How to measure and improve team performance

Agile Metrics IN ACTION

Christopher W. H. Davis

FOREWORD BY Olivier Gaudin

SAMPLE CHAPTER

MANNING

Agile Metrics in Action

by Christopher W. H. Davis

Chapter 1

brief contents

Part 1

Measuring agile teams

Agile development has guidelines instead of hard-and-fast rules. Many teams that practice agile struggle with measuring their processes and their teams, despite having all the data they need to do the measurement.

Chapter 1 navigates through the challenges of agile measurement. You'll learn where you can get data to measure your team, how to break down problems into measureable units, and how to incorporate better agile measurement on your team.

Chapter 2 puts what you learned in chapter 1 into action through a case study where a team uses several open source technologies to incorporate better agile measurement. They identify key metrics, use the tools to collect and analyze data, and check and adjust based on what they find.

1

Measuring agile performance

This chapter covers

- Struggling with agile performance measurement
- Finding objective data for measuring agile performance
- Answering performance questions with data you're generating
- Adopting agile performance measurement

There isn't a silver-bullet metric that will tell you if your agile teams are performing as well as they can. Performance improvement is made possible by incorporating what you learn about your team's performance into how your team operates at regular intervals. Collecting and analyzing data in the form of metrics is an objective way to learn more about your team and a way to measure any adjustments you decide to make to your team's behavior.

1.1 Collect, measure, react, repeat—the feedback loop

Working with metrics in a feedback loop in parallel with your development cycle will allow you to make smarter adjustments to your team and help improve communication across your organization. Here are the steps in the feedback loop:

- *Collect*—Gather all the data you can about your team and performance. Understand where you are before you change anything.
- *Measure*—Analyze your data.
  - Look for trends and relationships between data points.
  - Formulate questions about your team, workflow, or process.
  - Determine how to adjust based on your analysis.
- *React*—Apply the adjustments based on your analysis.
- *Repeat*—Keep tabs on the data you've determined should be affected so you can continuously analyze and adjust your team.

The feedback loop depicted in figure 1.1 naturally fits into the operations of agile teams. As you're developing, you're generating and collecting data; when you pause to check and adjust, you're doing your analysis; and when you start again, you're applying lessons learned and generating more data.

Continuous delivery and continuous improvement

The word *continuous* is everywhere in agile terminology: continuous integration, continuous delivery, continuous improvement, continuous testing, continuous (choose your noun). No matter if you're doing Scrum, Kanban, extreme programming (XP), or some custom form of agile, keeping your stream of metrics as continuous as your check-and-adjust period is key.

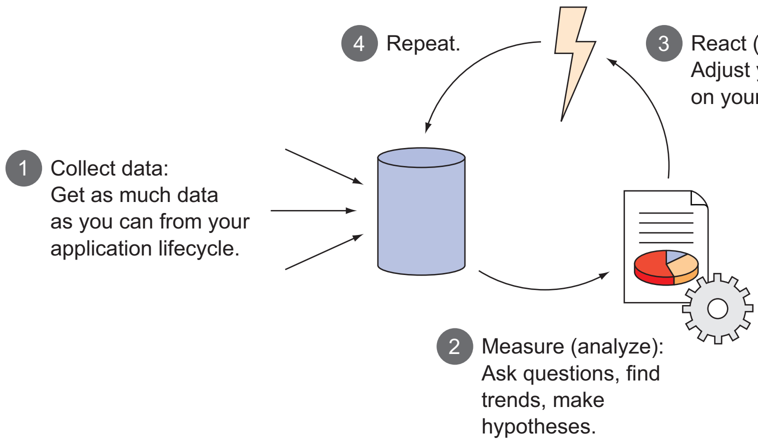

4 Repeat.
3 React (apply): Adjust your team based on your findings.
1 Collect data: Get as much data as you can from your application lifecycle.
2 Measure (analyze): Ask questions, find trends, make hypotheses.

Figure 1.1 The feedback loop: collecting data from your process, asking questions, and tweaking your process

To begin you need to know where you stand. You're probably already tracking something in your development process, like what was accomplished, how much code is changing, and how your software is performing.

Your questions will drive the analysis phase by providing a lens with which to view this data. Through this lens you can identify data points and metrics that help answer your questions. These data points then become the indicators of progress as you adjust your process to get to an ideal operating model for your team. Once you have questions you want to answer, then you can start identifying data points and metrics that represent them. At that point you can adjust how your team operates and track the metrics you've identified.

1.1.1 What are metrics?

"A method of measuring something, or the results obtained from this."

