Flow Metrics for Scrum Teams

Because story points were never a part of Scrum anyway

Daniel Vacanti and Will Seele

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

This version was published on 2022-08-10

ISBN 979-8-9867724-2-4

This is a Leanpub book. Leanpub empowers authors and publishers with the Lean Publishing process. Lean Publishing is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

© 2022 Daniel Vacanti and Will Seele
## CONTENTS

The Most Important Chart That You Have Never Heard Of (And How to Use It) .......................... 33
Conclusion .................................................................................................................. 35
Endnotes ....................................................................................................................... 36

Chapter 5 - Sprint Planning, Part I .......................... 37
What Can Be Done In This Sprint? .......................... 37
MCS in Sprint Planning .................................................. 39
The Language of Forecasting ........................................... 47
But What About Complexity? ........................................ 48
Endnotes ....................................................................................................................... 50

Chapter 6 - Sprint Planning, Part II .......................... 51
Right Sizing ......................................................................................... 51
Tooling ................................................................................................. 54
Drastically Reduce Planning Time ........................................ 54
Conclusion .................................................................................................................. 55
Endnotes ....................................................................................................................... 56

Chapter 7 - The Daily Scrum ........................................ 57
The SLE ......................................................................................... 58
Percentiles as Intervention Triggers ........................................ 60
What If PBIs Aren’t Flowing As Expected .......................... 62
A Daily Scrum Example ................................................................. 68
Conclusion .................................................................................................................. 70
Endnotes ....................................................................................................................... 71

Chapter 8 - The Sprint Review ........................................ 72
What Did We Learn Last Sprint? ........................................ 72
When Is It Done? ................................................................. 73
What Can Be Done? ............................................................ 75
Now What? ................................................................. 77
Conclusion .................................................................................................................. 77
Endnotes ....................................................................................................................... 78

Chapter 9 - The Sprint Retrospective ........................................ 79
## CONTENTS

The Triangle ................................. 80  
Clusters of Dots ............................. 81  
Gaps ........................................... 83  
Internal and External Variability ........ 84  
Conclusion ................................. 86  
Endnotes ................................... 86  

**Chapter 10 - Tooling** ...................... 87  
What To Look For In A Tool ................ 88  
Red Flags .................................. 92  
Conclusion ................................. 94  
Endnotes .................................. 94  

**Chapter 11 - Getting Started** ........... 95  
Starting Steps .............................. 95  
Conclusion ................................. 97  

Bibliography ............................... 98
Why This Book

This book came about from Slack discussions between Dan & Will about many of the things Dan disliked about Scrum. He still isn’t the greatest fan, but he’s improving. We sought to create a useful guide to more data-driven ways of managing work, and hopefully help some teams move away from Story Point theater. Mostly, we hope these tools will help your team spend less time planning and discussing plans, and more time building great things.

Thanks

The authors would like to thank James Scrimshire for the cover art. Also, thanks to our proofreaders Colleen Johnson, Stephan Vlieland, Stas Pavlov, Prateek Singh, and Frank Steeneveld for their feedback. We’ve tried to remove all the jokes about Jira. We still don’t like it.
Chapter 1 - Let’s Begin

Before you can even think about applying Flow Metrics to your Scrum implementation, there are a few things that you need to have in place. Unfortunately, the Scrum Guide is largely silent on these crucial pieces, so we’ll have to spend a bit of time explaining them in detail here. To make things a little more interesting, let’s turn the rest of this chapter into a drinking game. Every time we say “flow” or “workflow” from now on, take a drink. So go get your favourite tipple ready and let’s proceed.

What Is Flow?

The whole reason for the existence of your Scrum Team is to deliver value for your customers/stakeholders. Value, however, doesn’t just magically appear. Constant work must be done to turn potential product improvements into tangible customer value. The steps needed to turn a product improvement idea into something concrete that our customers find valuable is called a process.

You have chosen Scrum as the framework upon which to build your process for value delivery. It’s a common misunderstanding that Scrum itself is a process. It is not. It is a framework within which you create and continuously improve a value delivery process. Whether you know it or not, you and your team have built a value delivery process that goes way beyond Scrum. That process may be explicit or implicit, but it exists.

We cannot overstate the importance of this concept enough, because having an understanding of process is fundamental to the understanding of Flow. Once your process is established, then Flow
is simply defined as the movement of potential value through that process.

**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 the flow of work.

[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. -Dan]

The reason you should care about Flow is because your ability to achieve Flow in your process will dictate how effective, efficient, and predictable you are as a team at delivering customer value—which, as we stated at the beginning, is the whole reason you are here.

### Setting Up To Measure Flow

The teams that we see struggling with Scrum are not really struggling with Scrum. They are struggling with ownership or they are struggling with Flow. We can’t help you with the former, but we’re certainly able to help you with the latter. Thankfully, Flow comes with a set of basic metrics that give us tremendous insight into the health and performance of our process. But before we can talk about what metrics to use, we must first talk about what we need to be in place in order to calculate those metrics.

All metrics are measurements and all measurements have the same things in common: a start point and an end point. Flow is no different. Therefore, to measure flow, we must know what does it mean for work to have started in our process, and what does it mean for work to have finished in our process.
Getting started with Flow Metrics means first having a well-defined point at which work is started and having a well-defined point at which work is finished.

The decision around started and finished may seem trivial, but we can assure you it is not. Because this is where things in Scrum get much more nuanced (if not downright complicated) and where the Scrum Guide, as a guide to a framework and not a process, provides little to no actual guidance.

It is here that you might be thinking that in your context you already have an established workflow such as To Do -> Doing -> Done. You might further suggest that “started” simply means when an item moves from To Do to Doing and “finished” means when an item moves from Doing to Done. While technically not wrong, there are a few problems with this thinking.

First, usually the items that teams are tracking through a To Do -> Doing -> Done process are tasks. In other words, most teams will take a Product Backlog Item (PBI), and break it down into tasks like “analyze, build, test, check, fix”. They will then then track the progress of these tasks throughout the Sprint. This is potentially problematic because—as has been repeated several times now—Flow is specifically concerned with value delivery. But in Scrum value is defined at the PBI level, not at the task level. Which means to use Flow Metrics in our Scrum context, we must first define what “started” and “finished” means for a PBI.

[Note: Breaking down items into tasks comes from a good place, and can certainly be helpful for a brand new team to better understand the kind of work that’ll be required to get something done. In fact, most workflows can be derived from seeing what tasks are repeated with every PBI. But at some point this turned from being a good idea for some to the pervasive myth that it’s a mandatory part of “doing Scrum”. It isn’t. That’s what gives us the ability to use the Kanban strategy inside Scrum. -Will]

On the surface, it might seem like the latter choice—setting a
finished point—might be fairly straightforward in Scrum: just map your finished point to what your Definition of Done (DoD) requires. In other words, most of the time the finished point in your Scrum workflow would be any time you’ve added one or more PBIs to your Increment and that Increment has met your Definition of Done (DoD).

But what if your initial DoD does not include (a) that the Increment was shipped, (b) that once shipped value was validated, or (c) that the value feedback is incorporated into future decision making? And what if our customer wants answers to “when will it be done” in each of those cases? The Scrum solution would be to change your Scrum Team’s DoD so that it does include those extra criteria (remember your Scrum Team can always raise the quality outlined in the organization’s DoD—you just can’t lower it to below the organizational standard). However, organizational constraints may prevent you from changing your team’s DoD to be much broader in scope (e.g., your team is not allowed to deploy code to production). The beauty is that in this instance Scrum is not restrictive; rather, it is fairly flexible.

When using Scrum, there is nothing stopping you from defining a workflow where work finishes after your given DoD has been met. (This is where we get into the discussion of “Done” vs. “Done Done” vs. “Done Done Done” etc.) For example, in the above workflow (Options -> Discovery -> Building -> Validating -> Done), let’s say that (and we’re not saying this is a good idea) the Scrum Team is only involved in the “Building” step and that the DoD only addresses Increments that come out of the Building phase. If this were the case, then nowhere in Scrum does it say only measure what happens in Building. Nothing is stopping you from measuring through to Done, or through to Validating, or just through Building.
For those people familiar with the Kanban concept of Definition of Workflow (DoW), what we’re saying here is that your DoW should cover your DoD but may, and probably will, extend beyond it.

The same is true of choosing a started point. There is nothing stopping you from choosing a started point where work begins before the Scrum Team actually gets involved. This should be very good news to you, because when does a Scrum Team actually get involved with work? Does work only start when development begins within the Sprint? Does work start when a PBI has been selected for the Sprint Backlog? Does work start when refinement begins on a PBI? Does work start before refinement? We think those are all great questions and different contexts may have very different answers—none of which are necessarily right or wrong. The beauty of Scrum is that it puts no restriction on what you choose as that started point.

It gets a little more complicated than that because it is perfectly allowed in Flow to have more than one started point and more than one finished point within a given workflow. In the above example maybe we want to measure started from both Discovery and Building and finished from both Validating and Done. Why we might want to do that will depend on what questions you want to answer based on the metrics you collect. For example, maybe we want to know both how long it takes items to get done from when we first start working on them (Discovery to Done) as well as how long our Building step takes (Building to Validating). Any and all of these permutations are allowed.

Not only are the different permutations allowed, it is encouraged that you experiment with different started and finished points in your process to better understand your context. You will quickly learn that changing the definition of started/finished will allow you to answer very different questions about flow in your process. If all goes well, expanding your started/finished points will get you down the path toward true business agility.
Conclusion

Your job as a Scrum team is to deliver value to your customer(s). By choosing Scrum, you have selected a very specific framework upon which to implement a process for value delivery.

If you haven’t already, you need to sit down with your team and decide–for your process–what does it mean for a PBI to have started and what does it mean for a PBI to have finished. All other flow conversations will be dependent on your boundary decision.

Once defined, the movement of potential value from your defined started and finished points is what is called Flow. The concept of movement is of crucial importance because the last thing we want as a team is to start a whole bunch of PBIs but never finish them. That is antithesis of value delivery. What’s more, as we do our work our customers are constantly going to be asking (whether we like it or not) questions like “how long” or “how many”–questions which require our understanding of movement to answer.

That’s where Flow Metrics come in.
Chapter 2 - The Basic Metrics of Flow

The four measures to track for Flow are:

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

In last chapter, we spent a lot of time talking about the importance of having a well defined point at which work is started and a well defined point at which work is finished. The reason for this importance is that all basic metrics of flow are defined in terms of those started and finished points. Again, Scrum itself does not require these started/finished points to be formally identified beyond the Sprint boundaries, but if you want to use Flow Metrics to a more granular level in a Scrum context you must go through the exercise of defining those points in your process. We will assume that your team has these defined for our discussion going forward.

One final word of caution: these four measures represent only the minimum required to gain insight about flow in Scrum. Teams and organizations may and often should use additional context-specific measures that assist in making data-informed decisions¹. Feel free to add those additional measures to your context as you see fit—as long as they are not story points. Yes, that was (mostly) sarcasm.
Work In Progress

The generic definition for WIP in a given flow context is: All discrete units of potential customer value that have entered a given process but have not exited. In a Scrum context, the discrete units of work are called Product Backlog Items (PBIs); so, to calculate WIP you simply count the PBIs within your process boundaries as defined above. That’s it: just count PBIs and you will have calculated WIP.

Your first question might be, “how does complexity fit into the WIP calculation?” The short answer is it doesn’t. This is probably the hardest concept to grasp for people who have been taught that capacity is a function of PBI complexity. It isn’t. There is nothing in the principles of flow that require you to understand the relative complexity of items that are moving through your process. We will explain why this is in the next chapter, but for now we’re going to ask you to suspend disbelief and just accept that as true. The very good news is that if you hate estimating in story points, then you can drop that practice immediately upon the adoption of Flow Metrics. But more on that a little later.

