POCKET-SIZED INSIGHTS FOR SOFTWARE TEAMS

TEAM GUIDE TO METRICS FOR BUSINESS DECISIONS

Mattia Battiston & Chris Young

2

conflux

Team Guide to Metrics for Business Decisions

Mattia Battiston and Chris Young

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

This version was published on 2021-06-24

ISBN 978-1-912058-65-5

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.

Contents

1. *Team Guides for Software*

Pocket-sized insights for software teams

The *Team Guides for Software* series takes a *team-first approach* to software systems with the aim of **empowering whole teams** to build and operate software systems more effectively. The books are written and curated by experienced software practitioners and **emphasize the need for collaboration and learning, with the team at the center**.

Titles in the *Team Guides for Software* series include:

1. *Software Operability* by Matthew Skelton, Alex Moore, and Rob Thatcher
2. *Metrics for Business Decisions* by Mattia Battiston and Chris Young
3. *Software Testability* by Ash Winter and Rob Meaney
4. *Software Releasability* by Manuel Pais and Chris O'Dell

Find out more about the *Team Guides for Software* series by visiting: http://teamguidesforsoftware.com/

2. Conflux Books

Books for technologists by technologists

Our books help to accelerate and deepen your learning in the field of software systems. We focus on subjects that don't go out of date: fundamental software principles & practices, team interactions, and technology-independent skills.

Current and planned titles in the *Conflux Books* series include:

1. *Build Quality In* edited by Steve Smith and Matthew Skelton
2. *Better Whiteboard Sketches* by Matthew Skelton
3. *Internal Tech Conferences* by Victoria Morgan-Smith and Matthew Skelton
4. *Technical Writing for Blogs and Articles* by Matthew Skelton

Conflux Books also publishes the acclaimed *Team Guides for Software* collection: *Software Operability*, *Metrics for Business Decisions*, *Software Testability* and *Software Releasability*.

Discover the *Conflux Books* series: confluxbooks.com

conflux

3. Acknowledgements

We've learnt and taken huge inspiration from many experts and practitioners whilst writing this book. We would like to thank the following people for their help and inspiration (in alphabetical order): Alexei Zheglov, Andy Carmichael, Bazil Arden, Dan 'KanbanDan' Brown, Dan Vacanti, David J. Anderson, David Lowe, David Shrimpton, Dimitar Bakardzhiev, Donald Reinertsen, Emily Webber, Gaetano Mazzanti, H□kan Forss, Jo□o Miranda, Johanna Rothman, John Cutler, Karl Scotland, Larry Maccherone, Melissa Perri, Mike Burrows, Pawel Brodzinski, Prateek Singh, Robin Weston, Sam L. Savage, Seb Rose, Simon Machin, Steve Smith, Troy Magennis, Vasco Duarte, Woody Zuill.

We also give a big "thank you" to our reviewers: Amy Phillips, Clare Sudbery, Don Maclellan, Heidi Helfand, Helen Meek, Nick Brown, Stefania Marinelli, Stefano Luzi Crivellini, Victoria Morgan-Smith, Peter Marriott.

We owe a great deal of gratitude to our publisher, Matthew Skelton, and our editor, Manuel Pais, for their guidance and patience.

Mattia would also like to thank his family - Aindrea, Anthony and William - for their support and patience over the many days and nights that he spent working on this book, and the people in the NIM team at Sky, who over the years have inspired and supported many of his experiments.

Chris too would like to thank his family - Toni-Marie, Elco and Louie - for their support, Kate Gray for being a great friend and mentor and James Carpenter, Craig Russill-Roy and Andrew Fox from Honeycomb.

– Mattia Battiston and Chris Young

4. Praise for *Metrics for Business Decisions*

"I love this book. It gets straight to the point providing easy to understand, and practical guidance on how to use data to answer all the commonly asked questions. Questions such as "How do we know this works?" and "When will it be done?'. Highly recommended to anyone who works with a team looking to improve their ability to build and operate software systems."

– **Amy Phillips**, Engineering Manager at Gousto

"A beautifully written book about metrics that answers real key questions that teams face daily. With clear explanations, examples, myths and getting started tips this should be anyone's go to book for making metrics easy and informative."

– **Helen Meek**, Coach, Trainer & Consultant

"Despite my 20 years of Agile experience the book surprised me with a brand new and fresh perspective on the metrics matter! Each topic is thoroughly covered and explained with great clarity, nothing is taken for granted, and many false myths will be dismantled.

You will understand deeply the essence of 'Why' and 'How' a metric should be adopted and 'Where' to focus on to get real benefits like eliminating bottlenecks, making good decisions and providing realistic forecasts. This book is a 'must' for your Agile library!"

– **Stefano Luzi Crivellini**, Chief Delivery Officer at Creactives S.p.A.

"Software development teams need to go beyond estimation in order to forecast when work will be done. They also need to prove that

their software is valuable for customers. Mattia and Chris's book is an excellent guide for how to have data-driven conversations, and it provides many tools and resources for success."

– **Heidi Helfand**, Author of *Dynamic Reteaming*. Director of Engineering, Excellence at Procore Technologies

"Mattia and Chris guide us through the questions that we've all been asked: How fast are we going? How long will this take?

Full of real examples and suggestions they share their experience in a direct and actionable way. A book to read and to have on our desk to take decisions based on better conversations generated by data."

– **Stefania Marinelli**, Agile Manager at Hotels.com

"I love that this is written specifically for team members, rather than tech leadership - there are questions that all teams need to ask themselves about the flow, quality and value of their work. The clear examples and explanations in here will spark curiosity in any team member and equip them to use their own data to help them focus on what matters most."

– **Victoria Morgan-Smith**, Director of Delivery, Internal Products at Financial Times

"Team Guide to Metrics for Business Decisions feels like a long overdue publication that has been missing from the Lean-Agile world. Whilst there can be no doubt about the wealth of useful material on metrics that is out there for practitioners to consume, I've always felt like there was something missing. … Through reading this book it's safe to say my opinion has changed.

I would consider this book essential reading for people new and/or experienced in the lean-agile world. The questions it answers, the perspectives it gives and stories told make it a book with great learnings and practical guidance for everyday application in your organization."

– **Nick Brown**, Agile Lead (IFS) at PwC UK

5. Index of Case Studies

6. Introduction

"Why should the team care about metrics?"

How can software teams keep a laser-like focus on outcomes whilst improving practices and flow? How can software teams understand where the bottlenecks are in their processes? And how can software teams provide reliable forecasts of when work will be done? The answer to these questions is: software teams should use Business Metrics. Teams that use Business Metrics understand Throughput, Lead Time, Forecasting, Flow, Quality, and Value - all measures that speak directly to business outcomes. By using Business Metrics, your software team will produce software that is more business-relevant with more certainty and less waste.

6.1 What is covered in this book

Our goal in this book is to equip teams with the means to help them **measure the value and quality of their work together with the time and resources required to get the work done**.

By doing this we hope to make it easier for teams to work on the things that matter the most, both to the team and the wider business or organization for which they work, and do so at the right time.

To be able to measure you need metrics. Metrics give teams a shared, tractable, resource which can be poked and prodded, questioned and explored. From these metrics come conversations and insights that we have found help teams succeed.

What we are not trying to do is tell teams how to do the work. This is not a methodology book. It's not about how to use Kanban, Scrum or whatever your preferred way of working is. Rather it is about how to instrument the work and its outcomes in order to make both better for everyone involved.

6.2 All the time in the world

> Time is an executive's scarcest and most precious resource. And organizations - whether government agencies, businesses, or nonprofits - are inherently time wasters.
>
> – Peter F. Drucker The Effective Executive

As we and our teams build and run software we are continually asking and being asked questions:

- How much do we have left to do this week/month/year?
- How much time do we have to finish this?
- How long does it take us to deliver things?
- How much does it cost per month to operate our software?
- What should we do next?

These questions all have one thing in common. They are about our use of and the value of time. To help us answer these questions we need to measure. And we must always measure over time.

For example, say we are running an e-commerce website and we measure how many people are using our website at any given time. Just knowing that there are 1000 people using our website right now isn't particularly useful.

If, on the other hand, we know that we usually have 1000 people at a time between nine in the morning and five in the evening but

that this number drops off to only 100 overnight we can use that information to decide how much computing resources we want to put in place at what time of the day.

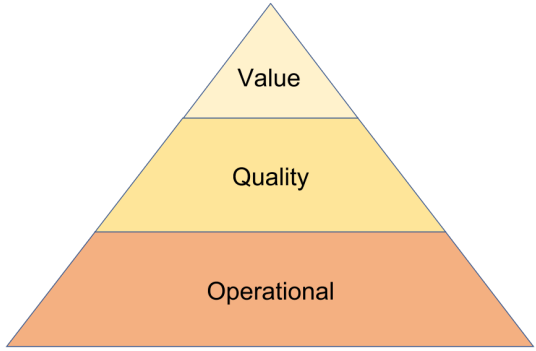
Or, to take another example, we notice that over the past few weeks the number of defects being logged against our system has been going up at a steady rate, from two defects/week to four, then six and now eight defects/week. Then we know we have a trend that we should be worried about.

This is an actionable book primarily for teams as time is the *entire* team's scarcest and most precious resource. Time is wasted if we don't know what we should be working on and when.

We can quantify the cost of delay by measuring the **operational effectiveness** of our work, its **quality** and the **value** it creates.

