four items represented as one dot). Discuss those. If we still feel like we have more capacity for the day, discuss the next oldest dot (which as we also saw before is in reality three items represented as one dot in Testing). Discuss those. We have now considered seven items. Maybe at that point, we decide that is all we can take on for today. Perfect. The meeting is over and let's get to work.

Some of you might still be worried about all of those other items on the board. Why not discuss those? Well, let's consider that for a second. What would be the problem with allocating team capacity to work on the item that I mentioned earlier in Analysis Active? If we as a team decide we want to work on that item instead of the four oldest in Testing, what would happen to those four oldest items? That's right, they would just get older. What's the problem with that?

This gets us to how I would like you to "see" the Aging Chart. I personally don't look at the Aging Chart as dots on this particular chart. The way that I look at the Aging Chart is every dot here is a potential dot on my Scatterplot (LL Assumption #2). In the last chapter, we learned that the text on the chart itself tells us this. When an item gets to done, its dot disappears from the Aging Chart and reappears on the Scatterplot (at the appropriate height for Cycle Time). Simply put, the higher a dot gets on the Aging Chart, the higher it will be on the Scatterplot. If we constantly allow items to Age unnecessarily, then over time the dots on our Scatterplot will get higher and higher, and correspondingly, our percentile lines will get wider and wider. As I've said so many times before, that is exactly what we don't want to happen.

Therefore, by examining our items' Age on a daily basis, we will be able to immediately spot issues with flow, and we will be able to take immediate action.

By examining our items' Age on a daily basis, we will be able to immediately spot issues with flow, and we will be able to take immediate action.

I cannot overstate the importance of that statement too much. Every day your team is making dozens of decisions that—whether they know it or not—impact Age. Decisions like, "What item(s) should we work on today?", "What item should we start next?", "Should we even

start something new?". "Should we be concerned about this blocked item?", etc. Traditional Agile frameworks give almost no guidance on how to handle these questions. What follows is an attempt to change that.

Actions to Take

I am going to present here 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 start. As you read through this, think about what other actions you might consider that might be specific to your team's context.

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 the extent that it stands out in the 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 of Brooks's Law[1]). This will often mean lowering WIP to help the aging item make progress. Team members who finish their items should be asked to help out with currently Aging work items instead of picking up new ones. Using this practice then, over time, you may even want to lower your WIP to below the number of members on the team.

This 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

Any work that is blocked or on hold is by definition 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

Read Don Reinertsen's "Principles of Product Development Flow"[2] 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 the 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 is 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."[3]

Still Valuable?

One final aspect to consider when "unsticking" work should be, "Is this item still valuable?" Maybe the reason a work item is Aging is because we are simply ignoring it. And maybe the reason we are ignoring it is that we no longer consider that item valuable. Obviously, all items have a perceived value when we start them (otherwise, why did you start them?) but it is always possible that while working on them we learn something that changes our minds. Maybe we can't get the feature to work the way we want it to. Maybe it does work but we understand the problem better now and need a different solution. Maybe it is technologically difficult or impossible to implement. Or maybe the business environment changed such that the item is no longer needed.

Whatever the reason, the second you decide that a work item is no longer valuable, then kick it out of your process. For me, that is one of the truest acts of agility. We've learned something and we took action. We don't fall victim to the sunk cost fallacy and continue to work on an item just because we started it. As any good product manager will tell you, there is nothing worse than investing money in the wrong thing. So if an item is not valuable, stop working on it!

As a quick aside, the above paragraph might seem to run afoul of LL's Assumption #2. Well done if you spotted that. In Section II, we will discuss how to "account" for items that need to exit our process abnormally so that we can still faithfully uphold all of the assumptions of LL.

Conclusion

If you are not paying attention to aging, you are missing the only real opportunity to achieve predictability.

The most basic way to use the Aging chart is to compare an item's current Age to how long it has taken us to complete items in the past. In other words, as items Age, we gain information about them. Are they taking too long? Are they spending too much time in one column? Have their chances of violating our SLE changed? This new information provides the actionable evidence we need to proactively manage for predictability. The percentiles from our Scatterplot work as perfect checkpoints to discuss if intervention is needed.

When it comes to interventions, consider breaking items up, pairing/swarming, unblocking blocked items, and removing items that are no longer valuable.

Once you start to pay attention to Work Item Age, and more importantly, once you start to take action on the information that aging gives you, then you will inevitably begin to complete items faster. Faster completion times will have the effect of lowering the percentiles on your Scatterplot over time. Lower percentile lines mean that we have smaller ranges of possible outcomes for the same percentile confidence. And that, if you recall, is exactly what we are after.

Key Learnings and Takeaways

- Use the Work Item Aging chart in your daily team meeting to quickly and in near-real time spot problems with flow.
- Compare the current Age of items with how long it has taken work items to complete in the past (Scatterplot percentiles).
- One area of focus is those items that have either breached their SLE or are about to breach their SLE.
- All other things being equal start with the oldest item first (regardless of column) and proceed to each next oldest item until you have a plan.
- Some actions you can take on aged items: break them up, pair/swarm on them, unblock them, or kick them out of the system if no longer valuable.

PART II: More Flow Principles for Predictability

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aamfp-10th.

Chapter 8 - Introduction to CFDs

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aamfp-10th.

What makes a CFD a CFD?

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aamfp-10th.

Conclusion

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aamfp-10th.

Key Learnings and Takeaways

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aamfp-10th.