Your next objection might be, “well if complexity doesn’t matter then certainly the size of the PBIs does” After all, the PBIs that come through your process will be of a wide variety of size. How can you possibly account for all of that variability and come up with a predictable system by just counting PBIs? While that is a reasonable question, it is not something to get hung up on. As with complexity there is no requirement to do any kind of upfront estimation of size when practicing flow, beyond having a very short conversation about the Service Level Expectation (explained in Chapter 6) when you pull an item. But more on this later, when we talk about Sprint Planning.

If you happen to already be using Kanban in Scrum context, then it should also be noted that there is a difference between WIP and
WIP limits. You cannot calculate WIP simply by adding up all the WIP limits on your board. It should work that way, but in reality it does not. This result should be obvious as most Kanban boards do not always have columns or boards that are at their full WIP limit. A more common situation is to have a Kanban board with WIP limit violations in multiple columns—or across the whole board. In either of those cases simply adding up WIP limits will not give you an accurate WIP calculation. The sad truth is there is no getting around actually counting up the physical number of items in progress to come up with your total WIP.

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

Cycle Time

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

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

This definition is based on one offered by Hopp and Spearman in their Factory Physics book² and, you will note, agrees exactly with the definition given at the beginning of this chapter. Defining Cycle Time in terms of WIP removes much—if not all—of the arbitrariness of some of the other explanations of Cycle Time that you may have seen (and been confused by) and gives us a tighter definition to start measuring this metric. The moral of this story is: you essentially have control over when something is counted as Work In Progress in your process. Take some time to define those policies around
what it means for an item to be “Work In Progress” in your system and start and stop your Cycle Time clock accordingly.

You will also not want to overlook the emphasis on “elapsed time”. The use of elapsed time is probably very different from the guidance you have previously been given. Most other methodologies ask you to measure only the actual amount of time spent actively working on a given item (if they ask you to measure time at all). This is sometimes called Touch Time. We happen to think this guidance is wrong. And we have a couple of reasons why.

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

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

There is still a more important reason to understand Cycle Time. Cycle Time is also the amount of time it takes to get feedback on what you deliver in your workflow. Ideally, Cycle Time represents the amount of time it takes to get customer feedback, if Done means Released in your workflow. 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. If Done does not mean Released in your workflow, you will still be able to get feedback from the next step downstream. In both of these cases, shortening Cycle Time will shorten the (ideally customer) feedback loop. And to shorten Cycle Time, you are going to first need to measure it.

**Work Item Age**

Age is by far the most important of all the Flow Metrics to track. This is the reason why it’s covered in great detail in Chapter 4 of this book (which may be why you started reading this in the first place). The definition of Work Item Age is:

**Work Item Age**: the total amount of time that has elapsed since an item entered a workflow.

In some literature you will see Work Item Age referred to as “WIP Age”. We will try to be consistent in this book and only use the term Work Item Age, but please know that both names can be used interchangeably.

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

**Throughput**

We’ve saved the easiest metric to define for last. Simply put, Throughput is defined as:

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

A further thing to know about Throughput is that many agile coaches and consultants use the words “Velocity” and “Throughput” interchangeably. While Velocity can be defined in terms that are the same as Throughput, most of the time when a coach says “Velocity” they mean “story points per sprint”. When defined in terms of story points, you should know that Throughput and Velocity are anything but synonymous.

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

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

The Throughput metric answers the very important question of “How many PBIs 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.
Chapter 2 - The Basic Metrics of Flow

Data Collection and Calculation

Metric definitions are all very well and good but are meaningless if you don’t know what data to collect or how to calculate each metric from that data.

In terms of data collection, this is where we harping on you to define started and finished points will finally pay off. Take a timestamp when a PBI crosses your started point and take another timestamp when that same PBI crosses your finished point. Do that for every PBI that flows through your process as shown in Figure X (forgive the American-style dates):

<table>
<thead>
<tr>
<th>PBI ID</th>
<th>Started</th>
<th>Finished</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>01/01/2022</td>
<td>01/03/2022</td>
</tr>
<tr>
<td>2</td>
<td>02/02/2022</td>
<td>03/03/2022</td>
</tr>
<tr>
<td>3</td>
<td>01/02/2022</td>
<td>03/04/2022</td>
</tr>
<tr>
<td>4</td>
<td>01/03/2022</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>01/04/2022</td>
<td></td>
</tr>
</tbody>
</table>

Figure 2.1: Timestamp Data

That’s it. To calculate all the basic Flow Metrics, this is the only data you will need. Even better, if you are using some type of PBI tracking tool to help your team, then most likely your tool will already be collecting all of this data for you.

The downside of using a tracking tool, though, is that you may not be able to rely on any out-of-the box metrics calculations that it may give you. It is one of the great secrets of the universe as to why many Agile tools cannot calculate Flow Metrics properly, but, for the most part, they cannot. To properly calculate each of the metrics from the data do as follows:

WIP

WIP is the count of all PBIs that have a started timestamp but not a finished timestamp for a given time period. 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 PBIs 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 would want to know what our WIP was for a given day, we would just count all the PBIs that had started but not finished by that date. For Figure 2.1, our WIP on January 5th is 3 (PBIs 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 is more relevant for your specific situation. Or hours. Or even Sprints. For your Scrum team, if you wanted to measure Cycle Time in terms of Sprints,
then the calculation would just be Finished Sprint – Start Sprint + 1 (assuming PBIs cross Sprint boundaries in your context). The point here is that this calculation applies in all contexts. However, as with WIP, our 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 (our guess is you won’t).

**Work Item Age**

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

The “plus one” argument is the same as for Cycle Time above. Our apologies, but you will never have a PBI that has an Age of zero days. Again, our strong recommendation is to track Age in days.

**Throughput**

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

<table>
<thead>
<tr>
<th>Work Item Id</th>
<th>Arrived</th>
<th>Departed</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>01/01/2022</td>
<td>03/01/2022</td>
</tr>
<tr>
<td>2</td>
<td>01/02/2022</td>
<td>03/02/2022</td>
</tr>
<tr>
<td>3</td>
<td>02/02/2022</td>
<td>03/03/2022</td>
</tr>
<tr>
<td>4</td>
<td>01/02/2022</td>
<td>03/04/2022</td>
</tr>
<tr>
<td>5</td>
<td>03/02/2022</td>
<td>03/04/2022</td>
</tr>
</tbody>
</table>

**Figure 2.2 - Sample Process Data**

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

<table>
<thead>
<tr>
<th>Completed Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>03/01/2022</td>
</tr>
<tr>
<td>03/02/2022</td>
</tr>
<tr>
<td>03/03/2022</td>
</tr>
<tr>
<td>03/04/2022</td>
</tr>
</tbody>
</table>
Figure 2.3 - 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 our data, those counts look like this:

<table>
<thead>
<tr>
<th>Completed Date</th>
<th>Throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td>03/01/2022</td>
<td>1</td>
</tr>
<tr>
<td>03/02/2022</td>
<td>0</td>
</tr>
<tr>
<td>03/03/2022</td>
<td>2</td>
</tr>
<tr>
<td>03/04/2022</td>
<td>2</td>
</tr>
</tbody>
</table>

Figure 2.4 - Calculated Throughput

From Figure 2.4 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. In Scrum, your first inclination might be to calculate Throughput per Sprint: “we got 14 PBIs done in the last Sprint”. Let us very strongly advise against that and advise very strongly that you measure Throughput in terms of days. Again, it would be a book in itself to explain why, but let us just offer 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 Chapter 5); 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. You can easily derive Throughput per Sprint from that data if it still matters to you.
Randomness

We’ve saved the most difficult part for last. You now know how to calculate the four basic metrics of flow at the individual PBI level. Further, we now know that all of these calculations are deterministic. That is, if we start a PBI on Monday and finish it a few days later on Thursday, then we know that the PBI 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 Scrum Team’s Throughput is? How do we answer those questions?

Our guess is you immediately see the problem here. If, say, we look at our team’s Cycle Time for the past six Sprints, we will see that we had PBIs 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. Thankfully for us, probabilistic thinking is
the topic of the next chapter.

**Conclusion**

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

**Endnotes**

Chapter 3 - Probabilistic Thinking

For those of you who have read Dan’s “When Will It Be Done?”¹ book, this chapter can be reasonably skipped without loss of continuity. For the rest of you, please read on as the following concepts are foundational to getting the most out of Scrum using Flow Metrics.

The End of Determinism

For those of you who still commute in the post-pandemic world, how long will it take you to get to work tomorrow morning? This seems like straightforward question. Perhaps you immediately thought of a single value, say, 25 minutes. But think about your answer for a second. Will it really take you exactly 25 minutes? Our guess is your true answer is “it depends”.

If you drive to work, what are some factors that might affect your drive time? Traffic, road work, accidents, departure time, can all make a difference. If you take the train to work, your travel time might depend on how busy the trains are, whether there is work being conducted on the lines, or the (maybe not so) occasional industrial action.

Suppose we were to begin timing your commute to work every day. If we define the start of your commute as the moment when you step out the door of your home, and we define the end of your commute as the moment you step into the door of your workplace (where have we seen well defined start/finish points before?), then
we simply need to collect the timestamps at those two points and build a table like the one shown below:

<table>
<thead>
<tr>
<th>Day</th>
<th>Start Time</th>
<th>End Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>07:17</td>
<td>07:43</td>
</tr>
<tr>
<td>2</td>
<td>07:35</td>
<td>08:03</td>
</tr>
<tr>
<td>3</td>
<td>07:22</td>
<td>07:44</td>
</tr>
<tr>
<td>4</td>
<td>07:44</td>
<td>08:58</td>
</tr>
<tr>
<td>5</td>
<td>07:12</td>
<td>07:37</td>
</tr>
</tbody>
</table>

**Figure 3.1 - Sample Commute Data**

Given the example data in Figure 2.1, how long would it take for this person to get to work tomorrow? Take a moment to come up with your answer. When you have it, let me take a guess at how you came up with your answer. Did you take End Time and subtract Start Time to come up with a total elapsed time for each day? For example, on Day 1 did you take 07:43 minus 07:17 to come up with a travel time of 26 minutes? Good. Once you had all of the elapsed times, did you then average them to come up with your daily commute time? Not so good. If you were mathematically savvy, maybe you even calculated the standard deviation to come up with a confidence level? That is even less good.

Reducing a forecast to a single number is questionable, and having that single number take the form of an average usually makes things even worse (more on averages below).

Why?

The question “how long will it take me to commute to work tomorrow?” is essentially asking you to make a forecast. In other words, you are being asked to predict the future. But the future is full of uncertainty. As we just discussed, we don’t know if our commute will be affected by traffic, interrupted by an accident, or if nothing extraordinary will happen. Your actual commute time will be severely influenced should any one or more of these factors occur. Another way of thinking about this is to consider that before you step out of your door in the morning, there are many possible
futures with respect to your commute—each with its own chance of occurring.

The good news is that we have a branch of science that deals specifically with uncertainty (situations where multiple possible future outcomes exist). That science is called probability. Whenever many possible outcomes are involved, one must take a probabilistic approach—as opposed to a deterministic one. To me, the essence of probabilistic thinking can be summed up in one, concise statement:

“To think probabilistically means to acknowledge that there is more than one possible future outcome.”

A simple way to explain this type of thinking is to consider rolling two fair, six-sided dice. It is impossible to predict with 100% certainty the outcome of any given roll before the dice have been cast. That does not mean, however, that the results of rolling two dice are completely unpredictable. Actually, there are several things that we know about each roll:

- There is a 0% chance of rolling 1 or 13
- The possible outcomes are any whole number between 2 and 12 (inclusive)
- The most likely outcome of any given roll is 7
- There is a specific probability associated with rolling exactly 7. In fact, the chance of getting exactly 7 is—well, do you know the chance of this happening? The answer will be given below.