6.3 Value, quality, & operational metrics

We can split metrics into three tiers based on their purpose. Each tier supports the ones above it:



Metrics pyramid: value, quality, operational

At the bottom we have operational metrics. These are the underlying 'hygiene factors' that indicate the ways of working required to produce the value and quality outcomes required: throughput, flow, variability, and so on. These metrics cast a light into otherwise dark corners and give us situational awareness.

Next comes quality. We want to be able to measure the quality both of the work we produce and also how we produce it (how we operate). The quality of the work we do affects both the operational effectiveness and its value. If our code is full of bugs then this prevents people using it from getting value out of it. Plus we spend more of our time fixing those bugs rather than delivering valuable new features.

At the top is value. This is where we want to end up, being able to measure the value of our work - the outcomes and results it produces. Otherwise, how do we know it is worth doing and spending time on in the first place?

Operational metrics

We start at the base of the pyramid with operational metrics.

We need these to be able to answer questions like:

- How much do we have left to do this week/month/year?
- How much time do we have to finish this?
- How long does it take us to deliver things?

To answer these questions we need metrics derived from how the team works, as opposed to their output. We measure things like:

- Delivery rate - also called throughput. How fast is the team delivering the work?
- Cycle time - also called lead time. How long does it take the team to complete the work?

We suggest that teams start by looking at their operational metrics as these are within the team's *Zone of Control*. They can then be used as an example to those within the team's *Sphere of Influence* (Adzic2014) such as customers, product managers or heads of business.

Quality metrics

Once we can see our operational metrics, we are in a position to look at the quality of the work we are doing. Like operational concerns, quality is within the team's zone of control. As David Anderson put it "[quality] is a technical discipline that can be directed by the function manager" (Anderson2010). It is something that as a technical or engineering team *you* own.

Raising the quality of your work - "getting your own house in order" - puts you in a good position to have fruitful conversations with your customers and collaborators, and is a way of strengthening trust with them.

We need to be able to answer questions like:

- Does our software consistently do what people expect it to do?

- Is our software available at the times and in the places where people need it?
- When things go wrong with our software, what does it take to get them fixed?

A way of qualifying, quantifying and then measuring quality is to think of it in terms of what John Seddon calls "failure demand":

> Value demand is 'demand we want', demand that the service is there to provide. Failure demand is demand caused by a failure to do something or do something right for the customer.
>
> – John Seddon "Systems Thinking in the Public Sector"

For a team building and running software, failure demand is caused either by the defects in the software put into production or by building something that doesn't fulfil the customer needs, that is something that does not provide value.

Value metrics

In our experience there is rarely a shortage in the supply of ideas for things to do with a team's scarce time. We need to know which of those are going to create value.

This means we need to be able to answer questions like:

- What should we do?
- When should we do it?
- What should we do next?

To answer this we need some measure of the worth of what we seek to achieve and the degree to which our actions are having the desired outcome. These are our *value metrics*.

For example:

- How many customers have we gained this month?
- How much are they spending?
- Which features are they using the most?
- How long do people stay with us as customers?
- Do they recommend us to their friends?

Value metrics help to provide a common goal for the team by emphasizing the purpose and outcomes of the software, reducing the time needed for decision making and avoiding time wasted on interesting but less valuable work.

6.4 How to use this book

The metrics we generate are not an end in themselves. They are there to answer questions and to enable the team to make better business decisions. Each chapter of this book poses one of the questions we hear the most when working with teams:

- Chapter on Throughput - How fast are we going?

In the fast-paced world of software development we have found that it is all too easy to lose track of how much you can actually get done in say a week or a month. When demand from your customers is piling up, by looking at the operational metric 'Throughput' you can get a handle on how well balanced your capability is to deliver against this demand.

- Chapter on Lead Time - How long will this take?

How long is a piece of string? Software is complex and requirements are emergent. Both customers and the team want to know how long a piece of work is going to take to complete. By looking at the operational metrics "Lead Time" and "Cycle Time" we show you how to answer this perennial question.

- Chapter on Forecasting and planning - When will it be done?

Now we're motoring. We have a predictable delivery rate and can give our customers realistic expectations of when individual pieces of work will get done. When it comes to bigger chunks of work though, like epics or projects, questions remain: When will it be done? How much can we do by a particular date? In this chapter we look at how to make forecasts that help us take those decisions with confidence, dealing with uncertainty and highlighting risks early on.

- Chapter on Flow - Where's the bottleneck?

As we look at queues of work alongside work that is in progress we will see that some of the queues are longer than others. Where these 'bottlenecks' appear tells us a lot about how we are working and what we can do to improve it.

- Chapter on Metrics for Quality - How do we know this works?

There is no point knowing how quickly we can deliver 'stuff' if that stuff doesn't work. In this chapter we look at the concept of "Failure Demand". We see how we can measure it and look at ways to remove it from the equation.

- Chapter on Metrics for Value - Why are we making this software?

Value metrics help us to answer the question "Why are we doing this?". You could even decide to 'Start with Why' as Simon Sinek famously said (Sinek2011) and read this chapter first. The other chapters are all about "getting the house in order" - doing a great job. Value metrics help us to take this practice of instrumenting

and reflecting upon our work and apply it to the outcomes and purpose of the work itself. We can start measuring the impact of our work: by choosing to measure impact we prompt conversations about purpose ("why?").

6.5 Get Started!

Each chapter is readable independently, containing the necessary level of detail to be understood and actionable on its own, without requiring any of the other chapters in the book to be read first (although certainly reading the full book will provide a more comprehensive understanding of the concepts and practices and their inter-relations).

We've structured the book in an order that is intended to take you and your team from a practice with no metrics to one where metrics are driving your day-to-day decisions about the work.

That doesn't mean you have to read it linearly cover to cover. This is a book for teams and we suggest you start with a conversation with the other members of your team about what is and what is not working for you.

Ask them the questions above - 'How fast are we going?', 'What are we doing next?', 'Why are we making this software' etc. - and see how much interest you get and what answers people have. Which question do they most want answering? Our thesis in this book is that these questions are best answered by using metrics. By working through the chapters with your team you should be able to discover more about the work you are doing and better answer the questions.

6.6 Feedback and suggestions

We'd welcome feedback and suggestions for changes.

Please contact us at:

- email: info@bizmetricsbook.com
- Twitter: @BizMetricsBook
- Leanpub: leanpub.com/metricsforbusinessdecisions/feedback

Mattia Battiston & Chris Young

7. Throughput: How fast are we going?

Key points

- **Most teams have a need for predictability**, they need to answer questions like "How much work can we complete? How long will it take?"
- **Story points and velocity often give us little predictability**
- **Use throughput** (number of work items completed in an iteration) **to know how fast the team is going**
- **Use throughput to improve planning and forecast** how much work can be completed in the next iteration
- **Check the throughput trend to know if the team is improving**

This chapter focuses on **throughput**, the number of work items that were completed during a time period.

7.1 Why should I care about throughput?

As a **team member**: throughput helps you become more predictable, so you are able to make more realistic commitments. You

have data to fight the pressure to fit more work into the available time and thus avoid death-march projects (DeathMarch).

Using throughput you also spend less time estimating, which leaves more time for other work. Meetings should get faster as decisions become less based on personal opinions and more on data.

As a **Scrum master** or **coach**: throughput helps you check if the team is improving and validating the effectiveness of experiments. Facilitating planning meetings and estimation sessions becomes much easier as you focus on the content of the stories rather their size.

As a **product owner**: throughput gives you realistic expectations of what the team can and can't achieve. You start thinking in terms of "What are the chances that we can fit this story in the next couple of weeks?" rather than relying on the team promising that "they'll try".

As a **manager**: throughput helps teams make realistic promises so you'll know when they're likely to start/finish work. You'll have data to play what-if scenarios.

As a **customer**: throughput gives the team the ability to make you realistic promises. Disappointments become the exception rather than the norm. You can be confident that when the team says they'll get something done, they probably will.

7.2 What's the problem with velocity and story points?

This chapter is all about answering questions like "How fast are we going?" or "How much work can we complete?". Typically Agile teams use story points and velocity to come up with an answer.

Story points are a measure of the perceived size of a user story. For example 'Copyright on webpage footer' might be considered small

and thus allocated a single story point, whereas 'Enforce password strength policy' might be considered large and so be given ten story points.

Teams then apply their *velocity* - that is the number of story points per sprint the team can deliver - to decide how many stories they can include in the next sprint. They might also use velocity to determine how many sprints it will take to complete a given set of user stories (Cohn2016 Cohn2014).

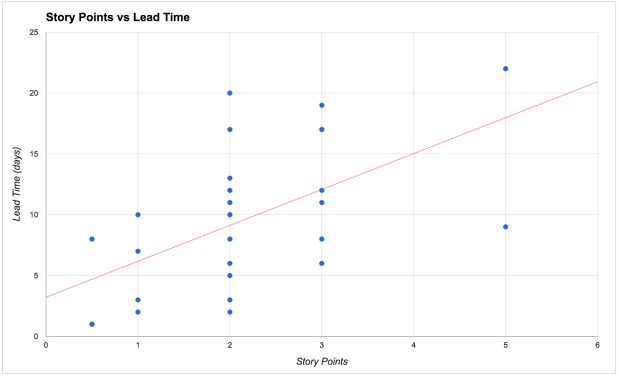
These practices are so common amongst Agile teams that they are considered the norm. However, it's not all roses: let's look at some of the problems with this approach.

The problem: low correlation with lead time

When Mattia first joined his team in late 2013, they were playing planning poker (PlanningPoker) and estimating using story points. The team was used to it and had been doing that for a long time. Mattia wondered if their estimates were making sense and if they were at all predictable, so the team started to gather some data. Little did they know that they were in for a big surprise…

The chart below shows the results found by Mattia and his team. In particular, for each story, we can see the relationship between estimated story points and how long it actually took to complete the story (in other words, the story's lead time).