—*metrics* defined by Google

In the scope of this book metrics will represent the data you can get from your application lifecycle as it applies to the performance of software development teams. A metric can come from a single data source or it can be a combination of data from multiple data sources. Any data point that you track eventually becomes a metric that you can use to measure your team's performance. Examples of common metrics are:

- *Velocity*—The relative performance of your team over time
- *Changed lines of code (CLOC)*—The number of lines of code that were changed over time

Metrics can be used to measure anything you think is relevant, which can be a powerful tool when used to facilitate better communication and collaboration. These metrics in effect become key performance indicators (KPIs) that help measure what's important to your team and your business.

Using KPIs and data trends to show how certain data points affect behaviors and progress, you can tweak the behavior of your team and watch how the changes you make affect data that's important to it.

1.2 Why agile teams struggle with measurement

As you drive down the road, the gauges on your dashboard are the same as the gauges in the cars around you. There are highway signs that tell you how fast you should go and what you should do. Everyone on the road has gone through the same driving tests to get a driver's license and knows the same basic stuff about driving.

Agile development is nothing like this. The people involved in delivering a software product have different roles and different backgrounds. Their idea of what *good* means can vary substantially.

- A developer might think that *good* means a well-engineered piece of software.
- A product owner might define *good* as more features delivered.
- A project manager may think *good* means it was done on time and within budget.

Even though everyone is doing something different, they're all headed down the same road.

So now picture a bunch of people driving down the same road in different vehicles with totally different gauges. They all need to get to the same place, yet they're all using different information to get there. They can follow each other down the road, but when they pull over to figure out how the trip is going, each has different ideas of what the gauges in their vehicle are telling them.

Agile is a partnership between product owners and product creators. To make the most out of that partnership you need to smooth out the communication differences by turning the data you're already generating in your development process into agreed-upon metrics that tell you how your team is doing.

Let's look at some universal problems that end up getting in the way of a common understanding of agile metrics:

- Agile definitions of measurement are not straightforward.
- Agile deviates from textbook project management.
- Data is generated throughout the entire development process without a unified view.

All of these are common problems that deserve exploring.

1.2.1 Problem: agile definitions of measurement are not straightforward

There are a few commonly accepted tenets about measuring agile that tend to be rather confusing. Let's start with common agile principles:

- *Working software is the primary measure of progress.* That statement is so ambiguous and open to interpretation that it makes it very hard for teams to pinpoint exactly how to measure progress. Essentially the point is you are performing well if you're delivering products to your consumers. The problem is the subjective nature of the term *working software*. Are you delivering something that works per the original requirements but has massive security holes that put your consumer's data in jeopardy? Are you delivering something that is so non-performant that your consumers stop using it? If you answered yes to either question, then you're probably not progressing. There's a lot more to measuring progress than delivering working software.
- *Any measurement you're currently using has to be cheap.* So what's included in the cost associated with gathering metrics? Are licenses to software included? Are you looking at the hours spent by the people collecting measures? This statement belittles the value of measuring performance. When you start measuring something, the better thing to keep in mind is if the value you get from the improvement associated with the metric outweighs the cost of collecting it. This open statement is a good tenet, but like our first statement, it's pretty ambiguous.
- *Measure only what matters.* This is a bad tenet. How do you know what matters? When do you start tracking new things and when do you stop tracking others? Because these are hard questions, metrics end up getting thrown by the wayside

when they could be providing value. A better wording would be "measure everything and figure out why metrics change unexpectedly."

1.2.2 Problem: agile focuses on a product, not a project

One of the strengths of agile development methods is the idea that you are delivering a living *product*, not completing a *project*. A project is a defined set of time within which something is developed and tracked; a product is a living thing that continues to change to meet the needs of the consumer. This is shown in figure 1.2.

A good example of a project would be building a bridge. It gets designed, built to spec, and then it stands likely unchanged for a long time. Before you start the project you design exactly what you need so you can understand the total cost, and then you track the progress against the plan to make sure you stay on schedule and budget. This process is the art of project management.

A good example of a product is a mobile application geared toward runners that shows their paths on a map and aggregates their total mileage. It's a tool; you can run with it and get lots of data about your workouts. There are several competing apps that have the same functionality but different bells and whistles. To keep up with the competition, any app that competes in that space must constantly evolve to stay the most relevant product its consumers can use. This evolution tends to happen in small iterations that result in frequent feature releases which are immediately consumed and tracked for feedback that helps shape the direction of the next feature.