Acknowledging that the future must be described in terms much like these is precisely what we are going to need to do in order to provide more practical forecasts.
The Flaw of Averages

Maybe your answer to my commute question was something more like “On average it takes me 25 minutes to get to work”. As stated above, you may have calculated an average for the commute data to answer how long it will take. Anytime you communicate a forecast in terms of an average, you have fallen victim to the Flaw of Averages (FoA).

The Flaw of Averages (FoA)² is a concept detailed in the book of the same name by Dr. Sam Savage and is very closely related to probabilistic thinking. 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 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 Bill Gates or Jeff Bezos walk 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. Forecasting isn’t one of those.

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, Dan currently lives in South Florida and as you may know Florida is fairly prone to being hit by 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 about 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 illustrate, if you were to roll a single, fair, six-sided die, the expected outcome (or, more crudely, average outcome) is actually 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 from before?) 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 actually 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? Unfortunately, that’s exactly what many Scrum teams do.

**Conclusion**

As we will see in the coming chapters, the true power of Flow Metrics for Scrum Teams comes in acknowledging their probabilistic nature. Not every PBI that flows through your process will take exactly the same amount of time; not every Sprint will you get exactly the same number of PBIs to done, etc.

Embracing probabilistic thinking will allow you to make more accurate plans and will ultimately allow you to be more predictable for customers/stakeholders. But what are we thinking probabilistically about and what does probabilistic thinking have to do with Flow Metrics? Funny you should ask...
Endnotes

Chapter 4 - Two Charts

Before we go much further, we need to explore how to visualize two of the four Flow Metrics, Cycle Time and Work Item Age. Cycle Time will be displayed in a chart known as a Scatterplot and Work Item Age will be displayed on a (surprise) Work Item Aging Chart. These two charts will be useful to us as we explore how to use Flow Metrics and Scrum Events in the next section. If you are familiar with Cycle Time Scatterplots and Work Item Aging Chart, then this chapter can be reasonably skipped without loss of continuity. If you have never heard of those charts or if you want to brush up on your understanding, then please read on.

The Cycle Time Scatterplot

If you have never seen a Cycle Time Scatterplot before, then here is your chance:
As you can see from Figure 4.1, across the bottom (the X-axis) is some representation of the progression of time. The X-axis represents a timeline for your process. You will notice that Figure 4.1 shows the timeline progression from left to right. This is not a requirement, it is only a preference. However, all Cycle Time Scatterplot examples in this book will show a time progression 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: days, weeks, months, etc.

To generate a Scatterplot, any time a PBI completes, you find the date that it completed across the bottom and plot a dot on the chart at a height according to its Cycle Time. For example, let’s say a work item took seven days to complete and it finished January 1,
2022. On the Scatterplot, you would go across the bottom to find January 1, 2022 and then go up and put it a dot at seven days.

Note that you could have several items that finish on 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.

Over time as you plot more and more PBIs complete, a random set of dots will emerge on your chart. Based on the discussion in the last chapter, this randomness is to be expected given the uncertainty that our teams encounter as we do work. The randomness of the dots is simply an expression of different risks being realized at different time for different PBIs. Given that, an approach we can take is to segment the chart to gain a greater understanding of the risk associated with certain outcomes.

[Note: There’s a fun Retrospective exercise you can do with your team here if you’re currently also working with story points. Try to find a relation between the estimated story points and the actual Cycle Time of PBIs. Your team might find it very hard to do so. Commence the discussion. -Will]

**Percentile Lines on a Scatterplot**

One way to segment risk is to draw percentile lines on our chart. A percentile line is a horizontal line for which that percent of dots fall on or below the line. This calculation is shown in Figure 4.2 below.
Figure 4.2 - The 50th Percentile Line Added to a Scatterplot

In Figure 4.2, the 50th percentile line occurs at eight days. That means that 50% of the PBIs that have flowed through our process took eight days or less to complete. So we can say that when a PBI enters our process it has a 50% chance of finishing in eight days or less. That is without doing any estimation! (More on this concept in a later chapter).

Using this same approach, we can calculate any percentile. A commonly used percentile is the 85th. Again, this line represents the amount of time it took for 85% of our work items to finish. In Figure 4.3 below you can see that the 85th percentile line occurs at 15 days. That means that 85% of the dots on our chart are on or below that line, and 15% of the dots on our chart are above that line. This percentile line tells us is that when a work item enters our process it has an 85% chance of finishing in 15 days or less, again, with no
estimation.

The 50\textsuperscript{th}, 85\textsuperscript{th}, and 95\textsuperscript{th} percentiles are probably the most popular “standard” percentiles to draw. You may see other percentiles, though, so we have included Figure 4.4 with a few more.
At the risk of repeating ourselves, these percentiles act as one way of segmenting the uncertainty in our process. Having this segmentation will be required when we talk about how to leverage a Service Level Expectation (SLE) in our Scrum Events—but that discussion will have to wait for the next section.

Be mindful of the language here. A SLE is not a SLA: an Expectation is not an Agreement, and thus shouldn’t be treated as such. Be sure to explain the difference when people confuse the two. Additionally, many teams will skip the “or less” part when talking about the percentiles. This can cause misunderstanding within the team and with stakeholders. You’re going to need to be a bit pedantic about this until it’s common knowledge.

**Why Percentiles Are Preferred**

There are at least three reasons why we like the percentile line
approach to segmenting Scatterplot data. First, notice that when we described how to draw the standard percentile lines on a Scatterplot, we 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, you do not need to know how your data is distributed. These percentile line calculations work independent of any underlying distribution.

Second, percentiles are extremely easy to calculate. You simply count up all the dots and multiply by percentages. No advanced degree in statistics is required!

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. As we said before: “If Bill Gates walks into a bar, then on average everyone in the bar is a billionaire”. Obviously average is no longer a useful statistic in that case. The same type of extreme outlier phenomenon happens in our world. However, when you do get those extreme Cycle Time outliers, your percentile lines will not budge all that much. This robustness in the face of outliers is a strong advantage for percentile lines in Cycle Time analysis.

The Most Important Chart That You Have Never Heard Of (And How to Use It)

The problem with a Cycle Time Scatterplot –if there is one– is that by definition, a dot does not show up until a PBI has finished. However, if a PBI is taking too long to finish then waiting until it has finished to do something about it is too late. What we need, therefore, is a view into our process data while PBIs are still in
progress. We will call this view an Aging Work In Progress chart (or “Aging Chart” for short).

To understand an Aging chart, let’s consider a process workflow that looks like this Analysis Active -> Analysis Done -> Dev Active -> Dev Done -> Testing -> Done where “Analysis Active” is our started point and “Done” is our finished “point”. Note that we’re not saying this is a good workflow, we’re just using this for illustrative purposes. Hopefully you will be much more creative when mapping out your workflow.

An example of an Aging Work In Progress chart for this particular workflow might look like Figure 4.5:

![Figure 4.5 - Aging Work In Progress Chart for Sample Workflow](image)

Before we get into how this chart should be used, let’s quickly go over the anatomy of the chart so you know what you are looking at. Unlike the Scatterplot, you can see the whole chart has been
segmented into columns to match your process’s workflow. Up the side of the chart is the Age of PBIs.

Age in this context is different from Cycle Time on the Scatterplot. What we are representing on this chart (Figure 4.5) 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 the Aging Chart you can use whatever time units you want: days, weeks, months, sprints, etc. Thus, every dot on the chart represents a PBI 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 (recall that you should have tracked the timestamp for when the item entered your process!).

You will also see percentile lines on the Aging Chart. These percentile lines are exactly the same lines that we calculated for our Scatterplot. You overlay percentile lines on the Aging Chart so that you can see how current PBIs in progress are flowing as compared to the total amount of time it took previous items to complete.

Being able to compare items’ current Age to our past flow performance will help tremendously during our Daily Scrum. As this is the second time that we’ve mentioned Scrum events in this chapter, then maybe it is time to move on to a deeper discussion of those.

## Conclusion

Tracking metrics on their own is of little use unless we have an effective way to display that data. The Cycle Time Scatterplot and WIP Aging Chart are two of a myriad of analytical graphs that we can employ for data-informed decisions. Its just that these two will be most helpful as we talk about introducing Flow Metrics into the Scrum events. For a more detailed discussion of these two
charts as well as for a fuller discussion of other useful flow-based charts, please see Dan’s two books, “Actionable Agile Metrics for Predictability”¹ and “When Will It Be Done?”².

Endnotes

Chapter 5 - Sprint Planning, Part I

The 2020 version of the Scrum Guide¹ lists three topics that need to be covered in the Sprint Planning event:

- Why is this Sprint valuable?
- What can be Done in this Sprint?
- How will the chosen work get done?

Flow Metrics can help with answering one of these questions, but we’re not going to tell you which one.

By the way, this is a doozy of a chapter, so you’ll maybe want to go make yourself a cuppa before you settle in.

What Can Be Done In This Sprint?

“Selecting how much can be completed within a Sprint may be challenging. However, the more the Developers know about their past performance, their upcoming capacity, and their Definition of Done, the more confident they will be in their Sprint forecasts.”¹

As Scrum is only a framework, the Guide is purposefully vague on how Sprint Planning should actually be accomplished. And as most who are reading this manuscript know, much has been published on proper techniques to plan a Sprint. Further, many of you also know Dan’s thoughts on the established orthodoxy. Rather than railing against a bunch of poor practices that aren’t part of Scrum anyway, we’d like to focus our time here on a detailed discussion of how Flow Metrics can help you to accurately plan a Sprint.
A Quick Thought Experiment Before We Begin

Let’s say you have just started a project and in two weeks you have finished ten items (at this point it doesn’t matter if you are doing Scrum or not). Let’s further say that there are fifty items remaining in the project, some of which are in progress. When your customer asks, “When will the project be done?”, how would you calculate how long it will take to complete those fifty items?

“Easy,” you say, “since we have completed ten items in two weeks that means that we get five items done per week. At a rate of five items per week it will take us ten weeks to finish the remaining 50 items. So our answer is ten weeks.”

In reviewing your response, we would like to suggest that there was one thing that was correct in your answer, and one thing that was wrong. Very wrong. The correct part of your answer was the metric you used to calculate your projection. You calculated the rate at which you were getting items done –five per week. You used the Throughput metric we explained in Chapter 2.

As we explained there, Throughput is a measure of how fast items depart a process. Stated slightly differently, Throughput is the number of work items that are completed per unit of time. The unit of time can be any unit that is useful to you–you can measure items done per day, per week, per Sprint, etc.

The thing you did wrong in your calculation, however, was to simply divide the remaining work (fifty items) by the Throughput (five items per week) to get your forecast (10 weeks). This type of thinking is a classic example of the Flaw of Averages¹ (FoA just sneaks up on you, doesn’t it?).

Using an average completion rate (Throughput) to project the completion date of multiple items in a backlog suffers from the same fatal flaw as the average commute time example from the last chapter. That fatal flaw is trying to reduce uncertainty down to a single number. In any planning activity, there is more than one possible future outcome, so we must think probabilistically.
So how do we take a probabilistic approach to Sprint Planning?

**Monte Carlo Simulation**

What is the probability of you flipping a fair coin and it coming tails? How do you know? Let’s say you are really bad at math like Will and have no clue whatsoever on how to model this problem mathematically. What would you do then? One thing you could try is just to flip the coin over and over and over again and track the results. The number of times the flips came up tails divided by the total number of times you flipped the coin is your probability. Now, to get an accurate answer you may have to flip the coin hundreds if not thousands of times, but over time if you track enough flips your results will converge on the correct answer.

Congratulations. By flipping a coin over and over and over again, you have just performed a Monte Carlo Simulation (MCS).

The Monte Carlo Method (and MCS specifically) is a statistical sampling technique where a simulation is used to predict the probability of different outcomes given the input of random variables. That definition is a mouthful and doesn’t really say much so we’ll do a deeper dive in the next section. For now, just know that any time you are faced with a problem that can’t be modeled mathematically, or would be too hard to compute if you could model it correctly, or would take too long to compute even if an answer existed, then you have a good candidate problem to be solved using MCS. Spoiler alert: Sprint Planning is a good candidate for MCS.