Mattia and his team were absolutely shocked to discover that 1-point stories were taking between two and ten days, 2-point stories were taking between two and twenty days, and 3-point stories were taking between six and eighteen days! There was very low correlation between their estimated story points and the actual lead time.



Very low correlation between story points and actual lead time

Such low correlation meant the estimates were adding very little value: the team really had no meaningful way of predicting how long a story would take based on its estimated points. Consequently, their velocity made little sense and the team was highly unpredictable.

The team therefore decided that it wasn't worth it to continue investing time estimating using story points anymore. Instead, they started to base predictions on historical data and the metrics described in this book.

In particular, throughput proved to be the metric that could answer their questions in a way that was easier, cheaper and more accurate.

What causes these problems?

There are three important factors that have a much higher impact on lead time than user story size, and that when left unmanaged make our teams throughput unpredictable.

First, do we have a **high amount of work in progress (WIP)**? When we work on too many things at the same time we are not able to focus on finishing the tasks that are already in progress. We waste time in context switching, the quality of our work decreases,

and even stories that appear to be simple end up taking longer than expected.

Second, are the **queues in our process** visible? How long does work sit in those queues? Very often in our processes there is some waiting time between one activity and another. For example: waiting for a developer to be free to start a story, waiting for someone to be free to test a story, waiting for the next release, etc. These queues are often invisible, they're not represented on our boards, and it's really common to ignore them when we estimate. When these queues are not managed they lead to a lot of work in progress but on hold, which in turns lead to high unpredictability.

Third, are **unexpected events** common? Events like being blocked by a broken test environment, having to clarify requirements, or 'urgent' problems such as a defect in production, make us interrupt and put on hold activities that have already been started.

When we have one or more of these problems - and most teams we've worked with certainly seem to - then the estimated story size doesn't matter as much as it has a very low correlation with the actual time taken to complete the work.

Even the simplest story is going to take ages if we're not focused, if we waste time in numerous hand-offs, or if it's constantly overtaken by more urgent activities. For more details on solving these problems see the chapter on Forecasting and planning.

Are story points bad?

The use of story points, or rather the non-use of them, has been a controversial topic in the Agile community. So let us be clear: by no means are we trying to discredit story points in this book. For many teams and companies they have represented a tremendous step forward from the past, and they're still using them successfully.

What we **are** saying is that if you recognize some of the problems described above, then chances are that story points are not giving

you much predictability.

Several people in the community have found similar results to ours:

- Vasco Duarte, one of the main proponents of #NoEstimates, found high correlation between story points and number of completed stories, suggesting that they're measuring the same thing (Duarte2012).
- Larry Maccherone analyzed data from 1000s of teams and found that throughput was the best measure for productivity and predictability (Maccherone2014).
- Ian Carroll found low correlation between story points and lead time for 25 teams (Carroll2016).
- Folks at ThoughtWorks found that they were getting the same predictability using story count (throughput) or story points (ThoughtWorks2013).
- Nader Talai presented very similar results, where throughput gave his team the same predictability as story points (Talai2014).
- Pawel Brodzinski, a bit provocatively, created a new set of planning poker cards where the only values are "1", "Too Big", "No Clue". For him the only interesting question is whether the story is small enough, without needing to estimate whether it's a 5 or an 8 points story (Brodzinski2015a).

Should I stop using story points and velocity?

By all means, if you're finding story points and velocity useful in your team please keep using them! What we invite you to do however is to ask yourself and your team: "Are our estimates really working?".

Whatever estimation process you use at the moment - story points, t-shirt size (Singh2016), ideal days (Rao2014), or others - generate

a chart like the one above. Is there a correlation between your estimates and the time taken to complete your stories? You might be surprised by the results.

Our advice is: don't drop whatever estimation process you're currently following, but start collecting data and generate some metrics. After a while you can compare your estimates and your metrics and decide what makes you more predictable. You should be able to start using your data to improve your estimates.

And if at some point you end up deciding that you want to drop story points, then having team data in your hands will make it a lot easier to convince the people around you.

7.3 What is throughput?

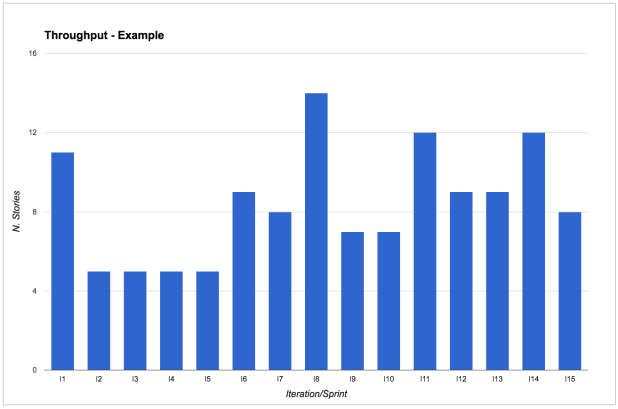
Throughput (often also called delivery rate, or story count) is the number of work items that were completed during a particular period of time.

Throughput is extremely simple to calculate and provides high value, that is why we think it makes for a great starting point.

How do I calculate throughput?

It's extremely simple: just count the number of work items, such as user stories, that have been completed in a fixed period of time. The period could be a two-week sprint, a week, a month, and so on. The important thing is to keep it consistent. If you do Scrum, or work with iterations, the easiest thing is to make it coincide with the length of your sprint. Otherwise, just pick a cadence that works for you.

For example, say our team chooses a two-week period. Then every two weeks we count how many stories have been completed in that period of time.



Example throughput - Number of stories completed per sprint

Tips for calculating throughput

If you use a *physical board*: you can simply count the number of cards that have reached your "Done" column (or whatever you consider as completed, according to the policies of your process). After you've counted the stories you might want to clean your "Done" column, so that next time it will be easier to see what has moved there since.

If you use an *electronic board*: some tools let you extract this information. If yours doesn't, you can quite easily calculate the throughput just by knowing the end date of each work item. Simply calculate what iteration/sprint that end date falls into. For an example, have a look at our public repository (https://github.com/SkeltonThatcher/bizmetrics-book) or at Troy Magennis' collection of spreadsheets (MagennisSpreadsheets).

Split by work item type

We recommend splitting throughput by *work item type* in order to know how many of each type we usually get done in a given period.

This will enable us to make much better predictions (more on this in chapter 6).

Different work item types are usually characterized by:

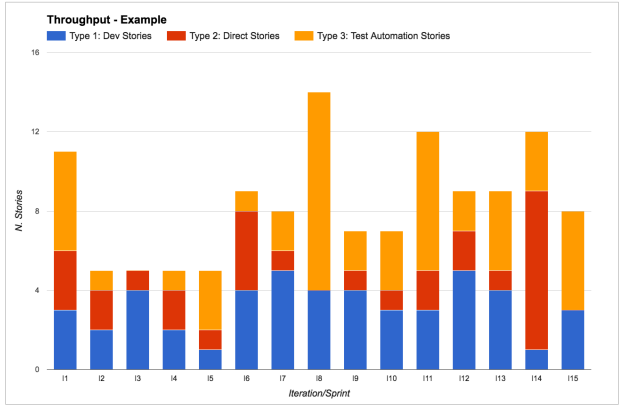
- Whether they follow a *different process* (for example, some user stories might not need to be tested, whilst some have to go through a special test environment)
- Follow the same process but at a consistently *different speed* (for example, work we do in our legacy codebase is consistently much slower than work on the greenfield app).

Don't go overboard though: typically we'd expect teams to only have a handful of work item types.

In Mattia's team they use:

- "Dev Stories" to represent "normal" work. These stories go through each step of the process before being "done" - they are analyzed, developed, tested, and then released.
- "Direct Stories" to represent work that goes straight from "development" to "done", like writing a report, investigating a support issue, responding to a team email. These stories follow a different process as they skip the test and release steps, and therefore have a much lower lead time than "Dev Stories".
- "Release Testing Automation" to represent work that the testers are doing to automate their manual regression tests. These stories follow the same process as "Dev Stories" but are usually much quicker.

Most metrics only make sense when analyzed for a particular work item type:



It's useful to split the throughput by work item types to make more accurate predictions

For example, say that we are working in iterations and we're dedicating one sprint to stories on a new technology that we're not familiar with. We don't have historic data for these, so we're unsure how long it will take. Our gut feeling is that it will take longer than usual, thus the throughput in this sprint will be quite different from other sprints.

It's important to have the data to acknowledge that the throughput change was due to this new type of work. We can then decide if we should treat this sprint as an outlier or if we need to revisit our forecasts instead.

Case study: differentiating work items

João Miranda, Engineering Manager at a large European bank, writes:

We went agile 7 years ago, starting with a small number of teams going all-in for Scrum as our base framework. These teams were formed to execute strategic projects, so they had

full attention from their (internal) clients and did not have to deal with production issues (maintenance teams would handle those).

Once we started scaling out Scrum to all the development teams (about 70), we faced new challenges. Today development teams have multiple responsibilities: new features; maintenance and bug fixing; refactorings and optimizations; urgencies (unplanned work). Balancing all these needs while keeping the sprint scope stable is a hard problem.

The approach we took, which serves us well to this day, is simple. We split each team's available capacity into bands. A band is a percentage of the available effort in a sprint. The "new features" band includes all scenarios that provide new capabilities to users, regardless of feature size. Enabling a new loan product, even if it's just some parameterization in a database? New feature. The "maintenance and bug fixing" band includes all fixes in production and configuration changes (for example changing loan rates). The "refactorings and optimizations" band includes all the work the team does to explicitly reduce the size of the "maintenance and bug fixing" band (in short, paying down technical debt).