Frequently, software products are built through the lens of a project, which ends up mixing project management techniques typically used for large predictive projects

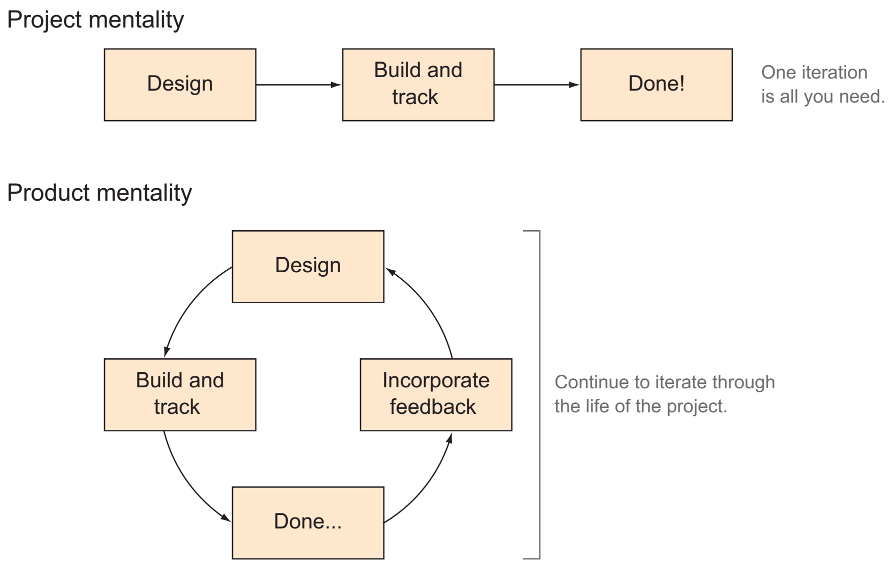
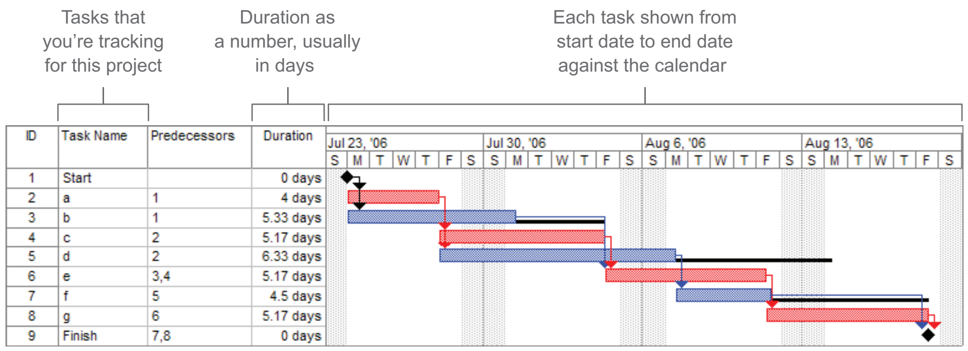


Figure 1.2 Project vs. product mentality



ID	Task Name	Predecessors	Duration
1	Start		0 days
2	a	1	4 days
3	b	1	5.33 days
4	c	2	5.17 days
5	d	2	6.33 days
6	e	3,4	5.17 days
7	f	5	4.5 days
8	g	6	5.17 days
9	Finish	7,8	0 days

Figure 1.3 An example Gantt chart

with agile product management used for consistent iterative delivery. This ends up putting project constraints on agile projects that don't fit. An example would be using a Gantt chart on an agile project. Gantt charts are great for tracking long-running projects but usually cause heartache when used to track something with a definition that is regularly in flux. An example Gantt chart is shown in figure 1.3.

From a project-management perspective it would be great to have a clear prediction of the cost of a complex feature that may take several sprints to complete, which is why these tools end up tracking agile teams.

1.2.3 Problem: data is all over the place without a unified view

Your team is generating a lot of data throughout your entire software development lifecycle (SDLC). Figure 1.4 shows the components that are typically used by an agile software delivery team.

The first problem is there is no standard for any of these boxes. There are several project-tracking tools, plus different source control and continuous integration (CI) systems, and your deployment and application-monitoring tools will vary depending on your technology stack and what you're delivering to your consumers. Different systems end up generating different reports that don't always come together. In addition, there's no product or group of products that encompasses all of the pieces of the software development lifecycle.