### MCS in Sprint Planning

From the Scrum Guide regarding Sprint Planning, “...the Developers select items from the Product Backlog to include in the current Sprint”.¹ When the Scrum Team has agreed to a (provisional) Sprint Goal, they can then look at what PBIs would help reach this Goal. This is an iterative process, as they might find they have more
capacity (meaning they could have a more ambitious Sprint Goal), or the Goal is too ambitious for the amount of PBIs they’re able to pull. In other words, once a candidate Sprint Goal is found, the Developers are being asked to forecast how many PBIs they will get done in this Sprint. Given all of the possible variables associated with forecasting something like that (dependencies, interrupts, pandemics, etc.), there is no way you could reasonably come up with an equation (like \( E=mc^2 \)) that would adequately model a solution to this problem.

[Note: Don’t let me catch you trying to fill up your Sprint without identifying a Sprint Goal. The Goal always supersedes whatever scope you’ve pulled, so you have the flexibility required to deal with complexity. Probabilistic forecasting doesn’t negate this. - Will]

But what if there was a way that we could look at how many items we completed in the past and used that as a guide for how many items we could complete in the future? And what if that model could incorporate the risk associated with those future outcomes? Enter our friend Monte Carlo Simulation.

Monte Carlo methods vary, but in general, they tend to follow a typical pattern:

1. Define a probability distribution of possible inputs
2. Randomly select values from the input distribution and perform a computation on the selected inputs
3. Aggregate the results (usually by employing a Results Histogram)
4. Repeat steps 2-3 an arbitrary number of times until you have a clear picture of what the result set looks like

In Sprint Planning, this looks like:

**Step 1: Define a probability distribution of possible inputs.** Whenever (in an Agile context, anyway) you hear a question
framed as “how many items...” you should immediately think “Throughput”. As with the earlier project thought experiment, for this problem we need a rate metric for forecasting and the Flow rate metric is Throughput. The probability distribution is simply the set of historical daily completion rates of PBIs. Recall from Chapter 2 that Throughput is a random variable and its distribution is given by the historical completion rates of PBIs for our Scrum Team.

Does that mean we take all of the historical Throughput data that we have and feed that whole dataset into our MCS? Well, possibly, but not quite. The dirty little secret of MCS is that we are assuming that the future we are modeling roughly looks like the past we have data for. The beauty of using professional Scrum is that in most contexts this is a safe assumption. But it does still mean that we will have to make an educated guess about what subset of historical data will help us with our planning. This practice is definitely more art than science, but we do have a couple of heuristics to help us:

1. More recent data is usually better than less recent (older) data. It’s possible that you’ve heard that “more data is better than less data”. That statement is not necessarily true because of this recency principle. Let’s say you had Throughput data on your Scrum Team going back 10 years. If more data is better than less data does that mean you should use all 10 years’ worth of Throughput for your simulation? Obviously not. But how much recent data do I need to perform a successful simulation? There is no exact answer to that question, but the general rule of thumb is a minimum of 10-12 data points and ideally as much as 20 points as a comfortable minimum. Before you get too scared, that doesn’t mean you need to run 20 Sprints before you have enough data to perform an MCS. Remember from Chapter 2 that we recommend using days as your Throughput unit of time. That means if you are doing two week Sprints then in two Sprints you should have more than enough data to get started. (Two Sprints of two weeks
each would be 28 days of Throughput data. Yes, weekends count—Google it!

2. But more recent data is no panacea, either. Let’s say you are a Western European Scrum team that has a Sprint beginning on January 7. Would it make sense to use the last Sprint’s data to plan the upcoming Sprint. Again, obviously not because of the Christmas and New Year’s holidays. In America, that problem gets even more challenging because you can’t necessarily use November data either due to the Thanksgiving holiday. It is thus up to you to decide what subset of historical data is most reflective of the future you planning for. The advantage of MCS is that it is such a quick and easy tool to use that you could realistically play around with multiple inputs and see what the results are (but more on that later).

**Step 2: Randomly select values from the input distribution and perform a computation on the selected inputs.** This is where the MCS rubber meets the road. To project out how many PBIs will finish, we are going “simulate” how many items get done for each consecutive day into the future of the upcoming Sprint. The way that works is, say, for example, we are starting our simulation on January 1. For January 1, then, we are going to select a random Throughput number from our historical (input) Throughput data. Let’s say we randomly chose a 3. For the purposes of the simulation, that means that our team got 3 PBIs done on January 1. We then repeat the same calculation for the next day, January 2. Let’s say we randomly choose a 5 for January 2. That means we now have a total of eight PBIs done so far this Sprint. Do that for the total number of days in the Sprint, add up the results and that is **one possible outcome** for the total number of PBIs to be completed in the next Sprint (this is commonly referred to as one “run” of the simulation).

[Note: The algorithm that we used here where we simply randomly select a historical Throughput input is just one such option for our
Chapter 5 - Sprint Planning, Part I

simulation. There are many others. For more information, see Dan’s book “When Will It Be Done?”

Step 3: Aggregate the results. For each “run” of the simulation, we are going to track the results in what we are going to call a Results Histogram. What the Results Histogram is and how to interpret it will be discussed shortly.

Step 4: Repeat steps 1-3 an arbitrary number of times until you have a clear picture of what the result set looks like. With each run of the simulation, the probability distribution as described by the Results Histogram will become clearer and clearer. For example, think of the coin flipping example. Do you think you would have a better idea of probability of getting tails if you flipped two times or two thousand times? Even so, MCS will converge on a result fairly quickly so all more runs will do is get you a better looking Results Histogram. For most Sprint Planning, we’ve found 1,000-2,000 runs gives pretty good results. Anything much more than 10,000 doesn’t usually provide much better quality. Try this for yourself to see it in action.

Interpreting the Results Histogram
As was just stated, the outcome of your Sprint Planning MCS will be a Results Histogram. This histogram represents the shape of your risk as it relates to the multiple possible future outcomes; i.e., the different chances of you getting different numbers of PBIs to Done. [An interesting aside is that the Results Histogram of a MCS will usually approximate a Normal Distribution. This is due to something in probability called the Central Limit Theorem (CLT). You need not worry yourself with the specifics of the CLT, only know that in general the Results Histogram will usually resemble a bell curve.]

So how do we turn our Results Histogram into an answer to “What can be done this Sprint?”

Since the Results Histogram represents the shape of your risk associated with possible outcomes, the first thing we need to do
is segment that risk into what are acceptable outcomes vs. what are not acceptable outcomes. We can do this by drawing what are called percentile lines on the histogram.

Before we begin, let us first point out an important principle of the Histogram that we just generated. If you sum the heights of all the bars on the chart together you will get the number of runs that made up the simulation. For example, if we ran 10000 runs in our simulation, then adding up the heights of the bars in the resultant chart would give you 10000. The probability of any one particular outcome, then, is the height of a given bar divided by the total number of runs.

To illustrate, take a look at the Results Histogram in Figure 5.1 below.

Figure 5.1 is the result of running an MCS to forecast a two week (14 day) Sprint. In this example, the MCS was run 10,000 times to get the results you see here. Focus your attention on the tallest bar in the middle of the chart. Because it is the tallest bar of the histogram, that means that that outcome occurred the most of any of the outcomes of the simulation. In other words, that’s our most likely outcome. It’s a little hard to see from the image, but that bar occurs at 18 items. The height of that particular bar is 656. That means that this Scrum Team has a 6.56% (656/10,000) chance of getting exactly 18 PBIs to done in the next Sprint. That’s not very good, is it? (Do you remember that the most likely outcome is not
very likely?)

But there is another way to interpret these results. For the sake of argument, let’s say that we did plan on getting 18 items completed in this Sprint. If we got exactly 18 items done, then that would be a good result. But what if we got 19 items done? You see from the histogram that there is indeed a chance that we could finish 19 items. If we told our Product Owner we could get 18 items done, but we actually finished 19 items, how do you think they would feel? My guess is they would consider that a good result. The same would be true if we got 20 items done, 21, 22, and so on...In fact, any outcome to the right of 18 (inclusive) would be considered acceptable as illustrated in the green shaded area of Figure 5.2 below:

![Figure 5.2: 18 Or More Items Complete](image)

To get the probability of completing 18 or more items done, simply add up the heights of all the bars in the green shaded area and divide by 10,000. In this example, the heights of those bars add up to 5,000, so there is a 50% chance of our team completing 18 or more items in this Sprint (by the way, you have just calculated the 50th percentile for your results distribution).

50% is not bad but it’s not great. What’s great is that using this technique we can compute the probability of any range of outcomes. The typical way to approach planning, therefore, is to determine how much risk our team is willing to live with and plan for that. So if being wrong 50% of the time is too much risk, how much risk is
tolerable? Again, for the sake of argument, let’s say our team wants to plan on being right 85% of the time. Simple. All we do is start at the right of the chart and start adding up the heights of until we get to 8500 (8500 / 10,000 = 85%). This is shown in Figure 5.3 below:

![Figure 5.3: 18 Or More Items Complete](image)

For these results, this 85th percentile occurs at 12 which means we have an 85% chance of getting 12 or more items done this Sprint. You’ll observe that to go from a 50% chance of being correct to an 85% chance of being correct that we went from forecasting 18 or more items to complete to forecasting 12 or more items complete. This result should be intuitively obvious: to increase our chance of being correct, we are going to forecast we can get less items to done.

We can use then this same approach to calculate any probability you want: 70%, 95%, 30%, whatever. Figure 5.4 shows some standard percentiles overlaid on top of a Results Histogram:

![Figure 5.4: Standard Percentiles](image)

After seeing this, your first inclination (and certainly the first
inclination of your customers) might be to say what is the 100% chance of being correct? First, if you are thinking that, then that means that you didn’t read Chapter 3. In that chapter we explained that there is no such thing as 100% in forecasting. Your second inclination might then be to think what your chance of being 99% correct is. While this is a more valid question to ask, it is not very useful. Although not drawn on the chart in Figure 5.4, the 99th percentile occurs somewhere around 1. So to be 99% sure this Sprint, you will forecast you can get 1 or more PBIs to done. Go say that to your PO with a straight face.

The idea is to pick a risk profile that is reasonable for your context. In most situations, planning to 99% is absurd. Rather, consider what is a reasonable success rate that allows you to plan with some confidence, but still acknowledges that we may be wrong from time to time. When starting out many teams find the 85th percentile strikes this right balance—though there is no hard and fast rule. If you are doing prototyping or greenfield development, then something like 70% (or even 50%) may be acceptable. If you are implementing government regulations or face major fines/penalties for being late, then maybe something up closer to 95% is better. What’s great about MCS is you can see all of those options all on one results chart and plan accordingly.

**The Language of Forecasting**

We did not call it out explicitly in the previous discussion, but you will notice that all of forecasting statements followed the same pattern. That is, each forecast contained a range of possible outcomes, and the probability of an outcome in that range occurring. For example, we said, “We have a 50% chance of getting 18 or more PBIs to done in this Sprint”. My recommendation is to always communicate your forecasts using this pattern.

Strictly speaking, while forecasting the number of PBIs to be
completed in a Sprint is specifically the Developers’ accountability, the Scrum Guide is pretty clear that Sprint Planning should be done in close collaboration with the Scrum Team’s Product Owner. Imagine that in Sprint Planning the Developers ran an MCS as above and then communicated to their PO that they could get 18 items done. What do you think your PO is thinking in that moment? Of course they are thinking that the team has a 100% chance of getting exactly 18 items done (when in reality we know that chance is more like 6%). We think this is a crucially important point because many times it is the PO who has a greater understanding of the business risk associated with a given Sprint Goal. If all you tell the PO is “18” then you are hiding vital pieces of information that will prevent the PO from being viable collaborative partner (not to mention that simply saying “18” is not very transparent). Our feeling is that both the Developers and the PO should have equal input into what percentile each Sprint should be planned to. This is especially true considering that the Scrum Team as a whole is accountable for achieving the Sprint Goal since the 2020 Scrum Guide update.