Chapter 8a - Constructing a CFD

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aamfp-10th.

A Simple Example

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aamfp-10th.

What About Knowledge Work?

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aamfp-10th.

Conclusion

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aamfp-10th.

Key Learnings and Takeaways

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aamfp-10th.

Chapter 9 - Flow Metrics and CFDs

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aamfp-10th.

Work In Progress

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aamfp-10th.

Approximate Average Cycle Time

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aamfp-10th.

Average Throughput

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aamfp-10th.

Conclusion

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aamfp-10th.

Key Learnings and Takeaways

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aamfp-10th.

Chapter 10 - Interpreting CFDs

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aamfp-10th.

Mismatched Arrivals and Departures

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aamfp-10th.

Flat Lines

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aamfp-10th.

Stair Steps

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aamfp-10th.

Bulging Bands

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aamfp-10th.

Disappearing Bands

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aamfp-10th.

The S-Curve

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aamfp-10th.

A Boring CFD

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aamfp-10th.

Conclusion

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aamfp-10th.

Key Learnings and Takeaways

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aamfp-10th.

Chapter 11 - Conservation of Flow Part I

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aamfp-10th.

Arrivals and Departures Revisited

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aamfp-10th.

Arrivals and Departures on a CFD

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aamfp-10th.

Conclusion

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aamfp-10th.

Key Learnings and Takeaways

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aamfp-10th.

Chapter 12 - Conservation of Flow Part II

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aamfp-10th.

Just-in-time Prioritization

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aamfp-10th.

Just-in-time Commitment

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aamfp-10th.

Exceptions to Conservation of Flow

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aamfp-10th.

Conditioning Flow and Predictability

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aamfp-10th.

Conclusion

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aamfp-10th.

Key Learnings and Takeaways

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aamfp-10th.

Chapter 13 - Flow Debt

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aamfp-10th.

Approximate Average Greater Than Actual Average

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aamfp-10th.

Approximate Average is Less Than Actual Average

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aamfp-10th.

Approximate Average Roughly Equal to Actual Average

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aamfp-10th.

How Different is Different?

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aamfp-10th.

Conclusion

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aamfp-10th.

Key Learnings and Takeaways

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aamfp-10th.

Chapter 14 - Pull Policies

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aamfp-10th.

Class of Service

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aamfp-10th.

The Impact of Class of Service on Predictability

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aamfp-10th.

Slack

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aamfp-10th.

Conclusion

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aamfp-10th.

Key Learnings and Takeaways

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aamfp-10th.

PART III: Getting Started With Predictability

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aamfp-10th.

Chapter 15 - Getting Started

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aamfp-10th.

Defining Your Process

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aamfp-10th.

Capturing Data

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aamfp-10th.

How Much Data?

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aamfp-10th.

Create an Aging Chart

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aamfp-10th.

Create a Scatterplot

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aamfp-10th.

Some Pitfalls to Consider

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aamfp-10th.

Conclusion

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aamfp-10th.

Key Learnings and Takeaways

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aamfp-10th.

Chapter 16 - Next Steps

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aamfp-10th.

Forecasting

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aamfp-10th.

Forecasts for a Single Item

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aamfp-10th.

Forecasts for Multiple Items

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aamfp-10th.

Little's Law (Again)

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aamfp-10th.

Other Methods to be Wary About

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aamfp-10th.

Continue Learning

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aamfp-10th.

Appendices

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aamfp-10th.

Appendix A - Introduction to Little's Law

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aamfp-10th.

We Need a Little Help

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aamfp-10th.

A Different Perspective

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aamfp-10th.

It is all about the Assumptions

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aamfp-10th.

Assumptions as Process Policies

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aamfp-10th.

Segmenting WIP

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aamfp-10th.

Kanban Systems

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aamfp-10th.

Size Does Not Matter

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aamfp-10th.

Forecasting

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aamfp-10th.

Conclusion

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aamfp-10th.

Key Learnings and Takeaways

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aamfp-10th.

Appendix B - Interpreting Cycle Time Scatterplots

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aamfp-10th.

The Triangle

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aamfp-10th.

Clusters of Dots

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aamfp-10th.

Gaps

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aamfp-10th.

Internal and External Variability

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aamfp-10th.

Conclusion

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aamfp-10th.

Key Learnings and Takeaways

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aamfp-10th.

Appendix C - Cycle Time Histograms

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aamfp-10th.

What is a Histogram?

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aamfp-10th.

Constructing a Histogram

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aamfp-10th.

Conclusion

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aamfp-10th.

Key Learnings and Takeaways

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aamfp-10th.

Endnotes

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aamfp-10th.

Chapter 1

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aamfp-10th.

Chapter 2

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aamfp-10th.

Chapter 3

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aamfp-10th.

Chapter 3a

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aamfp-10th.

Chapter 4

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aamfp-10th.

Chapter 5

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aamfp-10th.

Chapter 6

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aamfp-10th.

Chapter 7

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aamfp-10th.

Chapter 8

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aamfp-10th.

Chapter 8a

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aamfp-10th.

Chapter 10

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aamfp-10th.

Chapter 11

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aamfp-10th.

Chapter 12

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aamfp-10th.

Chapter 13

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aamfp-10th.

Chapter 14

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aamfp-10th.

Chapter 15

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aamfp-10th.

Bibliography

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aamfp-10th.

Acknowledgements for the 10th Anniversary Edition

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aamfp-10th.

About The Author

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aamfp-10th.