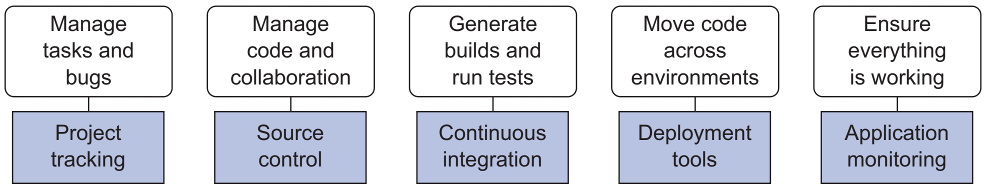

Manage tasks and bugs
Project tracking
Manage code and collaboration
Source control
Generate builds and run tests
Continuous integration
Move code across environments
Deployment tools
Ensure everything is working
Application monitoring

Figure 1.4 Groups of systems used to deliver software and what they do

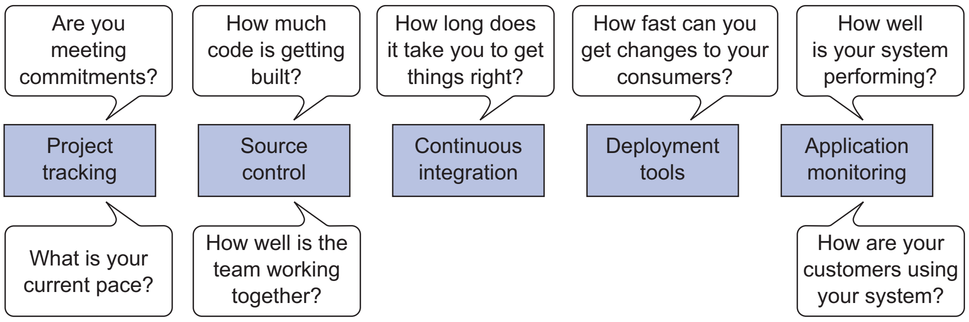

Are you meeting commitments?
How much code is getting built?
How long does it take you to get things right?
How fast can you get changes to your consumers?
How well is your system performing?
Project tracking
Source control
Continuous integration
Deployment tools
Application monitoring
What is your current pace?
How well is the team working together?
How are your customers using your system?

Figure 1.5 Questions you can answer with data from systems in your SDLC.

The second issue is that people in different roles on the team will focus on using different tools throughout the SDLC. Scrum masters are likely looking for data around your tasks in your project system, and developers pay the most attention to data in your source control and CI systems. Depending on the size and makeup of your team, you may have a completely different group looking after your application monitoring and even perhaps your deployment tools. Executives likely don't care about raw source control data and developers may not care about the general ledger, but when presented in the right way, having a unified view of all data is important in understanding how well teams are performing. No matter what angle you're looking from, having only a piece of the whole picture obviously limits your ability to react to trends in data in the best possible way.

1.3 What questions can metrics answer, and where do I get the data to answer them?

In the previous section we noted that all the data you're collecting in your SDLC is a barrier to understanding what's going on. On the flip side there's a huge opportunity to use all that data to see really interesting and insightful things about how your team works. Let's take the components of figure 1.4 and look at questions they answer in figure 1.5.

When you combine these different data points, you can start answering some even more interesting big-picture questions like the ones shown in figure 1.6.

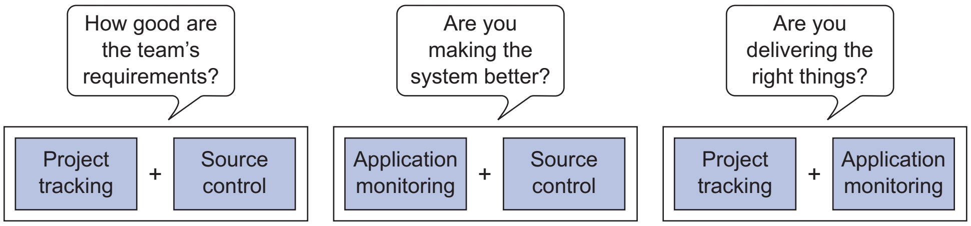


Figure 1.6 Adding data together to answer high-level questions

Your team is already generating a ton of data through CI, your task management system, and your production-monitoring tools. These systems are likely to have simple APIs that you can use to get real data that's better for communication and for determining KPIs and that you can incorporate into your process to help your team achieve better performance. As you know, the first step in the feedback loop is to start collecting data. By putting it into a central database, you're adding the systems from figure 1.4 to the feedback loop in figure 1.1 to get figure 1.7.