**But What About Complexity?**

Nowhere in the discussion of MCS so far have we mentioned estimating the relative complexity of the PBIs we are forecasting. That’s because with an MCS approach, estimation is not needed. It is unfortunate that the Agile community has been sold the snake oil that is story points, but the truth of the matter is that estimated relative complexity is an extremely poor predictor of capacity. The brilliance of the Monte Carlo Method is that if you choose your inputs properly, then most variables that could affect future outcomes will be encapsulated in your historical flow data and the need for any estimation goes away.

Think about it for a second. Your historical Throughput (not to
mention the other Flow Metrics) will include data for PBIs that as you worked on them:

- had a wide range of complexities
- were similar to other PBIs you had worked on in the past
- were not at all similar to other PBIs you had worked on in the past
- were blocked due to internal dependencies
- were blocked due to external dependencies
- were interrupted by other work
- were interrupted by holidays and vacations
- and so on...

The only thing your historical data won’t include are things that are impossible to plan for (like pandemics that shut the world down for several months). But, when they occur, those events would undermine any forecasting technique.

Your Scrum team encounters risk and uncertainty every day as you do your work. That risk and uncertainty is captured in the historical record in terms of your ability to get PBIs to done – namely, your Throughput data. Choose your historical input data right and chances are that you have covered the majority of the factors that could affect your Sprint Plan. And on top of that, you will still have the added flexibility of a Sprint Goal allowing for changes in whatever you’ve pulled (to some degree).

And that’s not even the best bit (although it really is). The best bit is that with the right tool, an MCS can be performed in minutes as opposed to estimating complexity which could take hours.

Speaking of complexity and sizing, there is still one aspect of Sprint Planning that we need to cover. But as this chapter has gone on long enough, we think it’s best that we pause and regroup before we talk about the critical subject known as Right Sizing.
Endnotes

Chapter 6 - Sprint Planning, Part II

We took our historical Throughput, fed it into an MCS, got an answer based upon our tolerable risk, so planning is done, right? Not quite.

Let’s say our MCS has spit out a range of possible outcomes with a given acceptable percentage of uncertainty such as “8 or more items with 85% chance of success”. That means for planning purposes we should pull in 8 PBIs into this Sprint. To be clear, however, the MCS does not tell you which items you should pull in, only that you have capacity for up to 8. It is the accountability of the Developers on our Scrum Team, therefore, to choose (in collaboration with the Product Owner) which 8 PBIs to pull in. The point here is that the MCS will not tell you which items you can get done in the next Sprint, it will only give you the minimum capacity to plan for. This can certainly be helpful in discussing what Sprint Goal you would want to commit to.

BUT! It’s a little more nuanced than that. The results of the MCS do not suggest that you can pull just any old PBI into the Sprint. Whether you realized it or not, what the MCS results really tell us how many right-sized PBIs can reasonably be done.

Right Sizing

“The Scrum Team may refine these items during this process, which increases understanding and confidence.”¹ The question from most Scrum Teams is “how much refinement is enough refinement”?
From a flow perspective the answer to questions like that is almost always “less than you think”.

There is a pervasive myth about flow that all items that move through your process have to be of the same size. After all, since you are not estimating complexity upfront then the only way to understand the work is to same size everything, right? Wrong. There is nothing in flow theory that demands that all items that flow system be exactly the same size. In fact, there is a whole theory of variation that acknowledges that not only do items not have to be of exactly the same size but that there is also nothing you could ever do to make them all exactly the same size—even if you wanted to. That is, variation in work item size will always exist.

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

The idea then is to understand what is the degree of variation that our flow system can handle. For a Scrum Team doing two week Sprints, for example, that variation in PBI size could be anywhere from 1 to 14 days, but in reality should probably be a little less than that. To get a notion of what “right sized” means in our context will require us to look at our historical Cycle Time data and calculate something known as a Service Level Expectation or SLE.
What an SLE is exactly, and how to calculate it will be topics covered in a later chapter, but know this for now: as you go through Sprint Planning and as you start to select PBIs based on the results of your MCS, you and your team should have a quick conversation around whether your item is right sized or not. For example, your SLE might be something like “85% of PBIs finish in 7 days or less” (does that language sound familiar?). So for each PBI that you select, your conversation might go something like this: “based on what we know about this PBI right now—which is not much–do we think we have an 85% chance of getting it done in 7 days or less?” If the answer to that is yes, then conversation is over and you pull the PBI into the Sprint. If the answer is no, then more refinement may be required. Remember the Developers always have the option to either (a) pull the PBI in as is and take that risk, or (b) choose not to work on that PBI in this Sprint.

[Note: I personally am biased on this point and believe that if the PBI is the next ordered one from the Product Backlog, that an effort should be made at refinement (refinement to me meaning: break the PBI up into several other PBIs–not tasks!–or change the acceptance criteria, or whatever). As you know, however, the ultimate accountability for selecting PBIs lies with the Developers so it is up to them how they proceed (this is one of many points that the Scrum Guide and I do not see eye to eye on). -Dan]

One final thing about right sizing for now is the Scrum Guide goes further and says that once PBIs are selected, the Developers could go even further. From the Scrum guide¹: “This is often done by decomposing Product Backlog items into smaller work items of one day or less.” Often doesn’t mean always, and you’ll find that you can comfortably skip this step when using Flow Metrics. If your items are right sized, then –from a flow perspective at least– no further breakdown should be needed.
**Tooling**

There are a lot of tools out there to help you use MCS for Sprint Planning, and if you are familiar with Dan’s work, you know he has a very strong bias to one tool in particular.

Having said that, another strong recommendation we have for you when starting out with MCS is to do a couple of models manually yourself. Using the four step algorithm above, it is a straightforward exercise to build out a spreadsheet template to do random selections and create a results histogram. We make this recommendation for two reasons:

1. Performing an MCS manually yourself will not only help you learn the technique to a greater depth, but will also give you confidence in the approach as a valid forecasting technique.
2. Possibly more importantly, if you are able to build MCS models yourself, then when you do finally graduate to using a tool you will be able to check the tool’s results of any simulation against yours. This validation is something we normally do when trying a new tool as you never really know how tools are calculating things behind the scenes (ahem, Jira).

**Drastically Reduce Planning Time**

We know this chapter sounds like Dan is on one big Scrum Guide rant and Dan certainly apologizes for that. He’s still learning. But there is one final point with regard to the Scrum Guide that we need to make when it comes to using MCS in Sprint Planning.

The Scrum Guide allows teams to budget up to 8 hours for Sprint Planning, as a timebox. It is the single largest budget for an event in the guide. For the same Sprint, Sprint Reviews only get up to four
hours, Sprint Retrospectives only get up to 3 hours, and if you were to add up all the Daily Scrums over a one month Sprint, you would only get to between 5 and 6 hours. 8 hours is a lot of time, and the authors have rarely seen it fully utilized. Will can count the amount of times on one hand. That said, many teams we’ve worked with bring it down to about 2-3 hours. With proper use of Flow Metrics you’ll find you’ll need far less than that.

If you choose the right MCS tool, you can get the “What?” part of the Sprint Planning down to literally a few minutes. Monte Carlo Simulations take seconds to run and then you’ll need a minute or two to interpret results. Take the time you need to figure out the “Why” part of your Sprint, establishing your Sprint Goal. After this, take a few minutes to select which PBIs are to be done, and ditch the “How?” part (as your process should already be established per Chapter 1) and your Sprint Planning event should be done in about an hour. Maybe even 30 minutes if you’ve got a clear Product Goal, Sprint Goal and well-aligned team. All of this regardless of Sprint length. No more wasted time estimating. No more useless fights over whether this is a 2-point or a 3-point item. Just run your MCS and go. That is truly Agile. Why? Because you don’t drive out uncertainty by planning work, you only drive out uncertainty by doing work.

**Conclusion**

We want to know what (how many) PBIs can be done in the next Sprint. A solution to this problem is straightforward: feed historical Throughput data into an MCS and interpret the results to make probabilistic forecasts about what can be done.

The fundamental assumption when using MCS for Sprint Planning is that the future that we are trying to forecast roughly looks like the past that we have data for. One of the big advantages of using Scrum is that if you are following the framework professionally,
then you are generally safe (pun intended) to use past data for future predictions.

In a very short amount of time you can get a very accurate forecast of what can be done in the Sprint thus allowing you to quickly focus all of your attention on what is important—actually doing the work. Which is the very topic we will cover next.

Endnotes

Chapter 7 - The Daily Scrum

If you believe that managing Flow gives you the best shot at getting PBIs to done, then the actionable plan coming out of your Daily Scrum should be all about addressing problems with Flow. The best metric to indicate Flow problems is Work Item Age.

To understand why, let’s take a step back and revisit the flow metric of Cycle Time (CT). 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.

Until a PBI 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 PBI is done. Thus, CT is really a measure of “time to validated feedback” (we are assuming here, of course, that your DoD includes some type of validated feedback—see Chapter 1 for more information).

However, CT itself can only be calculated at or after the moment when the PBI is actually done. Before it is done the only metric we have is the item’s Age. That aging process starts immediately once work begins. And we know the aging process only stops once the PBI is done (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 (you may have read somewhere about Agile being all about responding to change...). But what we do know is that age going up represents a increase in the risk that one or more bad things might happen. And the ultimate risk is that we spend a long time working on something that ends up not being valuable. As Dan’s 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.

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

The answer is the Service Level Expectation or SLE. SLEs are another one of those topics that you have probably not heard much about.

**The SLE**

In both the MCS Results Histogram and the Cycle Time Scatterplot we used percentiles to segment data along lines of uncertainty. In MCS, those lines helped us understand the risk associated with getting *multiple* PBIs to done. The Cycle Time Scatterplot is a little different in that each dot on the chart represents how long it took for a *single* item to get to done. The percentile lines on the Scatterplot then represent the risk associated with getting an individual PBI to done.

Understanding uncertainty at the individual PBI level is so important because it is at the individual PBI level where Flow actually happens. As individual items move through our process, we need
to know if they are flowing the way we expect them to. Like MCS, that expectation is based on how much uncertainty we are willing to live with in terms of how long it should take for a single PBI to flow from started to finished in our process.

To illustrate this point, think about it this way: In a two week Sprint, do you think 100% of PBIs will always complete within 14 days? We know this is impossible. So what percentage of time are you willing to be wrong? 85%? 70%? Something else? This is where the percentiles on the Cycle Time Scatterplot come in, because those percentiles will tell us our percentage chance of finish a PBI with a given range of time.

Take a look once more at the percentiles example from Chapter 4:

![Figure 7.1 - 30th, 50th, 70th, 85th, 95th, and Mean Percentile Lines](image)

This data is telling us something very interesting. It is telling us that
it should be *expected* that 50% PBIs should finish in 8 days or less; and that it should be *expected* that 85% of PBIs should finish in 15 days or less; and that it should be *expected* that 95% of PBIs should finish in 22 days or less. Based on how we are running our process right now, all of those outcomes should be considered routine. [As an aside, if you don’t like what your data is telling you then you need to change your process. Scrum has a wonderful event for just that which we will, of course, cover in a later chapter.]

From these percentages, pick which outcome you are most comfortable living with and then that range and probability becomes what is known as your Service Level Expectation, or SLE. To use the language of Chapter 5, the SLE is your forecast for how long it should take an individual item to flow through your process from started to finished.

So how do we apply our newly formed SLE to help with the Daily Scrum?

### Percentiles as Intervention Triggers

Recall from Chapter 4 that we made the statement:

“You will also see percentile lines on the Aging Chart. These percentile lines are exactly the same lines that we calculated for our Scatterplot. You overlay percentiles lines on the Aging Chart so that you can see how current items in progress are progressing as compared to the total amount of time it took previous items to complete.”