Finally, "urgencies" is an explicit buffer to handle all unexpected activities that happen during a sprint. In this way, we help the team to avoid overcommitment and keep the sprint plan mostly stable.

The allotted size of each band is unique per team and may change if the team's dynamics changes. For instance, if a team has a lot of maintenance work, chances are its "maintenance and bug fixing" band is larger than average. But its "refactoring and optimizations" band will also be larger so that the team can bring back things under control.

We don't have any default or baseline for the size of each band, context is king.

7.4 Use throughput for short term predictions

"How many stories can we complete in the next sprint?"

"How much can we get done in the next two weeks?"

These are common questions most teams get asked during sprint planning (in Scrum), queue replenishment (in Kanban), or simply when talking to stakeholders.

The good news is that the team can answer them by looking at their throughput stats.

THROUGHPUT STATS

	Total	Type 1: Dev Stories	Type 2: Direct Stories	Type 3: Test Automation Stories
Average	8.4	3.2	1.9	3.3
Median	8.0	3.0	1.0	3.0
Std Deviation	2.9	1.3	2.0	2.6
Min	5.0	1.0	0.0	0.0
Max	14.0	5.0	8.0	10.0
Mode	5.0	4.0	1.0	1.0
5% percentile	12.6	5.0	5.2	7.9
25% percentile	10.0	4.0	2.0	4.5
50% percentile	8.0	3.0	1.0	3.0
80% percentile	5.0	2.0	1.0	1.0
95% percentile	5.0	1.0	0.0	0.7

These stats about our throughput help us make short-term predictions

These stats are telling us a lot of useful information:

- Average (mean) and median: how many stories do we complete during a sprint, typically?

  (by the way, we usually prefer to look at the median rather than the mean average as it's less impacted by outliers, so it's usually a more accurate representation of reality)

- Standard deviation: how much variability in throughput have we got?

  (we usually look at its trend to see if variability is increasing/decreasing. The actual value is not important as it's only reliable when our data follows a Gaussian distribution, which is rarely the case in a software development process (Zheglov2014a))
- Min and max: number of stories ever completed.
- Mode: what's the most common throughput?

  (this is the number that occurs most often, so it's the one that people usually remember off the top of their head)
- Probability percentiles: what's the likelihood of completing that many stories in one sprint?

Let's focus on stories of type 1 for a second ("Dev stories" in our example) and look at their percentiles. The 80th percentile tells us that in 80% of sprints we complete at least two "Dev Stories". Similarly, looking at the 50th and 25th percentiles we know that in 50% of sprints we complete 3 stories, and only in 25% of sprints we complete 4 stories. Therefore if we want to have a high level of confidence that the "Dev Stories" we start working on will be completed within two weeks, we should only pull in the next 2-3 "Dev stories".

We could risk it and pull in a fourth one, but the chances of getting it done are very slim - 25% chance, to be precise - so we wouldn't want to make any promises to our stakeholders about that fourth story!

If we now look at type 2 ("Direct Stories"), we know that in 80% of sprints we also complete one "Direct Story" on top of the other "Dev Stories". So we can also safely pull in one of these stories. Similarly, we can confidently expect to complete one or two "Test Automation Stories".

In short, for each story type, we look at the throughput of that type of work in order to make predictions about how many stories we can complete in the next period.

Notice how this approach makes planning sessions a lot faster: since we know that we rarely complete more than four stories, there's no point talking about stories that we won't even look at.

Queue replenishment: "How often should we decide what to work on?"

In our experience working with Scrum teams, a common struggle we see is planning for the next two weeks.

Regardless of the decisions taken during sprint planning, the team's plans often get trumped by new urgent stories, emergencies, and stakeholders changing their mind.

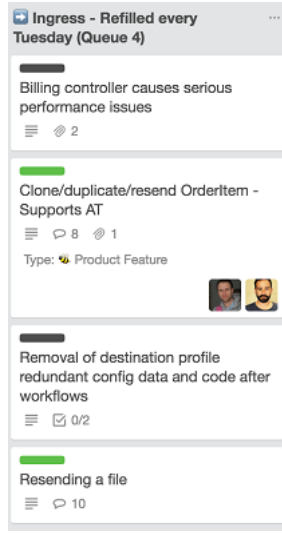
The team feels that they're not reactive to change, and that they should be more flexible changing scope or priorities. That's a symptom that their batch size of two weeks is too large for them, and they should instead plan shorter periods of time. And so they find themselves asking questions like "How often should we do planning? How many stories should we have in our 'Next' column?".

Don Reinertsen (Reinertsen2009) teaches us that reducing batch size is almost always a good idea, but where is the sweet spot? The more often we plan the more flexible we can be, but we also don't want to spend all our time planning, and the people we need to make those decisions might not always be available. Luckily, we can use our throughput to decide how often we should replenish our queue.

Let's have an example: say that our team would like to plan every week instead of every two weeks, meaning that every week we have a chance of reviewing priorities and changing direction. How many stories should we put in our 'Next' column? Well, let's use

our throughput stats: in a week we usually complete three to four stories. So let's have four stories in our 'Next' column, and every week we will replenish it.

At Honeycomb TV the throughput of the technology team, which uses Kanban, is such that a queue of four items empties out once a week. Hence every Tuesday a Google Hangout takes place where the technology team and its customers, the wider business, gather around the Trello board and decide what to put into the queue next.



Queue Replenishment

In the screenshot above there are two pieces of value work - that is features (green label) - and two pieces of operational overhead (black label).

When deciding how often to replenish the queue, you need to take into consideration the product owner's availability. If the product owner is needed for planning but is only available every three days that means that we can't possibly replenish more frequently than every three days. We'll size our queue based on how many stories we usually complete in three days.

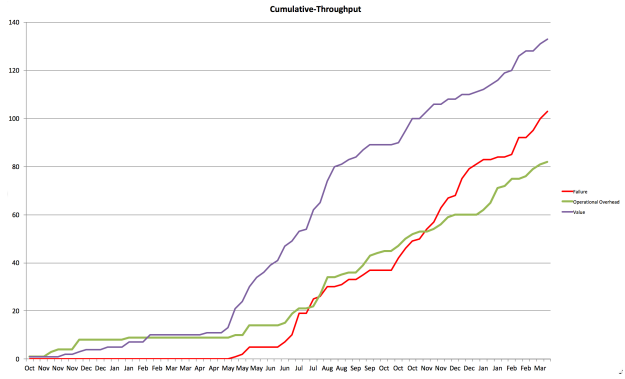
Queue replenishment: "Are we reducing technical debt?"

We can use throughput to analyze the percentage of effort we spend on work items of each type. A typical usage would be to make sure that we are working on technical debt items: we can agree as a team on a percentage of capacity to reserve for technical debt, and honor it at queue replenishment.

For example, Chris's team realized that they were spending too much effort fixing defects, taking away time from other value-adding activities. Recently they had been focusing too much on adding new features and weren't doing enough to keep technical debt under control, resulting in a high number of issues.

They decided on a a new team policy: whenever they were pulling in new work, for every three business stories they would pull in a technical story.

Using this policy made sure that the team wasn't pressured into working only on business work and that technical debt was being addressed.



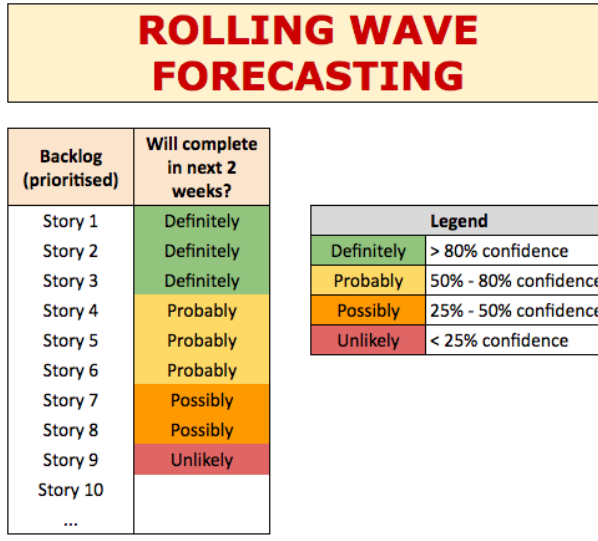
Cumulative Throughput

Visualizing predictions with rolling wave forecasting

A nice way to visualize our predictions is by using a technique called "rolling wave forecasting", introduced by Vasco Duarte in his book "No Estimates" (Duarte2015).

We visualize the stories in our backlog using different colors. Each story gets a different color based on how confident we are that they will be completed in the next two weeks:

- green = definitely
- yellow = probably
- orange = possibly
- red = unlikely



Rolling wave forecasting for visualizing our predictions

These levels of confidence are taken from our data, using the percentiles from our throughput. If the 80th percentile is three

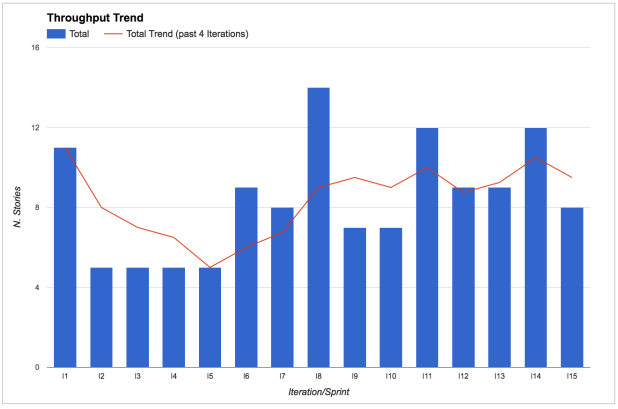
stories, then we're 80% confident that the first three stories in the backlog will be completed in the next two weeks (green). We're 50% confident that we'll complete up to six stories, so stories four to six are yellow. And so on for orange and red.