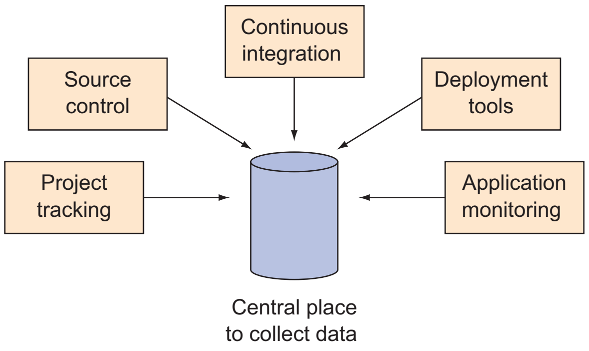


Figure 1.7 Collecting data from numerous places

There are several ways to do this:

- Semantic logging and log aggregators
- Products like Datadog (www.datadoghq.com) or New Relic Insights (newrelic .com/insights)
- An open source database like Graphite (graphite.wikidot.com/) to collect and display data
- A DIY system to collect and analyze metrics

If you're interested in building something yourself, check out appendix A, where we talk through that topic. If you have any of the other tools mentioned here, then by all means try to use them.

For now let's look at where you can get this data and what systems you should put in place if you don't already have them.

1.3.1 *Project tracking*

Tools like JIRA Agile, Axosoft OnTime, LeanKit, TFS, Telerik TeamPulse, Planbox, and FogBugz can all track your agile team, and all have APIs you can tap into to combine that data with other sources. All of these have the basics to help you track the common agile metrics, and although none goes much deeper, you can easily combine their data with other sources through the APIs. From these systems you can get data such as how many story points your team is completing, how many tasks you're accomplishing, and how many bugs are getting generated. A typical Scrum board is depicted in figure 1.8.

Here are questions you can answer with project-tracking data alone:

- How well does your team understand the project?
- How fast is the team moving?
- How consistent is the team in completing work?

Project-tracking data is explored in depth in chapter 3.

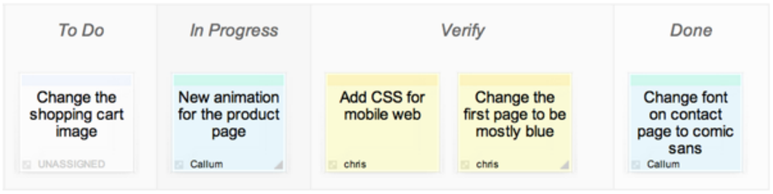

To Do
In Progress
Verify
Done
Change the shopping cart image
UNASSIGNED
New animation for the product page
Callum
Add CSS for mobile web
chris
Change the first page to be mostly blue
chris
Change font on contact page to comic sans
Callum

Figure 1.8 A typical Scrum board to track tasks in a sprint

1.3.2 Source control

Source control is where the actual work is done and collaboration across development teams happens. From here you can see which files are changing and by how much. Some source control systems allow you to get code reviews and comments, but in other cases you need additional systems to get that type of information. Tools like Stash, Bitbucket, and GitHub have rich REST-based APIs that can get you a wealth of information about your codebase. If you're still using SVN or something even older, then you can still get data, just not as conveniently as Git- or Mercurial-based systems. In that case you may need something like FishEye and Crucible to get more data around code reviews and comments about your code.

Here are two questions you can answer from source control alone:

- How much change is happening in your codebase?
- How well is/are your development team(s) working together?

We dive deep into source control in chapter 4.

1.3.3 The build system

After someone checks something into source control, it typically goes into your build system, which is where the code from multiple check-ins is integrated, unit tests are run, your code is packaged into something that can be deployed somewhere, and reports are generated. All of this is called continuous integration (CI). From here you can get lots of great information on your code: you can see how well your team's changes are coordinated, run your codebase against rule sets to ensure you're not making silly mistakes, check your test coverage, and see automated test results.

CI is an essential part of team software development, and there are several systems that help you get going quickly, including TeamCity, Jenkins, Hudson, and Bamboo. Some teams have taken CI past the integration phase and have their system deploy their code while they're at it. This is called continuous delivery (CD). Many of the same systems can be employed to do CD, but there are products that specialize in it, like ThoughtWorks, Go CD, Electric Cloud, and Nolio.

Whether you're doing CI or CD, the main thing you need to care about is the information that is generated when your code is getting built, inspected, and tested. The more mature a CI/CD process your team has, the more data you can get out of it.

Here are questions you can answer from CI alone:

- How fast are you delivering changes to your consumer?
- How fast can you deliver changes to your consumer?
- How consistently does your team do STET work?

More details on the data you can get from this step are found in chapter 5.