As items age, we gain information about them. The percentiles on our Aging Chart work as perfect checkpoints to examine our newfound information. We will use these checkpoints to assist in coming up with our actionable plan during the Daily Scrum.

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 taking more time than half the PBIs we have seen before. That might give us reason to pause. What have we found out about this item that might require us to take action on it? Do we need to swarm on it? Do we need to break it up? Do we need to escalate the removal of a blocker? The urgency of these questions is due to the second thing we can say when an item’s age reaches the 50th percentile. When we first pulled the PBI into our process it had a 15% chance of violating its SLE (that is the very definition of using the 85th percentile as your SLE). Now that the item has hit the 50th percentile, the chance of it violating its SLE has doubled from 15% to 30%. Even if that does not cause concern, it should at least cause conversation. This is what actionable planning in the Daily Scrum is all about.

When an item has aged to the 70th percentile line, we know it is older than more than two-thirds of the other items we have seen before. And now its chance of missing its SLE has jumped to 50%. Flip a coin. The conversations we were having earlier (e.g., pair, swarm, break the item up) should now become all the more urgent. And they should continue to be urgent as that PBI’s age gets closer and closer to the 85th percentile. The last thing we want is for that item to violate its SLE—even though in this example we know it is going to happen 15% of the time. We want to make sure that we have done everything we can to prevent a violation occurring. The reason for this is just because an item has breached its SLE does not mean that we all of a sudden take our foot off the gas. We still need to finish that work. Some customer somewhere is waiting for their value to be delivered.
What If PBIs Aren’t Flowing As Expected

Pairing, Swarming, and Mobbing

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

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

- Completing downstream tasks earlier - As an item ages in an earlier stage, we can enable faster flow through later stages. We can perform steps in the later stages earlier so that the task does not continue to age unnecessarily once it is past the current stage.
- Dividing PBI tasks amongst team members - If the item itself cannot be broken down into deliverable chunks, it is possible to identify sub-tasks for the item. Different team members can take on the varying subtasks in parallel to help the item move forward.
- Removing Sticking Points - Often getting fresh perspectives on a problem a single team member has been facing helps in
coming up with creative solutions. Whether these are just a result of “rubber ducking” or cross-functional pairing to get new perspectives, they help an aging item make progress.

**Unblocking Blocked Work**

By definition, any work that is blocked or on-hold is by definition not flowing. These PBIs 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” book and you will quickly realize that one 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 a PBI 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 PBIs to flow in small batches of value. This means breaking things down into small, manageable—but still valuable—chunks.

We talked about the practice of right sizing in the last chapter, but as a quick review, Prateek Singh communicated this guidance to one of his teams (he used User Stories for PBIs):

“The Cycle Time Scatterplot for our team showed 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 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 refinement 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.”

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

Breaking Down PBIs

When you discover that a PBI is too large for your process, your first hypothesis should be that this large PBI is probably composed of smaller, individually deliverable bits of value. Packaging multiple items into a single PBI can negate many of the benefits that
managing for flow provides. For example, if we are operating with a WIP limit of 1, but that one item could potentially have been 5 separate items, our WIP is actually five times bigger than what is visible. WIP often hides in large PBIs. In order to expose our actual WIP, we should be breaking down work into individual “feedback-able” pieces early and often.

Let’s now walk through some simple strategies that can be used to break down PBIs. It is a list that Dan, Will, and others have successfully used in the past to break work down. Multiple of these strategies can be applied to the same item as well. The end goal is to create smaller units of work that can help us get faster feedback from our customers.

**Using Acceptance Criteria**

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

For example, consider the following PBI.

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

- AC 1 - Reviewers should be able to see the title of the paper separate from the body
- AC 2 - Reviewers should be able to see the word count of the description
- AC 3 - Reviewers can grade each section separately
• AC 4 - Reviewers are able to view the main hypothesis in a separate section
• AC 5 - Reviewers can optionally separate out experimentation details to be assessed separately

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

Using Customers & Personas

Many teams we’ve encountered over the years serve very big customer bases with their products, whether these are software product, robots or HR policies. Regardless of the product, many of these teams talk about “the customer” in a very generic, all encompassing way. Sometimes even with a hint of pride: “our customers are all the office workers worldwide”. While this sounds great for the press and future investors, it doesn’t allow for much focus in the teams. More importantly, it increases the size of the work. But we can also use our knowledge of our customers to break up work again.

Consider the following PBI:

“Enable secure login for employees.”

We might break this up in multiple ways for different customers:

• Employees working from home
• Employees working from the office
• Employees in a specific country or region
• Employees in a specific department of the company
• Employees in a specific role
The list can go on and on. By being more specific about our customers, we can deliver things quicker to those customers, allowing us faster feedback. We’ve found this works as well for internal as external customers. Not all customers are equal, and we can use that to our advantage.

**Assumptions**

Ok, this goes into a topic Dan is less comfortable with: Outcomes. But as described by Jeff Gothelf and Josh Seiden in the book Lean UX³ “Each design is a proposed business solution — a hypothesis. Your goal is to validate the proposed solution as efficiently as possible by using customer feedback.” In other words, as we said before in this book, all PBIs are an assumption of value until validated by the customer.

If we want to break up a PBI then, one thing we can ask is “what are all the assumptions we’re making about the value of this thing?”. And following that, we could break up the item into its constituent assumptions and deliver those separately. Taking the example from before, we might break up the item to research individual assumptions such as:

- The current login isn’t secure
- All employees require similar security practices
- All countries require similar security practices
- The secure login we designed is safe
- Security is best enforced by a technical solution

Another approach we could use, potentially in combination, is Giff Constable’s Truth Curve⁴. The article is a good read, but what it essentially boils down to is figuring out the following: Given the evidence we have as to the potential value of a PBI, what is the least amount of effort we should spend to figure out if it is worth investing more time/budget in this PBI, and growing our evidence
of value. In our case, this amount of effort should comfortable fit within your SLE. Continuing to use the security example, we’d probably start with:

- Some conversations to check why this item even exists: Is it a legal requirement? A compliance issue? Is the current system not working?
- If those PBIs indicate a need for change, do an observation study of the current users: What’s happening now? Why are the current policies insufficient?
- Run a paper prototyping session testing out new secure login functionality. Is it easy? Safe? Foolproof?
- Run a Wizard of Oz experiment to see if people use it when they don’t know they’re being observed.
- Build a prototype to validate the technology.

We can keep going, but hopefully you get the point. All of these PBIs can be delivered a lot faster, with faster feedback, than the vast starting point of “Enable secure login for employees”.

**A Daily Scrum Example**

Let’s say the Developers on a Scrum Team show up to their Daily Scrum and pull up a WIP Aging Chart that looks like this:
First we should point out that all data and metrics are context specific so this example is for illustrative purposes only. The decision that your team makes may be different than these based on contextual information and that is fine—actually, it is recommended. However, what follows is a strategy that should work most of the time.

- **Look at oldest items first:** What’s the oldest item on the board and is it currently being worked on? What do we need to do to make progress on that item today? Does it need to be broken up? Do we need to swarm/mob/pair on it? Is it still valuable? Rinse and repeat for the next oldest item until the team has their plan for the day.
• **Make note of excess WIP**: There’s 4 things in Dev Done, what’s happening there? We would obviously need to look at the Kanban board to know if 4 things is excess WIP or not.

• **Discuss items at risk of violating the SLE**: There’s work in Analysis done that’s already 13 days old. The chance of still finishing it within the SLE is fairly low at this point. Should we swarm on it? See if we can still finish it in the next two days (assuming the 85th percentile is your SLE)?

The simple heuristic for Daily Scrums is—all other things being equal—you want to focus your attention on the oldest PBIs first. To optimize flow, we want to minimize Age which means the oldest PBIs on your Aging Chart are good indicators that something is wrong. The last thing you want to do is ignore old items because all that is going to do is make those PBIs older.

**Conclusion**

The Daily Scrum is perhaps the Event most actively impacted by using flow metrics. While the Sprint Planning is dramatically shorter with this tool, the nature of that Event doesn’t change all that much. You still figure out a Sprint Goal and try to forecast what work can be delivered to reach that goal. With the Daily Scrum however, we see a change. The goal of the Daily Scrum remains the same: Inspect progress towards the Sprint Goal. But with the use of Flow Metrics, the conversation changes from “how do we work together today and help each other reach the Sprint Goal” (person focused) to “how old is this stuff, and how can we get it moving” (work focused). Using Flow Metrics in your Daily Scrum actually encourages more team work by focusing on the work to be done. Because when the work moves and gets done, you get the feedback you need to actually inspect your progress towards the Sprint Goal, not just the stuff you think you need to deliver.
Endnotes

Using Flow Metrics in a Sprint Review can easily derail the conversation into a scope discussion. On the other hand, using Flow Metrics wisely can open up the conversation with your stakeholders on timing, budgets and outcomes. Regardless, Flow Metrics allow you to switch the conversation to a more future-facing one. Many teams still struggle with running “demos”, with everyone present (correctly) feeling like this is a waste of time. When this isn’t due to a simple lack of knowledge on what the Event should be about, it’s mainly due to the fact they can’t really say anything about the future. Well, not anymore. Because now you’ve got measures, charts, and answers to the most pressing questions of your stakeholders.

Let’s talk about what all of this could look like.

What Did We Learn Last Sprint?

Any good Review (and Will is extremely opinionated on this topic) will start with re-aligning everyone present on the Product Goal and vision you’re trying to realize, and the Sprint Goal you had for that Sprint. Having reminded everyone why you spent time (and money!) building something, you can then talk about what you learned that Sprint. From the Scrum Guide¹: “The purpose of the Sprint Review is to inspect the outcome of the Sprint and determine future adaptations.” Now, this may be controversial, but we don’t think this necessarily means demonstrating your Product (though it could help from a show, don’t tell perspective). Rather, you should
be talking about what came out of you delivering value to your customers, using the current state of Product to illustrate points of feedback you received.

Following that, you can be transparent about things you learned from using your delivery process. Here’s where you can start using Flow Metrics. Did you run into any delays? Perhaps due to dependencies outside the team, or work sent back from downstream (if you’re in a situation where you don’t deliver to the end customer)? Showing your Cycle Time Scatterplot here, and zooming into last Sprint could show the impact of issues that your stakeholders might help you with. Or perhaps it just shows why you had to deviate from your initial scope to still meet the Sprint Goal.

This is also a good time for stakeholders to share market or business developments that might impact your workflow. Perhaps they might be entering foreign markets that require changes in your validation process. Or you might finally expand your teams’ workflow further downstream, because handoffs are bad (as they tend to introduce delays and loss of knowledge). Regardless, the first part of your Sprint Review should be an inspection of where you are today in terms of value and process.

When Is It Done?

Dan wrote a book about this. That was a pretty smart move, considering how prevalent this question still is in modern organizations. There’s a big can of worms here that we’re not going to open with regards to the question of whether or not “it” will ever be “done”. If you’re into that kind of stuff, we recommend you look at other sources. Sometimes it’s a valid question. Things like legislative requirements or contractual agreements on functionality certainly have clearly defined scope that needs to be met. Most time it isn’t though, as the complexity of work means you have no idea if feature A will result in outcome A until after you’ve built it.
All that said: If you can’t answer this questions, regardless of its validity, people get very nervous. Fortunately, we can answer this question somewhat using our trusty Monte Carlo technique. The technique for a Monte Carlo: When probabilistic forecast is a little bit different from the Monte Carlo: How Much technique, so we’ll go over it quickly.