7.5 Use throughput to validate experiments

"Are we improving?"

"Are we going faster/slower?"

One of the Kanban practices says "evolve experimentally" (AndersonCarmichael2016). Metrics are a fantastic tool to help us do that: we drive our continuous improvement through running experiments, and metrics help us decide whether those experiments are successful or not. Whenever we introduce a change, for example implementing an action from a retrospective, we rely on our metrics to know if the change is helping us improve.



Throughput trend - are we improving?

Throughput is great to know if we're going faster or slower. In particular, we look at the throughput trend by calculating the

throughput average over the past four iterations.

We look at the throughput trend rather than absolute values because continuous improvement is about going in the right direction, rather than hitting a particular target. So as long as the trend is going up, we're improving.

What would an experiment look like?

The key part of an experiment is treating our ideas as hypotheses and setting our expectations before we start: what do we expect to happen after we introduce this change? How and when are we going to know if this experiment has been successful or not?

For example, say that during a retrospective we come up with the idea that we should do more pairing between developers and testers because we think it will reduce the amount of rework due to bugs. Rather than just saying *"let's do more pairing"* we can formulate it as an experiment that might look like this:

Current situation: Several stories need rework due to bugs found during exploratory testing. Average of 0.5 bugs per story.

Our hypothesis: We believe that if developers and testers paired more, developers would learn how to prevent those bugs, resulting in less rework, and therefore having more time to complete other valuable work.

Action: For the next four sprints, each developer will spend at least one day a week pairing with a tester.

Expected results: Initial slow down in the first sprint due to time invested in learning about testing strategies, then in the last sprints we expect less bugs and less rework, therefore more stories completed.

How to measure our results: In the first sprint we expect throughput to go down of 1 or 2 stories. But in the following sprints we

expect less bugs and higher throughput (extra 1-2 stories) compared to now.

Notice that we're not just checking that the number of bugs goes down, as that could be achieved by simply spending more time on a story. As well as less bugs, we want the throughput to increase to make sure that we're actually improving.

This is just an example structure for your experiments. There are many techniques that can help here, like A3 thinking (A3Thinking) or the Toyota Kata (Rother2009 Forss2016).

Your experiments don't have to be as formal as this, but make sure to talk about what you expect before changing something, and then use your metrics to determine success, in particular analyze your throughput changes.

How do I know if my experiment passed/failed for the right reasons?

In our experiments we do our best to change only one variable at a time, to try and find a cause-effect relationship between our change and the results. Nevertheless, it's impossible to fully isolate an experiment. Real life is complex (as in Cynefin-complex (Cynefin)) and every day there are many things that influence the outcome of our work. So how do we know if an experiment was really successful, and that achieving the expected results was not just a matter of chance?

The best advice we can give is: use multiple metrics. Never rely on a single metric for checking your results, as a single metric is easily influenced by many other factors than your experiment, including people trying to cheat or 'game the system'.

For example, a common mistake is to set a particular velocity as a target. This very often results in the team estimating each story with a few extra points - what used to be a five is now an eight,

threes are now fives, and so on. By focusing on one metric (velocity) the team is unintentionally encouraged to optimize their behavior to reach that target, at the cost of cheating.

Instead, if you look at multiple metrics and observe that most of them have progressed, then it's much more likely that the experiment was the cause. Only when multiple related metrics are progressing are we really improving.

How much variation is normal?

Our teams are made of human beings working in complex environments, so some variation is perfectly normal. Actually, we'd be worried by a complete absence of variation: we'd interpret that as inflexibility and a lack of innovation.

But just how much variation is normal? Only you and your team can know. You're the only ones that know your context and can decide if something is a one-off outlier, rather than the symptom of a problem.

For instance, the "Throughput Trend" figure above shows that from iteration I6 onwards the throughput is oscillating around 8 stories per sprint. However, throughput for iteration I8 was considerably higher (14 stories). The team knew this was an outlier because they had received extra help from other teams. Variation is fine as long as you know what caused it.

Having said that, beware of tampering (Burrows2014): if you react to events too quickly you might be reacting to what is just normal variability, and end up making your system unstable.

Real-life examples

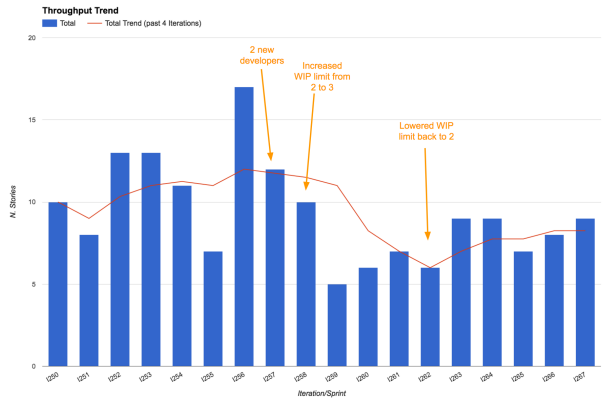
Increase WIP limit to prove that it's a bad idea

We recently had two new members join the team that Mattia works with. That made us six developers. We always pair program, so that

makes three pairs. When they joined the WIP (work in progress) limit in development was set to two, and we purposely left it at two to encourage swarming - we love multiple pairs to work on the same story to get it done faster, whilst also increasing knowledge sharing and removing dependencies from individuals.

The new developers however didn't seem to like this heavily-collaborative style and wanted to increase the WIP limit to three, so that they'd be "free" to work on a separate story. We thought this would be a mistake, but realized that unless we let them experience the impact of the change they wouldn't fully understand the power of a low WIP limit.

So we agreed on increasing the WIP, and guess what? Our throughput went down. We were delivering less stories. Not only that: the lead time went up. Having learnt an important lesson, the new developers were now convinced and we went back to a WIP limit of two.



Real-life experiment: we used throughput to prove that increasing WIP was a bad idea

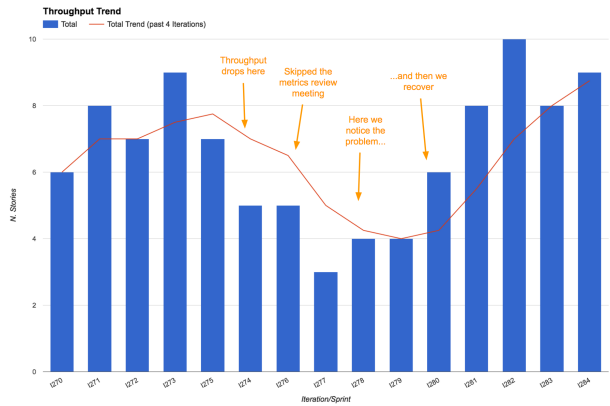
It's interesting to notice that even after setting the WIP limit back to two our throughput took a while to return to the original values. That's because of the cost of onboarding two new developers. Everyone in the team was going slower to take time to explain

things to them and to let them learn.

Metrics that generate questions, not just answers

Mattia's team was having one of their (usually) monthly metrics review meetings. Because of a very busy period more than two months had passed since the previous metrics review meeting.

We noticed a significant drop in the throughput trend, which initially surprised us. "What could possibly be causing this drop? Does this confirm our feelings?" we asked. As soon as we started discussing it we realized that actually most people were feeling like it hadn't been such a productive period - being very busy didn't result in more work done, and actually some people admitted to having felt frustrated and having lost some motivation as a result.



Real-life experiment: drop in throughput generated questions and surfaced important problems

Thanks to our metrics, a problem that was lying below the surface and intoxicating the team had become apparent, giving us a chance to fix it! We had a few retrospectives about these problems, identified some of the root causes and managed to recover, even increasing our throughput, when compared to before the busy period.

Remember, good metrics should surprise you every now and then. It's important that they generate questions as well, not just answers!

7.6 Use throughput for long term predictions

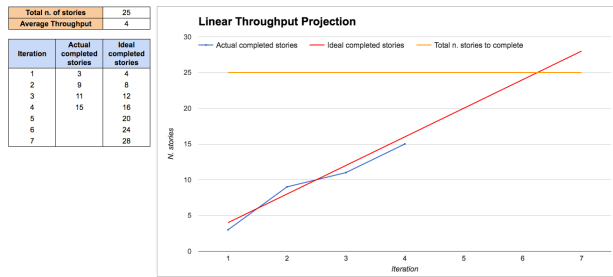
"How long will it take?"

"When are we going to complete this project/functionality/group of stories?"

Suppose we're analyzing a new project or a new piece of functionality and we need to answer the perennial questions "How long will it take? When will it be ready?".

If a simple back-of-the-envelope answer is enough for us then we can use the throughput average to make a simple linear projection, in a burn-up chart style.

Bear in mind that using an average for making plans or predictions can be very dangerous (the so called "Flaw of Averages": Plans based on average assumptions are wrong on average (Savage2012)). But if all you're trying to do is get a rough idea of duration then this could be enough.