1.3.4 System monitoring

Once your code goes into production, you should have some type of system that looks at it to make sure it's working and that tells you if something goes wrong, such as whether a website becomes unresponsive or if your mobile app starts to crash every time a user opens it. If you're doing a really great job with your testing, you likely are paying attention to your system monitoring during your testing phase as well making sure you don't see any issues from a support perspective before your code goes into production.

The problem with system monitoring is that it's largely reactive, not proactive. Ideally all your data should be as close to the development cycle as possible, in which case you'll be able to react to it as a team quickly. By the time you're looking at system-monitoring data in a production environment your sprint is done, the code is out, and if there's a problem, then you're usually scrambling to fix it rather than learning and reacting to it with planned work.

Let's look at ways to mitigate this problem. One way is to use system monitoring as you test your code before it gets to production. There are a number of ways your code can make it to production. Typically you see something like the flow shown in figure 1.9, where a team works in STET local development environment and pushes changes to an integration environment, and where multiple change sets are tested together and a QA environment verifies what the end user or customer will get before you ship your code to production.

Because teams typically have multiple environments in the path to production, to make system monitoring data as proactive as possible you should be able to access it as

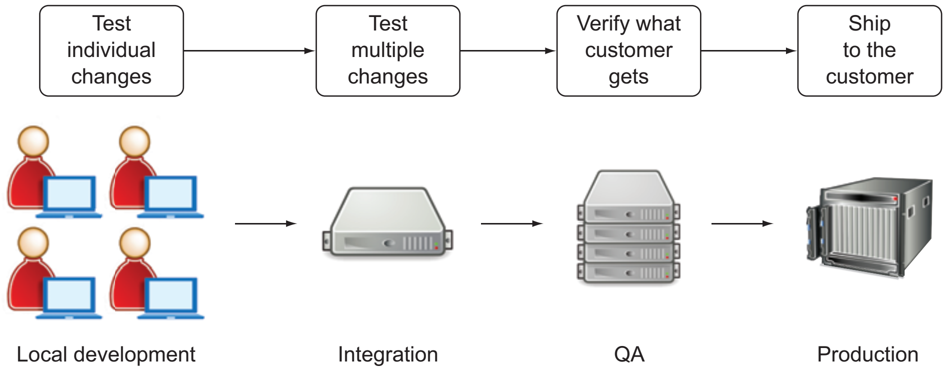


Figure 1.9 Typical environment flow in the path to production

close to the development environment as possible, ideally in the integration and/or QA stages.

A second way to mitigate this problem is to release new code to only a small number of your consumers and monitor how their activity affects the system. This is usually referred to as a *canary deploy* and is becoming more common in agile teams practicing CD.

Depending on your deployment platform, different tools are available for system monitoring. New Relic, AppDynamics, and Dynatrace are all popular tools in this space. We will go into much more detail about these in chapter 6.

All of the data we've looked at so far can tell you a lot about your team and how effectively you're working together. But before you can make this data useful, you need to figure out what is good and what is bad, especially in an agile world where your core metrics are relative.

Here are questions you can answer from your production-monitoring systems:

- How well is your code functioning?
- How well are you serving the consumer?

1.4 Analyzing what you have and what to do with the data

The second step in the feedback loop is analysis, or figuring out what to do with all the data you've collected. This is where you ask questions, look for trends, and associate data points with behaviors in order to understand what is behind your team's performance trends.

Essentially what you need to do is get all the data you've collected and run some kind of computation on it, as shown in figure 1.10, to determine what the combined data points can tell you. The best place to start is with the question you're trying to answer. When you're doing this, be careful and ask yourself *why* you want to know what you're asking. Does it really help you answer your question, solve a problem, or track something that you want to ensure you're improving? When you're trying to figure out what metrics to track, it's easy to fall into a rabbit hole of "it would be great to

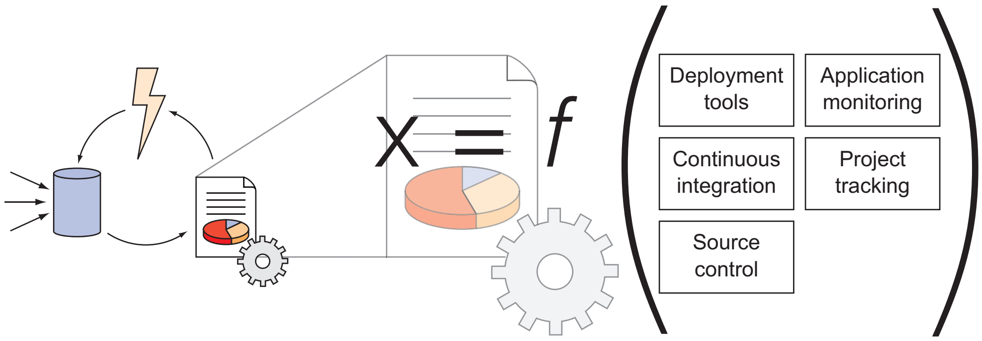


Figure 1.10 X is what you want to answer; some combination of your data can get you there.

have." Be wary of that. Stick with "just in time" instead of "just in case" when it comes to thinking about metrics. Then plug X into figure 1.10, where X is the question you want to answer.