Take an amount of items you want to get done. This can be an arbitrary amount (10, 50, 100, etc) or whatever amount is in your Product Backlog. Let’s take 10 for this example. Now look at your historical daily throughput numbers, we’ll use these for the forecast. Let’s imagine we have a history from the last 6 days of delivering 2, 1, 2, 2, 0, 1 items per day. Going forward, we can use these as possible throughputs, and see how much time it’d take, like so:

<table>
<thead>
<tr>
<th>Day</th>
<th>Throughput</th>
<th>Remaining</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>8</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>8</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

**Figure 8.1: Sample Throughput Forecast**

So in this “run”, it’d take 7 days to deliver these 10 items. But this isn’t a probabilistic forecast yet. If you remember from previous chapters, you’d want to run this simulation a few more times (about 1000-10,000 times more). In doing so, we’ll see our trusty distribution of possibilities emerge, and we can count again how often certain opportunities present themselves. Here though, we see the chance of delivering at least 10 items goes up as we move further in time.
Using this technique with your own data, you can give a probabilistic answer to the “When Will It Be Done” question. We can’t guarantee your stakeholders will like the answer though. Nerves calmed, you can then talk about whether or not all of those items are needed, what risk level people are comfortable with, and if there’s enough budget or time left to even get you to the projected dates.

What Can Be Done?

Of course, not every team deals with a fixed scope (fortunately) or even a known scope. Sometimes you find yourself in the opposite situation: You’ve only got a certain amount of time and/or budget, and you’re going to try the best you can in that time. In such a case, you’d use a *Monte Carlo: How Many* forecast. The same one you already use in your Sprint Planning, but with one difference: Instead of a forecast until the end of the Sprint, you’ll forecast until the date your time and/or budget runs out. That’ll give you a range of items and the associated probability, which can then fuel your conversation around what those items should be. Or, you know,
it could be way less than your stakeholders were hoping for, and you’ll be in for a nice budget conversation. Or a shouting match.

Here’s an example of what a *How Many* projection could look like in a Sprint Review, if you were to project out from 60 days (because that’s when we’re out of money). Note that this uses a more complete dataset than the earlier example.

![Monte Carlo: How Many](image)

Now, if you’re dealing with some contractually agreed upon scope of 120 items, you’re looking pretty great here. There’s a 95% chance of delivering AT LEAST 129 items. Stakeholders will be happy, and you’re good to go for a value discussion.

Of course, if the scope is bigger than this, it becomes progressively less likely you’ll be able to deliver everything. For instance, if your scope is 170 items, it’s almost a coin-flip, given that the 50% line is at 172. Please note (also for your stakeholders) that this means that in 50% of the projections you delivered UP TO 172 items. This is not a 50/50 chance of delivering exactly 172 items. In any case, this might be the moment you bring up the idea of delaying a release, prioritizing (de-scoping), further empowering the team or removing delays, or worst case: adding more resources. That last one (almost) never works.
Now What?

Of course, everything up to this point is related to output. And if output was all that mattered this would be the end of the conversation. But Scrum Teams aim to deliver value. Your Sprint Review should be focused around using the insights from your Sprint, feedback on your Product, value measures and marketplace inspection to update your plan (your Product Backlog). Flow Metrics are meant to help you in this process, not replace it. Probabilistic forecasting can give you insight as to how many items might get done, but your Sprint Review is meant to figure out what those items could be.

Never forget though to always remind your team and stakeholders of the percentages associated with your forecasts. There are no guarantees in a complex environment, so your projections might (and probably) will change over time. So make these conversations and measures a standard part of your Sprint Reviews.

So, with Flow Metrics as your foundation, explore that value!

Conclusion

Up until now, we’ve only talked about using Flow Metrics as a Scrum Team for controlling your workflow. This chapter explored the use of Flow Metrics beyond your Team, and as a part of Product Management. And we’ve hopefully convinced you of the tremendous value they can offer in improving your relationship with your stakeholders. From offering a data-based answer to the “When Will It Be Done” question, to being able to forecast how many items could fit within a certain amount of time: Your data will allow you to elevate the conversation, skipping the output-related questions and moving you quicker to value.
Endnotes

Chapter 9 - The Sprint Retrospective

Dan’s friend and colleague Frank Vega likes to point out, “Your policies shape your data and your data shape your policies”. In other words, your data is telling the exact story of the process you have designed. Not the one you wished for, not the one that you expected, but the one that you have implemented and are executing every day. That story is captured in the Cycle Time Scatterplot, and one of the best times to read that story is during the Sprint Retrospective.

The Cycle Time Scatterplot not only captures your story of your current Sprint but also of all past Sprints that you captured flow data for. The benefit of this historical record is that you can see certain patterns and clusters that may emerge over time on your Scatterplot. It is these shapes that are going to give us the biggest clue as to where to focus our process improvement efforts for the next Sprint and beyond.

What follows is a discussion of some of those patterns and how they might be interpreted for improvement. Much of this information is a restating of what Dan first published in his “Actionable Agile Metrics for Predictability” book¹. Once again, if you are familiar with that book, then you can reasonably skip this chapter without loss of continuity. However, as that original book was published with a purpose to be process agnostic, it might make sense to review some of the details below with a Scrum lens on.
The Triangle

A triangle-shaped pattern as shown in Figure 9.1 will appear in any situation where Cycle Time increases over time.

![Figure 9.1: A Triangle Pattern on a Scatterplot](image)

Notice how the dots in the above Scatterplot form a pattern that looks something like a triangle. Explaining this phenomenon is going to require us to review the fundamental property of Scatterplots: dots do not actually show up until a work item has finished. The items that have longer Cycle Times are going to need an extended period before they appear on the chart. That means that the longer the Cycle Time (the dot’s Y-component) the longer the amount of time we are going to have to wait (the dot’s X-component) to see that data point.

The triangle pattern appears whenever you start with a process with zero WIP. This is because it takes time to “prime the pump” and get items to done. Obviously, in those early stages work will be pulled in faster than it departs—even if we are limiting WIP. We are going to need time for each workflow step to fill up to its capacity and get a predictable flow going. This pattern is exacerbated in situations...
where teams feel they have to empty the process at regular intervals too. In this case what you will see over time is a repeated triangle pattern, which shows an issue with flow. An example of what we’re talking about is shown in Figure 9.2:

While the timeframes of both Figures 9.1 and 9.2 are much longer than a Sprint you can still see how this pattern could emerge over time in Scrum, given what we just explained above. From a flow perspective Figure 9.2 is terrible: the team started on everything around the same time, showing massive WIP. From a Scrum perspective, it just means that this pattern should be discussed in the Sprint Retrospective. As an exercise for the reader, what questions might you ask if you see the pattern in Figure 9.2 emerge in every Sprint?

**Clusters of Dots**

The second type of pattern that might emerge is an obvious clustering of dots on your Scatterplot. Consider, for example, the following chart in Figure 11.3:
Note the clusters of dots at the beginning of October 2008 (around the middle of Figure 11.3) and at the end of July 2009 (the lower right side of Figure 11.3). As with all of these analytics, the point is to be able to ask the right questions sooner. So, when we see clusters of dots like in Figure 8.11, we are at the very least going to want to ask “what’s going on here?” That should probably quickly be followed by “is this a good thing or a bad thing?” If it is bad, what can we do about it?

By the way, not all clusters of very low Cycle Times are good. Look at the cluster of dots for July 2009 again in Figure 8.11. What do you think might be causing our Cycle Times to have decreased so radically? Are you only thinking of good reasons? What might be some bad reasons that would cause this to happen? One sinister reason that we see all too often is mandatory overtime. It stands to reason that if your normal data is based on 8 hour days and 5-day work weeks that moving to 12 hour days and 7-day work weeks will probably make your Cycle Time look better (assuming, of course, that you continue to limit WIP!). But is that a good thing? Some managers would say yes. We would say otherwise. And from a predictability perspective, it is terrible. Not only are long periods of mandatory overtime not sustainable but it also skews our data.
Do you really want to be offering an SLE or making a forecast with mandatory overtime as one of the upfront assumptions that is baked in? If your answer to that question is “yes”, then this book is not for you.

**Gaps**

Gaps in the dots on your Scatterplot means that no work items finished in that particular time interval:

![Figure 9.4: Gaps on a Scatterplot](image)

In short, the lines that you see on Figure 9.4 are indicative of batch transfer. It is not uncommon for a Scrum team to generate a Scatterplot that looks like Figure 9.4. In this example it is quite obvious that the stacks of dots that you see are at Sprint boundaries—probably when there is a mad rush to complete PBIs before the end of the Sprint. But look at how the data thins out between those stacks. Is this a good thing or a bad thing? Is this even how Scrum is supposed to work? Either way, what impact is this having on our predictability? If you think it is a bad thing, what might you do change that?
Internal and External Variability

The reason that Scatterplots look like a random collection of dots over time is because of the variation that exists in your process. The first thing to know about variation is that it will always exist. From a predictability perspective, the point is not to always try to drive variation out; rather, the point will be to understand the causes of that variation in an attempt to make your process more predictable.

For example, take a look at Figure 9.5:

![Figure 9.5: Gaps on a Scatterplot](image)

At first glance you might be inclined to dismiss those dots at the top of the Scatterplot as outliers. You might question the value of including them since they are clearly one-offs. You might even (if you did not like yourself very much) do some further quantitative analysis to prove that those dots are not statistically significant. And you know what, if you made those assertions then we probably would not argue with you too strenuously. We would say, though, that while those points are outliers, they obviously happened and probably warrant some deeper investigation. We would also say that, while potentially statistically insignificant, there might be
some good contextual or qualitative reasons to keep them in from an analysis perspective.

To illustrate this point, consider what the chart in Figure 9.5 is communicating to us. The 50th percentile of Cycle Time is 20 days and the 85th percentile is 44 days. But you can see there is a work item on this chart that took 181 days! Can you think of some reasons that would have caused that particular work item to take so long? Maybe the team had a development dependency on an external vendor or a dependency on some other internal development team. Maybe the team did not have a test environment immediately available to them. Maybe the customer was not immediately available for sign-off. The shared theme for all of these reasons is that those work items took so long to complete due to reasons outside of the team’s control. And that is generally what you will find as you move “up the stack” of dots on a Scatterplot. More often than not, those outliers will be caused by circumstances that are outside of the team’s control.

The opposite is also generally true. As you move “down the stack” the work items that took less time to complete were generally due to reasons that were totally under the team’s control. For example, reconsider that work item that we just mentioned that took 181 days to complete. Do you really think that item would have taken 181 days if it was totally under control of the team that was working on it? Maybe, but probably not. Additionally, look at those dots that just barely violated that 85th percentile line. Do you think that there were things that the team could have done to ensure that the violation did not happen? Probably. Swarm or break up the item are two ideas that immediately come to mind.

We hope that you are getting a feel for the type of variability analysis that we are asking you to perform with these Scatterplots. Will all outliers be due to external causes? Certainly not. Maybe the team allowed an item that ended up being too big into the process. Maybe the team ignored an item once it had been pulled. Likewise, will there be external issues hiding in the shorter Cycle
Times? Almost certainly. But at least we have shown you how to use a Scatterplot with percentile lines to begin the conversations about how to address those issues. Further, the more you adhere to the assumptions of Little’s Law, the more confident we can be that the “up the stack” dots are due to outliers, and the “down the stack” dots are due to team policies.

**Conclusion**

Data without action is meaningless. The purpose of your Sprint Retrospective is to improve your overall process over time. Metrics are just one, but important, piece of that.

As you go into your next Sprint Retrospective, remember that your policies shape your data and your data shape your policies. Your data is telling you the story of your process. Are you listening?

**Endnotes**

Chapter 10 - Tooling

Here we go, the chapter that’s going to get us vilified on the interwebz. Tooling is an incredibly divisive topic. We’re not going to give you explicit recommendations for what you should be using though. What we’ll do instead is share some tips on what to look for in a tool in terms of features. They’re also conveniently listed in order of priority.

We’ll also share some red flags. If you’re in a situation where you’re forced to use one of these tools, you’ll know you’ll need to work around these. Better yet, avoid them all together.