Total n. of stories	25
Average Throughput	4

Iteration	Actual completed stories	Ideal completed stories
1	3	4
2	9	8
3	11	12
4	15	16
5		20
6		24
7		28

Simple linear projection using the throughput average

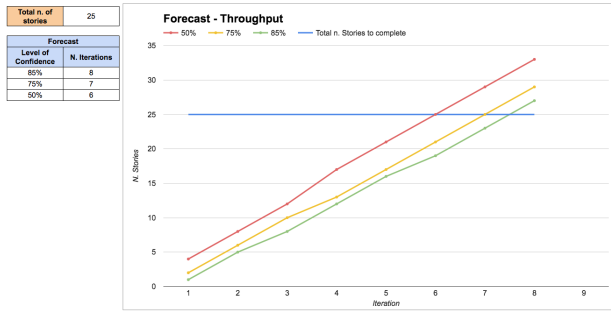
This approach assumes that you know how many stories you will need to complete, which is not always easy at the beginning of a

project. There are some techniques to help you with this that we cover in chapter 6.

If you want more precision and, more importantly, if you need to manage the risks of the project, it's better to use throughput as an input to do probabilistic forecasting.

Probabilistic forecasting is explained in detail in the chapter on "Forecasting and planning". Briefly, it works by extracting random throughput samples from our historical data in order to simulate how much work the team will complete in each iteration. This is called a Montecarlo simulation.

The results are expressed as a range of probabilities: we're 85% confident that we can complete this group of stories in eight iterations, but we're only 50% confident that we complete it in six iterations.



Use throughput for Probabilistic Forecasting and express results as levels of confidence

Real-life examples

Making predictions on the fly

Mattia and his team were discussing a new piece of functionality that would support the launch of a new product. Our business analyst (BA) presented the problem to the rest of the team, and we discussed possible solutions.

After agreeing on one solution, we broke it down into stories. Then the BA asked the perennial question "How long will this take?". We grabbed our throughput stats that we keep printed by our Kanban board, and quickly answered: "Well, we have ten stories here and our average throughput is three stories per iteration. So it looks like it would be roughly something around three to four iterations. Is this enough to answer your question? If not you can give us five minutes, and we'll plug these numbers into our forecasting spreadsheet to provide a more accurate forecast".

The BA replied "No don't worry, that's fine already. I just wanted to know if this would be ready for the launch of the new product in 6 iterations time. Sounds like we'll be fine". The meeting was over in less than an hour and we got started on that new functionality straight away.

7.7 Debunking some myths

"This only works if I have a lot of data"

You can start with a lot less data than you think:

- With 5 samples we are confident that the median will fall inside the range of those 5 samples, so that already gives us an idea about our timing and we can make some simple projections (see "rule of five (Vacanti2015)").
- With 11 samples we are confident that we know the whole range, as there is a 90% probability that every other sample will fall in that range (see the "German tank Problem (GermanTankProblem)"). Knowing the range of possible values drastically reduces uncertainty.

After that, each extra sample helps to further refine our precision. But a little data is enough to start. Also, when you have no data at

all, even a small amount of data is a great improvement and can put you in the right ballpark.

"This only works if all stories have the same size"

This is a myth. It doesn't matter if stories have different size. There is only one size that we care about: "small enough". As long as we split stories as small as possible, then it doesn't matter if they have different size, and we simply use throughput to make predictions.

First, as discussed in the story points paragraph, the size of a story has little correlation with its lead time. So even if you think two stories have the same size, they can take a very different amount of time.

Second, the lead time for stories of the same work item type will follow a known distribution, regardless of their size.

"This is easy to cheat, I'll just create a lot of tiny stories"

Well, yes, but in this case that's actually a useful side effect. Smaller stories have a number of benefits, so encouraging that behavior is not a problem.

There is a point where stories become so small that the overhead of managing so many stories becomes greater than the benefits, but the team should be mature enough to realize when that's happening.

7.8 Public resources

In this book's public repository (https://github.com/SkeltonThatcher/bizmetrics-book) you can find some useful resources for putting what we've talked about into practice:

- Link to a public Trello board that demonstrates how you can track information for your stories, including sprint when they were completed.
- Link to example spreadsheets that show how to analyze your data and generate all the throughput information that we've talked about.

7.9 Get started!

- Initially, don't drop whatever process you're currently following, but start collecting data and calculating your throughput. Simply count how many stories you have completed in each sprint.
- Use throughput at the next planning session to predict what you can achieve in the following sprint. Compare it with your usual velocity-based estimates.
- Use throughput for the next project to predict how long it will take. Compare it with your usual velocity-based estimates.
- Use throughput in the next retrospective to discuss if you're improving. Does it spark useful conversations?
- Compare your story points and throughput data and decide what gives you better predictability. If you decide to stop using story points, having data is going to be extremely helpful when it comes to convincing skeptical people.

7.10 Summary

Most teams have a need for predictability. We need to answer questions like "How much work can we complete? How long will it take? Are we nearly there yet?". Using the team's **throughput** we know how fast we're going and we can answer those questions.

By looking at our past throughput we can predict with confidence how much work we can complete in an iteration. The team becomes more predictable, making planning a lot easier.

Keeping an eye on throughput we can validate our experiments and decide whether the team is improving or not.

8. Lead Time: How long will this take?

(This chapter is not available in this edition of the book.)

9. Metrics for Quality: How do we know this works?

(This chapter is not available in this edition of the book.)

10. Forecasting and Planning: When will it be done?

(This chapter is not available in this edition of the book.)

11. Metrics for Flow: Where is the bottleneck?

(This chapter is not available in this edition of the book.)

12. Metrics for Value: Is this worth doing?

(This chapter is not available in this edition of the book.)

13. Terminology

AARRR Metrics: Acquisition, Activation, Retention, Referral, Revenue - a set of metrics defined by Dave McClure in 2007 to determine whether online services are performing well as a whole.

BDD: *see Behavior-driven Development*
Behavior-driven Development: an agile software development practice that encourages collaboration among developers, QA and non-technical or business participants in a software project. Popularized by Daniel Terhorst-North from 2006 onwards.

CFD: *see Cumulative Flow Diagram*
Cumulative Flow Diagram: metric to observe flow in our team and spot impediments, typically implemented by plotting the amount of work in each state of our process over a period of time.

Estimating: in the context of this book, it's the common style used by teams to predict how long a piece of work is going to take, relying on people's informed best-guess. Usually based on techniques like story points, velocity, t-shirt size, ideal days, or similar.

Flow: movement and delivery of customer value through a process (Vacanti2015). It's how consistently and continuously work moves through our process all the way to a customer.
Flow Efficiency: percentage of time that work spends being actively worked on, as opposed to the total amount of time the work takes, including wait time. A high flow efficiency means that work flows through our process as fast as it possibly can, with minimal wait time or delay.
Forecasting: the prediction of work completion, as well as many other business questions, based on the use of historical data to

simulate what might happen in the future. The results are expressed as a list of possible outcomes with their likelihood.

Lead Time: the time that a piece of work takes to go from start (commitment point) to end (last state of influence) of a process. Note: we are using the names that seem to be most common in the Kanban community, but be aware that different people use different terms when meaning the same things. Most famously, Dan Vacanti calls it "cycle time" in his excellent book "Actionable agile metrics for predictability" (Vacanti2015).
Little's Law: states that $averageLeadTime = \frac{averageWIP}{averageThroughput}$. In simple terms it means that, on average, if you want stories to go faster (shorter lead time), you either have to increase the throughput (get more done) or reduce the amount of work in progress (work on less things at the same time).

Metrics: in the context of this book, the measure of some aspect of the team's work. They help us gather insights, answer questions and enable the team to make better business decisions. We split them into value, quality and operational metrics.

OEM: Original Equipment Manufacturer.

Quality: conformance to requirements, as defined by Philip B. Crosby's book "Quality is Free" (Crosby1987).

Pirate Metrics: *see AARRR Metrics* **PO**: Product Owner.

RAS model: Reliability, Availability, Serviceability - model developed by IBM (RAS) to look at different dimensions of quality.

Story Points: a measure of the perceived size of a user story. For example, something that is perceived as simple and quick to implement might be allocated a single story point, whereas something complex and long might be given ten story points.

Throughput: the number of work items that were completed during a particular period of time (e.g. in a sprint, or an iteration). Often also called delivery rate, or story count.

Value: why we do what we choose to do. Metrics for value help us focus on the things that matter, allowing us to not waste time and do something worthwhile.
Velocity: number of story points that the team completes in a sprint. Often used to make predictions about what the team can deliver, e.g. to decide how many stories they can include in the next sprint, or to determine how many sprints it will take to complete a given set of user stories.

WIP: *see Work In Progress*
Work In Progress: any piece of work that has been started but has not been completed yet. High amounts of work in progress typically results in a number of problems, including slower progress (Little's Law), more time wasted in context switching, and lower quality.

14. References and further reading

Due to limitations of the LaTeX text processor used by Leanpub to generate the book from the manuscript files, some of the URLs in this section may not display correctly. A full list of references is available on the book website.

Visit BizMetricsBook.com for details.

14.1 Introduction

Adzic2014 - Gojko Adzic, 2014 'Zone of control vs Sphere of influence' 12-09-2014. [Online] https://gojko.net/2014/09/12/zone-of-control-vs-sphere-of-influence/ [Accessed 04-Feb-2018]

Anderson2010 - David J. Anderson, *Kanban: Successful Evolutionary Change for Your Technology Business*. Blue Hole Press, 2010.

Sinek2011 - Simon Sinek, *Start With Why: How Great Leaders Inspire Everyone To Take Action*. Penguin, 2011.

14.2 Chapter 1 - Throughput

AndersonCarmichael2016 - David J Anderson, Andy Carmichael, *Essential Kanban Condensed*. Blue Hole Press, 2016.