1.4.1 Figuring out what matters

If you're swimming in a sea of data, it's hard to figure out what data answers what question and which metrics you should track. One strategy I find useful when I'm trying to figure out which metrics matter is mind mapping.

Mind mapping is a brainstorming technique where you start with an idea and then keep deconstructing it until it's broken down into small elements. If you're not familiar with mind mapping, a great tool to start with is XMind (www.xmind.net/), a powerful and free (for the basic version) tool that makes mind mapping pretty simple.

If you take a simple example, "What is our ideal pace?" you can break that down into smaller questions:

- What is our current velocity?
- Are we generating tech debt?
- What are other teams in the company doing?

From there you could break those questions down into smaller ones, or you can start identifying places where you can get that data. An example mind map is shown in figure 1.11.

Thinking a project through, mapping, and defining your questions give you a place to start collecting the type of data you need to define your metrics.

1.4.2 Visualizing your data

Having data from all over the place in a single database that you're applying formulas to is great. Asking everyone in your company to query that data from a command line probably isn't the best approach to communicate effectively. Data is such a part of the fabric of how business is done today that there is a plethora of frameworks for

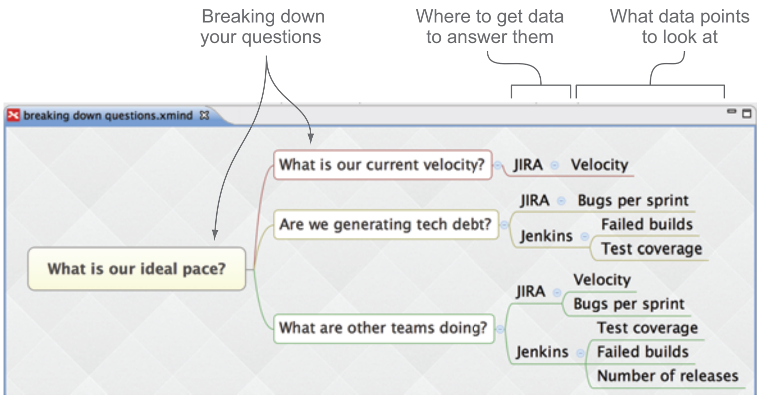

Breaking down your questions
Where to get data to answer them
What data points to look at
breaking down questions.xmind
What is our ideal pace?
What is our current velocity?
JIRA
Velocity
Are we generating tech debt?
JIRA
Bugs per sprint
Jenkins
Failed builds
Test coverage
What are other teams doing?
JIRA
Velocity
Bugs per sprint
Jenkins
Test coverage
Failed builds
Number of releases

Figure 1.11 Breaking down questions with XMind

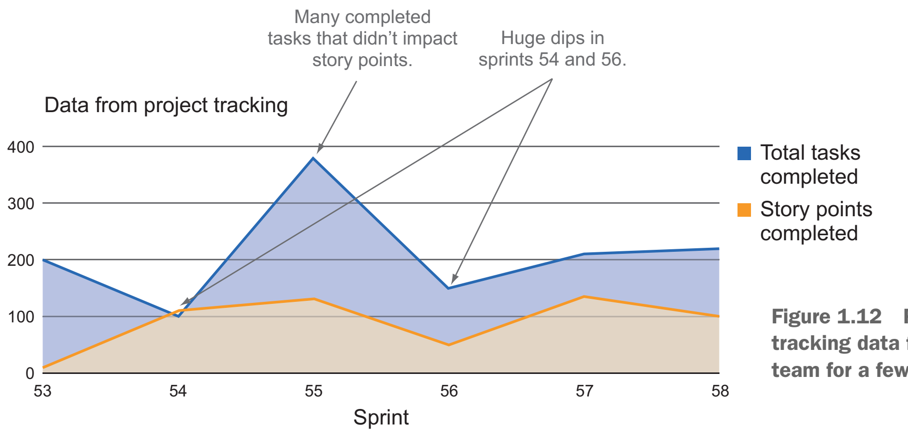


Figure 1.12 Project tracking data for a team for a few sprints

visualizing it. Ideally you should use a distributed system that allows everyone access to the charts, graphs, radiators, dashboards, and whatever else you want to show.