[Note: While I know most teams are beholden to what their organization selects, in my opinion the line in the Scrum Guide¹ in the Scrum Team section reading “They are structured and empowered by the organization to manage their own work.” means that Scrum Teams should be free to select whatever tool they believe will serve them best. Even if this means each team has their own workflow/backlog management tool. Most arguments against this spread come from either a “management needs to see things combined” or some sort of cost cutting background. This is utter nonsense. People outside the team should be looking at outcomes. Not outputs. And nothing slows an organization down faster than standardizing on the wrong things. -Will]

[Follow Up Note: Occasionally people raise the issue of tracking dependencies across teams through a tool. While ideally you should treat dependencies like any other impediment and try to resolve them, I’m conscious of the fact that for many teams they’re still a fact of life. If you absolutely must track them, this is a question of data, not tooling. Even in such a case where you’d want to track data across teams, I’d advise combining the data directly through the back-end, rather than taking the choice of tool away from the
team. Does this mean you should pick tools adhering to common standards with APIs for accessing data? Obviously. - Will]

**What To Look For In A Tool**

Here we go! This is your checklist for selecting the right tool. Either use it yourself or send this over to procurement. In the latter case, use the term “non-negotiable requirement”. Remember, pragmatism is where good things go to die. We’ll list our tips in order of importance.

**Data**

First and foremost, your tooling must generate good data. Most other things you can do with discipline, but if you need to manually keep timings on items that’s going to be a good chunk of your day gone. We’re serious here: Will once coached a team that had to work with a ticketing system that only tracked the time the item was created (not pulled), closed and since the item last moved, for some reason. That meant that for Cycle Time purposes someone from the team had to manually log what was in what step of the workflow once a day. Took about an hour. Each day. So yeah, find a tool that keeps time accurately. So what do we mean by that?

It should at the very least:

- Log the ID of the item
- Log the date (and potentially time) each item enters a step of your workflow.
- Correctly discard times downstream if you have to move something back (because it incorrectly proceeded).
- Export this in an easy to use format (CSV, Spreadsheet or API) so you can check it (and import it into other tools if needs be).
It’s great if it also:

- Logs the name of the item (so you don’t have to cross-reference)
- Logs the time the item spent blocked.
- Logs item type as an attribute (so you can filter).

Here’s an example of the kind of data you’d want from an item, using a Year-MonthDay-HourMinute notation, and blocked time in minutes:

<table>
<thead>
<tr>
<th>Item ID</th>
<th>Item Name</th>
<th>Product</th>
<th>Sprint</th>
<th>Design</th>
<th>Creation</th>
<th>Tweaking</th>
<th>Done</th>
<th>Type</th>
<th>Time Blocked</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Wobbles</td>
<td>2022</td>
<td>2022</td>
<td>2022</td>
<td>2022</td>
<td>2022</td>
<td>2022</td>
<td>2022</td>
<td>Feature 137</td>
</tr>
</tbody>
</table>

This is already a rich dataset. It contains everything we need to make all the charts we mentioned before. We can create Cycle Time Scatterplots using the “Sprint Backlog” and “Done” data, because if there’s no date in “Done”, it’s not Done. We can create a Work Item Aging chart using the data from each step in the workflow. We can measure Throughput by looking at the amount of items that got Done within a date range, and we can measure historical WIP by counting the items in each step within a date range. And we can do this by item type. For the cherry on top, we can even look at the time an item spent Blocked for some reason. This is going to be your most important feature. If it can’t do this, find another tool.

Easy Workflow Design

Products and their associated workflows will change over time for all the reasons that make work complex. Sometimes anticipated, most times due to sudden insights and influences. This means that your tool should be flexible enough to support fast and easy
workflow (re)design. Obviously, this starts with your team being able to alter their own workflow in the tool they use.

[Note: Honestly, the fact that there are teams to this day that have to ask some sort of “admin” to adjust their workflow, or are forced to use a workflow designed by someone else, is just infuriating. This disrespectful infantilization of teams really gets me riled up. -Will]

Do keep in mind that your workflow design is more than just creating columns. It also means defining and visualizing WIP limits, exit criteria, item types, pull policies, how you display blocked items, etc. Columns alone aren’t a great start. Though if you’re in a bind, you could cover the rest with discipline or recording things elsewhere. Do make sure that your data remains accessible using the new workflow. Your tool should be able to handle this gracefully. If you decide to cut a step in your workflow (for example the “Tweaking” step in our earlier example), your tool should no longer export the Tweaking step, even for items that have data there. But if you decide to reintroduce that step, you also don’t want any data to be lost.

What we’re saying here is that your tool shouldn’t limit or delay the evolution of your workflow.

**WIP Views & Warnings**

Ok, you’ve got your data and your workflow sorted. Next up on the list is something that’s nice to have: WIP views and warnings. The reason we’re saying this is a nice to have is because you can (and should) manage WIP primarily through discipline. The other reason is that if your WIP starts hitting double-digits in columns, it’s probably already too high anyway.

All jokes aside, the reason we’re talking about these is because there’s a common misconception in the market that in a flow system (or Kanban system) WIP limits only exists on a per-column basis. Whereas the Kanban Guide is quite clear when it says “A
WIP limit can include (but is not limited to) work items in a single column, several grouped columns/lanes/areas, or a whole board. This means your tooling should allow you to define WIP limits in whatever way (or combinations of ways) you please. Examples include:

- Per column
- Per item type
- Per lane
- Per person
- Per release
- In total
- (Any combination of) all of the above

On top of that, you’d want your tool to also warn you in some way when you’re going to break them, or even prevent it. Many tools will happily and silently allow you to keep pulling more work, with the only feedback being a tiny “(!)23/7” on top of the column. That’s not much of a warning, is it? If your team is serious about flow, you might want to outright prevent limits from being broken, or require extra effort for each item you want to pull over the limit. Perhaps even with a warning to an internal Slack or Teams channel.

Again, this is not meant to be a substitute for team discipline. But support from a tool can make this discipline a lot easier to maintain.

**Exit Criteria**

At this point we’re hitting complete luxury levels from a Flow Metrics perspective (though still basic Kanban hygiene). But when we’re visualizing a workflow we should do more than just list the steps: We should visualize when each step is done. In other words, what are the exit criteria for each column? Again, a team could just document these somewhere else, or even in a general Definition of
Done that isn’t specific to any particular column. But ideally, your Definition of Done should be spread out and visible throughout your workflow, in an easy to absorb format. Not many tools support this, but those that do make the lives of Developers a lot easier.

**Red Flags**

Having gone through the features you’re looking for, let’s now also look at some red flags you want to be on the lookout for. These will range from instant dismissal to “you can use the tool, just avoid this feature”. We again realize not every Team has the luxury of picking their own tool [grumble - Will], but even in that case this list should provide some talking points with the powers that be.

**Mandatory Estimates**

Rightsizing work means you don’t need to fill in estimates in the tool, because you’re not making traditional estimates. Note that estimates here can mean anything from Story Points to T-Shirt Sizing to actual hours. We use none of those. You could argue that saying “Yes” or “No” to an item fitting within the SLE is an estimate, but that doesn’t take away the fact that your tool won’t need to track this. So this field isn’t needed. “But authors,” we can hear you say, “why can’t we just fill in whatever in the estimates field?”. Because a tool that has mandatory estimates is going to try to be “helpful” about them, and this is going to serve as a distraction during your Sprints at best, and provide you with junk data at worst that’ll anchor your team or influence your stakeholders in unpleasant ways. Simply put, you will not need to enter any estimates in your tooling when you’re using Flow Metrics. A mandatory field for them is a big warning sign the tool doesn’t support good Flow Metrics. Unless an administrator (admin) set that up, in which case your Scrum Master should get your team
admin rights to their tools. Or have a good conversation with the administrator.

**Mandatory Assignments**

“The entire Scrum Team is accountable for creating a valuable, useful Increment every Sprint.”¹. Seems pretty clear, right? Right? Then why would a tool allow you to assign backlog items to specific people? That’s already pretty awful. You know what’d be even worse? The tool forcing you to assign work to specific people. This makes sure the rest of the team understands that clearly they don’t need to care about that item. All right, enough with the sarcasm. You want your tool to encourage teamwork, and for everyone to care about the Sprint Backlog, and swarm when necessary to get things moving that would otherwise violate the SLE. Individual assignments signal the exact opposite of that. So if a tool mandates it, turn off the setting or don’t use the tool. If you must use the tool, create a dummy team member called “Scrum Team” and assign all the work to them.

**Averaging Graphs**

At this point in reading this book you’ll have seen quite a few graphs. Graphs are nice, when they visualize the right things. Graphs showing averages don’t do that. Understanding probabilistic forecasting is hard enough for team that works with them daily, you don’t need the added distraction of meaningless averages. Worse still, your stakeholders might see it and start developing the wrong expectations. All in all, it’s better to have a tool that produces no graphs (as you can use other tools for this, including Excel) than a tool that shows bad graphs.
Conclusion

Tools can be a great help or a great hindrance. We hope this chapter provided you with some good criteria to use when choosing or evaluating a tool. A final tip we’d like to share is to start simple. Don’t rely on a tool to compensate for a lack of knowledge or discipline in a team. Get to a point where your team has achieved Flow in their delivery, and find a tool that supports that Flow.

Endnotes

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

**Starting Steps**

**Step 0: Monitor Work Item Age in your Daily Scrum**
The one big takeaway from this book is if you want to introduce Flow Metrics to your Scrum Teams, the place to start is with Work Item Age. And the time to look at Age is during your Daily Scrum. If you get Aging right, then everything else will fall into place. However, it is a bit disingenuous to say “start with Age” because chances are if you have been doing a more traditional implementation of Scrum that you don’t even have the basics in place to monitor Aging. So what follows are some steps you may need to do first in order to layer in Aging and other Flow Metrics into your Scrum-based process.

Define start and finish points
We need to determine when we consider PBIs to have started and when PBIs are done. These form the boundaries within which we will measure and manage all Flow Metrics. These boundaries will and should change over time, but we always need to be aware of what they are so we can effectively observe and improve the system.

**Determine all activities that help create value**
As PBI moves from the start point of a system, to the finish point of a system, we perform action on it to turn it into deliverable value. We need to determine what these activities are. For a piece of software these might be - Understanding the problem, Technical design, Creating the code, Code Review, Unit testing, Integration Testing, Build and Deployment, Customer Validation. Systems will have a varying number of activities that contribute to a work item becoming a piece of deliverable value. As we list them out we get an idea of the flow of the system. Aging really comes into its own when we understand the context of the value stream through which a PBI is moving.

**Select a starting Service Level Expectation**
Next, we need to create a common understanding of “How long does it take us to get work done?” via an SLE. We can do this by analyzing the cycle time distribution of items that have previously finished. Based on this distribution we can pick an SLE that represents the amount of time it takes the majority (70%, 85%, 95% etc.) items to get done. This SLE becomes the yardstick for items that are active in the system. Note that when starting out you most likely won’t have the data needed to calculate your SLE. That’s ok. Our recommendation is in the absence of data that you just pick a reasonable SLE to start. For example, if you are running a 14-day Sprint, a reasonable SLE might be “7 days or less @ 85%”. Or it might be something even shorter than that. The point is to just pick something so that you have your gauge to which you can compare Age during your Daily Scrum. Then, over time, once you have a good amount of representative data, you should adjust that SLE up
or down based on what your data is telling you. The time to have that conversation? You guessed it: the Sprint Retrospective.

Conclusion

There are many ways to get started with Flow Metrics in Scrum. This chapter provides a blueprint of the approach that we have seen work. The majority of the work to get started is around understanding your current activities and defining your workflow and policies. Once we have a common understanding of these, we can start watching the reasons why items age in our system and make the appropriate adjustments.

It’s up to you from this point forward. Use your data, inspect and adapt your workflow as needed, build great things. Good luck!
Bibliography


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


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

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


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