A3Thinking - Crisp, 'A3 Template'. [Online] https://www.crisp.se/gratis-material-och-guider/a3-template [Accessed 08-Feb-2018]

Brodzinski2015a - Pawel Brodzinski, 2015 'Story Points and Velocity: The Good Bits' 18-02-2015. [Online] http://brodzinski.com/2015/02/story-points-velocity-the-good-bits.html [Accessed 08-Feb-2018]

Burrows2014 - Mike Burrows, 2014 'The Kanban Survivability Agenda' 20-02-2014. [Online] https://www.infoq.com/articles/kanban-agenda-part3-survivability [Accessed 08-Feb-2018]

Carroll2016 - Ian Carroll, 2016 'No correlation between estimated size and actual time taken' 18-04-2016. [Online] https://www.iancarroll.com/2016/04/18/no-correlation-between-estimated-size-and-actual-time-taken/ [Accessed 08-Feb-2018]

Cohn2014 - Mike Cohn, 2014 'The Main Benefit of Story Points' 09-09-2014. [Online] https://www.mountaingoatsoftware.com/blog/the-main-benefit-of-story-points [Accessed 08-Feb-2018]

Cohn2016 - Mike Cohn, 2016 'What are Story Points?' 23-08-2016. [Online] https://www.mountaingoatsoftware.com/blog/what-are-story-points [Accessed 08-Feb-2018]

Cynefin - Dave Snowden, 'Cynefin Framework Introduction'. [Online] http://cognitive-edge.com/videos/cynefin-framework-introduction/ [Accessed 08-Feb-2018]

DeathMarch - 'Death march (project management)'. [Online] https://en.wikipedia.org/wiki/Death_march_(project_management) [Accessed 08-Feb-2018]

Duarte2012 - Vasco Duarte, 2012 'Story Points Considered Harmful - Or why the future of estimation is really in our past…'

25-01-2012. [Online] http://softwaredevelopmenttoday.blogspot.co.uk/2012/01/story-points-considered-harmful-or-why.html [Accessed 08-Feb-2018]

Duarte2015 - Vasco Duarte, *No Estimates*. Oikosofy Series, 2015. http://noestimatesbook.com/

Forss2016 - Hakan Forss, 2016 'KATA - Habits for lean learning Agile Australia 2016' 20-06-2016. [Online] https://www.slideshare.net/HkanForss/kata-habits-for-lean-learning-agile-australia-2016 [Accessed 08-Feb-2018]

GermanTankProblem - 'German tank problem'. [Online] https://en.wikipedia.org/wiki/German_tank_problem [Accessed 08-Feb-2018]

Maccherone2014 - Larry Maccherone, 2014 'The Impact of Lean and Agile Quantified: 2014' 26-01-2015. [Online] https://www.infoq.com/presentations/agile-quantify (around minute 22) [Accessed 08-Feb-2018]

MagennisSpreadsheets - http://bit.ly/SimResources contains many spreadsheet templates related to forecasting and modelling software projects, maintained by Troy Magennis.

PlanningPoker - 'Planning Poker'. [Online] https://www.agilealliance.org/glossary/poker/ [Accessed 08-Feb-2018]

Rao2014 - Mukesh Rao, 2014 'Estimation Using Ideal Days' 27-03-2014. [Online] https://www.scrumalliance.org/community/articles/2014/march/estimation-using-ideal-days [Accessed 08-Feb-2018]

Reinertsen2009 - Donald G. Reinertsen, *The Principles of Product Development Flow: Second Generation Lean Product Development*. Celeritas Pub, 2009. Chapter 5 in particular.

Rother2009 - Mike Rother, *Toyota Kata: Managing People for Improvement, Adaptiveness and Superior Results*. McGraw-Hill Education, 2009.

Savage2012 - Sam L. Savage, *The Flaw of Averages: Why We*

Underestimate Risk in the Face of Uncertainty. John Wiley & Sons, 2012.

Singh2016 - Vikram Singh, 2016 'Agile Estimation Techniques' 18-01-2016. [Online] https://www.scrumalliance.org/community/articles/2016/january/agile-estimation-techniques [Accessed 08-Feb-2018]

Talai2014 - Nader Talai, 2014 'Does size matter?' 09-04-2014. [Online] http://cdn2.hubspot.net/hubfs/2495218/valueglide_dec2016_files/DOCs/AgileOrganisation_EstimationJourney.pdf [Accessed 08-Feb-2018]

ThoughtWorks2013 - Various authors, 2013 'How do you estimate on an Agile project?'. [Online] http://info.thoughtworks.com/rs/thoughtworks2/images/twebook-perspectives-estimation_1.pdf [Accessed 08-Feb-2018]

Vacanti2015 - Daniel S. Vacanti, *Actionable Agile Metrics for Predictability: An Introduction*. Daniel S. Vacanti, Inc, 2015.

Zheglov2014a - Alexei Zheglov, 2014 'Inside a Lead Time Distribution' 07-09-2014. [Online] https://connected-knowledge.com/2014/09/07/inside-lead-time-distribution/ [Accessed 08-Feb-2018]

14.3 Chapter 2 - Lead Time

AndersonCarmichael2016 - David J Anderson, Andy Carmichael *Essential Kanban Condensed*. Blue Hole Press, 2016.

Berardinelli - Carl Berardinelli, 'A Guide to Control Charts'. [Online] https://www.isixsigma.com/tools-templates/control-charts/a-guide-to-control-charts/ [Accessed 12-May-2018]

Brodzinski2012a - Pawel Brodzinski, 2012 'Slack Time' 10-05-2012. [Online] http://brodzinski.com/2012/05/slack-time.html [Accessed 08-Feb-2018]

Carroll2016 - Ian Carroll. 'Why You Might Be Wasting Your Time

with Story Point Estimation'. Solutioneers (blog), 30 January 2020. https://www.solutioneers.co.uk/why-you-might-be-wasting-your-time-with-story-point-estimation/

Ma2011 - Dan Ma, 2011 'When Bill Gates Walks into a Bar' 04-09-2011. [Online] https://introductorystats.wordpress.com/2011/09/04/when-bill-gates-walks-into-a-bar/ [Accessed 12-May-2018]

Novkov2017 - Alex , 2017 'Track Your Aging Work In Progress' 23-02-2017. [Online] https://kanbanize.com/blog/aging-work-in-progress/ [Accessed 12-May-2018]

Savage2012 - Sam L. Savage, *The Flaw of Averages: Why We Underestimate Risk in the Face of Uncertainty*. John Wiley & Sons, 2012. For a shorter explanation by the same author see:
Sam L. Savage, 2002 'The Flaw of Averages' 11-2012. [Online] https://hbr.org/2002/11/the-flaw-of-averages [Accessed 18-May-2018]

Talai2014 - Nader Talai, 2014 - 'Does size matter?' https://cdn2.hubspot.net/hubfs/2495218/valueglide_dec2016_files/DOCs/AgileOrganisation_EstimationJourney.pdf

Vacanti2015 - Daniel S. Vacanti, *Actionable Agile Metrics for Predictability: An Introduction*. Daniel S. Vacanti, Inc, 2015.

Vacanti2018 - Daniel S. Vacanti, 2018 'Actionable Agile Metrics with Daniel Vacanti' 14-01-2018. [Online podcast] https://podcast.agileuprising.com/actionable-agile-metrics-with-daniel-vacanti/ [Accessed 12-May-2018]

14.4 Chapter 3 - Forecasting and planning

Anderson2011 - David Anderson, 2011 'Predictability and Measurement with Kanban by David Anderson' 17-10-2011. [Online] https://vimeo.com/32436546 [Accessed 18-May-2018]

Bakardzhiev2014 - Dimitar Bakardzhiev, 2014 '#NoEstimates Project

Planning Using Monte Carlo Simulation' 01-12-2014. [Online] https://www.infoq.com/articles/noestimates-monte-carlo [Accessed 21-Jun-2018]

Brodzinski2015b - Pawel Brodzinski, 2015 'Economic Value of Slack Time' 29-01-2015. [Online] http://brodzinski.com/2015/01/slack-time-value.html [Accessed 18-May-2018]

GermanTankProblem - 'German tank problem'. [Online] https://en.wikipedia.org/wiki/German_tank_problem [Accessed 08-Feb-2018]

Kingman Formula - Invistics, 2017, 'The King of Manufacturing Equations' 28-03-2017. [Online] http://www.invistics.com/the-king-of-manufacturing-equations/ [Accessed 18-May-2018]

Magennis2015 - Troy Magennis, 2015 'Agile 2015 - Risk - The Final Agile Enterprise Frontier' 06-08-2015. [Online] https://github.com/FocusedObjective/FocusedObjective.Resources/blob/master/Presentations/Agile%202015%20-%20Risk%20-%20The%20Final%20Agile%20Enterprise%20Frontier%20(Troy%20Magennis).pdf [Accessed 18-May-2018]

Sunk Cost Fallacy - David McRaney, 2011 'The Sunk Cost Fallacy' 25-03-2011. [Online] https://youarenotsosmart.com/2011/03/25/the-sunk-cost-fallacy/ [Accessed 18-May-2018]

Public GitHub repository for Business Metrics book - https://github.com/SkeltonThatcher/bizmetrics-book. This has many useful spreadsheets to help you get started.

Reinertsen2009 - Donald G. Reinertsen, *The Principles of Product Development Flow: Second Generation Lean Product Development*. Celeritas Pub, 2009. Chapter 5 in particular.

Savage2012 - Sam L. Savage, *The Flaw of Averages: Why We Underestimate Risk in the Face of Uncertainty*. John Wiley & Sons, 2012. For a shorter explanation by the same author see: Sam L. Savage, 2002 'The Flaw of Averages' 11-2012. [Online] https://hbr.org/2002/11/the-flaw-of-averages [Accessed 18-May-2018]

Vacanti2015 - Daniel S. Vacanti, *Actionable Agile Metrics for Pre-*

dictability: An Introduction. Daniel S. Vacanti, Inc, 2015.

14.5 Chapter 4 - Metrics for Flow

Anderson2013 - David Anderson, 2013 'Who is your Vice President of Delay?' 24-10-2013. http://djaa.com/who-your-vice-president-delay [Accessed 02-Dec-2018]

BlackSwanFarming - Black Swan Farming, 'Cost of Delay'. [Online] http://blackswanfarming.com/cost-of-delay/ [Accessed 02-Dec-2018]

Brodzinski2013 - Pawel Brodzinski, 2013 'Cumulative Flow Diagram' 15-06-2013. [Online] http://brodzinski.com/2013/07/cumulative-flow-diagram.html [Accessed 02-Dec-2018]

Burrows2014 - Mike Burrows, *Kanban from the Inside*. Blue Hole Press, 2014.

Deming1982 - W. Edwards Deming, *Out of the Crisis*. The MIT Press, 2000.

ForsgrenHumbleKim2018 - Nicole Forsgren PhD, Jez Humble and Gene Kim, *Accelerate: The Science of Lean Software and Devops: Building and Scaling High Performing Technology Organizations*. Trade Select, 2018.

Forss2013 - Hakan Forss, 2013 'LKUK13: The Red Brick Cancer' 03-12-2013. [Online] https://www.youtube.com/watch?v=sOb7Qqs2fe8 [Accessed 02-Dec-2018]

HumbleFarley2010 - Jez Humble, David Farley, *Continuous Delivery*. Addison Wesley, 2010.

Lowe2014a - David Lowe, 2014 'Little's Law' 28-10-2014. [Online] https://scrumandkanban.co.uk/littles-law/ [Accessed 02-Dec-2018]

Lowe2014b - David Lowe, 2014 'Little's Law and CFDs' 03-12-2014. [Online] https://scrumandkanban.co.uk/littles-law-and-cfds/

[Accessed 02-Dec-2018]

Lowe2015 - David Lowe, 2015 'Flow Efficiency' 01-06-2015. [Online] https://scrumandkanban.co.uk/flow-efficiency [Accessed 02-Dec-2018]

MagennisSpreadsheets - http://bit.ly/SimResources contains many spreadsheet templates related to forecasting & modelling software flow, curated by Troy Magennis.

Reinertsen2009 - Donald G. Reinertsen, *The Principles of Product Development Flow*. Celeritas Pub, 2009. See Chapter 5 in particular.

Rothman2015 - Johanna Rothman, 2015 'Resource Efficiency vs. Flow Efficiency, Part 1: Seeing Your System' 13-09-2015. [Online] https://www.jrothman.com/mpd/agile/2015/09/resource-efficiency-vs-flow-efficiency-part-1-seeing-your-system/ [Accessed 02-Dec-2018]

Skarin2008 - Mattias Skarin, 2018 'Agile game – Pass the pennies' 08-09-2008. https://blog.crisp.se/2008/09/08/mattiasskarin/1220882915232 [Accessed 02-Dec-2018]

Vacanti2015 - Daniel S. Vacanti, *Actionable Agile Metrics for Predictability: An Introduction*. Daniel S. Vacanti, Inc, 2015.

Wester2016 - Julia Wester, 2016 'Flow Efficiency: A great metric you probably aren't using' 25-09-2016. http://brodzinski.com/2012/05/slack-time.html [Accessed 08-Feb-2018]

Yip2018 - Jason Yip, 2018 'Why T-shaped people?' 24-03-2018. [Online] https://medium.com/@jchyip/why-t-shaped-people-e8706198e437 [Accessed 02-Dec-2018]

14.6 Chapter 5 - Metrics for Quality

Abela2006 - Andrew Abela. 'Choosing a Good Chart'. Extreme Presentation (blog), 6 September 2006. https://extremepresentation.

com/choosing_a_good-2/

Adzic2011 - Gojko Adzic, *Specification by Example: How Successful Teams Deliver the Right Software*. Manning Publications, 2011.

Big Buck Bunny - Wikipedia. 'Big Buck Bunny'. In Wikipedia, 27 August 2019. https://en.wikipedia.org/w/index.php?title=Big_Buck_Bunny&oldid=912659532

Cazaly2019 - Lynne Cazaly. Ish: The Problem with Our Pursuit for Perfection and the Life-Changing Practice of Good Enough. Lynne Cazaly, 2019.

CrispinGregory2009 - Lisa Crispin and Janet Gregory. *Agile Testing: A Practical Guide for Testers and Agile Teams*. Pearson Education, 2009. p.79

Crosby1987 - Philip B. Crosby. *Quality Is Free: The Art of Making Quality Certain*. New edition edition. New York u.a.: Penguin, 1987. p.15

Dettmer2007 - H. William Dettmer. *The Logical Thinking Process: A Systems Approach to Complex Problem Solving*. ASQ Quality Press, 2007.

Knaflic2015 - Cole Nussbaumer Knaflic. *Storytelling with Data: A Data Visualization Guide for Business Professionals*. John Wiley & Sons, 2015.

Magennis2017 - Troy Magennis. 'Data-Driven Coaching - Safely Turning Team Data into Coaching Insights'. 2017. https://www.infoq.com/presentations/data-coach/

Matts2013 - Chris Matts. 'Introducing Staff Liquidity (1 of n)'. The IT Risk Manager (blog), 24 November 2013. https://theitriskmanager.com/2013/11/24/introducing-staff-liquidity-1-of-n/

North2017 - Dan North. How to Fail with BDD | SkillsCast. Skillsmatter, 2017. https://skillsmatter.com/skillscasts/10695-how-to-fail-with-bdd

Reinertsen2009 - Donald G. Reinertsen, *The Principles of Product*

Development Flow: Second Generation Lean Product Development. Celeritas Pub, 2009. p.55

RAS - Wikipedia. 'Reliability, Availability and Serviceability'. In Wikipedia, 27 January 2019. https://en.wikipedia.org/w/index.php?title=Reliability,_availability_and_serviceability&oldid=880513040

Seddon2008 - John Seddon. Systems Thinking in the Public Sector. Triarchy Press, 2008.

Skelton2018 - Matthew Skelton. '5 Ways Site Reliability Engineering Transforms IT Ops'. TechBeacon, 9 October 2018. https://techbeacon.com/enterprise-it/5-ways-site-reliability-engineering-transforms-it-ops

14.7 Chapter 6 - Metrics for Value

Adzic2013 - Gojko Adzic. 'Scrum, Velocity, and Driving down the Motorway the Wrong Way'. Gojko's Blog (blog), 12 September 2013. https://gojko.net/2013/09/12/scrum-velocity-and-driving-down-the-motorway-the-wrong-way/

DeMarco1995 - Tom DeMarco. *Why Does Software Cost So Much?: And Other Puzzles of the Information Age.* New York, NY, USA: Dorset House Publishing Co., Inc., 1995. p43.

Drucker2012 - Peter Drucker. The Practice of Management. Routledge, 2012. p.29

Hubbard2014 - Douglas W. Hubbard. *How to Measure Anything Workbook.* John Wiley & Sons, 2014.

McClure2007 - Dave McClure. 'Startup Metrics for Pirates'. Business, 2007-08-08. https://www.slideshare.net/dmc500hats/startup-metrics-for-pirates-long-version

Perri2018 - Melissa Perri. *Escaping the Build Trap.* 1 edition. S.l.:

O'Reilly, 2018.

Richer2017 - Julian Richer. *The Richer Way*. Cornerstone, 2017.

Young2014 - Chris Young. 'Down To The Wire', 2014. http://worldofchris.github.io/down-to-the-wire/

15. About the authors

15.1 Mattia Battiston

Mattia Battiston

Mattia is originally from Verona, the city of love and home of Romeo of Juliet. He is a software developer and team leader with a great passion for learning and continuous improvement.

His focus is to help teams strive to get better, using Kanban, Lean, Agile, and of course a lot of data and metrics.

He's been interested in everything to do with Agile since the beginning of his career in 2008. He loves attending and speaking at conferences and meetups for sharing experiences and learning from each other.

15.2 Chris Young

Chris Young

Chris has been a computerist since age 12 when he was introduced to the magic of the Commodore PET. He works with engineering and management teams with the aim of getting the best results possible for all involved.

The role and value of metrics in this came from discovering the Lean/Kanban community around 2010 which got him measuring things in earnest.

He is an active member of the Lean/Agile/DevOps community speaking at Meet Ups and conferences across Europe including GOTO Berlin, Agile Cambridge, CukeUp and QCon.

15.3 Why we wrote this book

Mattia: having experimented with data and metrics for a long time, I wanted to write this book to share what worked for me and help people learn from my mistakes. Metrics have massively helped me and my teams make better decisions, have better discussions, and improve every day. I hope that reading this book will help others achieve the same, if not more.

Chris: I wanted to share my success and failure in using metrics to improve the work and outcomes of the various teams I've been

involved in. Having a foot in both the technology and business camps I have seen how shared metrics can provide a foundation for better communication and understanding and so lead to better outcomes. It's really important to me that this is a nurturing and helpful book and, like the work, it's very much a team effort.

16. Index