Keep in mind that data can be interpreted numerous ways, and statistics can be construed to prove different points of view. It's important to display the data you want to show in such a way that it, as clearly as possible, answers the questions you're asking. Let's look at one example.

If you want to see how productive your team is, you can start by finding out how many tasks are being completed. The problem with measuring the number of tasks is that a handful of very difficult tasks could be much more work than many menial ones. To balance that possibility you can add the amount of effort that went into completing those tasks, measured in figure 1.12 with story points.

Whoa! Look at those dips around sprints 54 and 56! This obviously points to a problem, but what is the problem? You know that the development team was working really hard, but it didn't seem like they got as much done in those sprints. Now let's take a look at what was going on in source control over the same time period, as shown in figure 1.13.

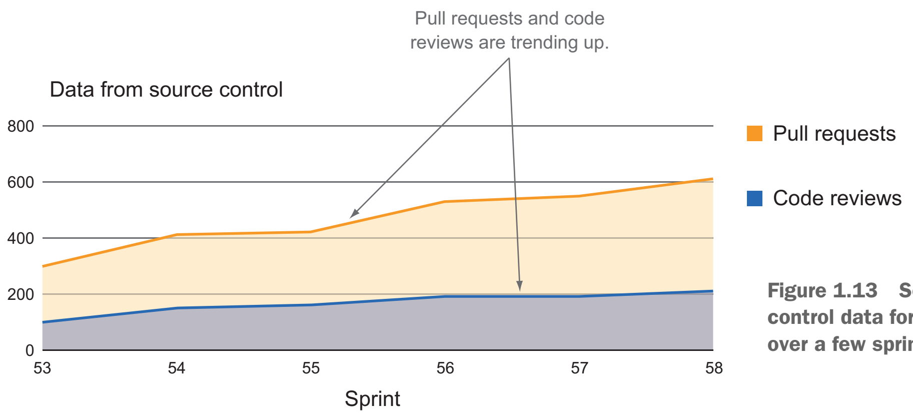


Figure 1.13 Source control data for a team over a few sprints

If anything, it looks like the development teams were doing more work over time—a lot more work! So if you're writing more code, and the number of bugs seems to be pretty constant relative to the tasks you're getting done, but you're not consistent in the amount of stuff you're delivering, what's the real issue? You may not have the answer yet, but you do have enough data to start asking more questions.

1.5 Applying lessons learned

Applying lessons learned can be the hardest part of the feedback loop because it implies behavioral changes for your team. In other words, collecting and analyzing data are technical problems; the final part is a human problem. When you're developing a software product, you're constantly tweaking and evolving code, but it's not always easy to tweak your team's behavior.

Whenever you're trying to change something because it's not good, you can easily perceive that someone is doing something bad or wrong, and who likes to be told that? The first key is always keeping an open mind. Remember that we're all human, no one is perfect, and we always have the capacity to improve. When things aren't perfect (and are they ever?), then you have an opportunity to make them better.

When you're presenting trends that you want to improve on, ensure that you're keeping a positive spin on things. Focus on the positive that will come from the change, not the negative that you're trying to avoid. Always strive to be better instead of running away from the bad.

Always steer away from blame or finger pointing. Data is a great tool when used for good, but many people fear measurement because it can also be misunderstood. An easy example of this would be measuring lines of code (LOC) written by your development team. LOC doesn't always point to how much work a developer is getting done. Something that has a lot of boilerplate code that gets autogenerated by a tool may have a very high LOC; an extremely complex algorithm that takes a while to tune may have a very low LOC. Most developers will tell you the lower LOC is much better in this case (unless you're talking to the developer who wrote the code generator for the former). It's always important to look at trends and understand that all the data you collect is part of a larger picture that most likely the team in the trenches will understand the best. Providing context and staying flexible when interpreting results is an important part of ensuring successful adoption of change. Don't freak out if your productivity drops on every Wednesday; maybe there's some good reason for that. The important thing is that everyone is aware of what's happening, what you're focused on improving, and how you're measuring the change you want to see.

1.6 Taking ownership and measuring your team

Development teams should be responsible for tracking themselves through metrics that are easy to obtain and communicate. Agile frameworks have natural pauses to allow your team to check and adjust. At this point you should have an idea of the good and bad of how you approach and do your work. Now, take the bull by the horns and start measuring!

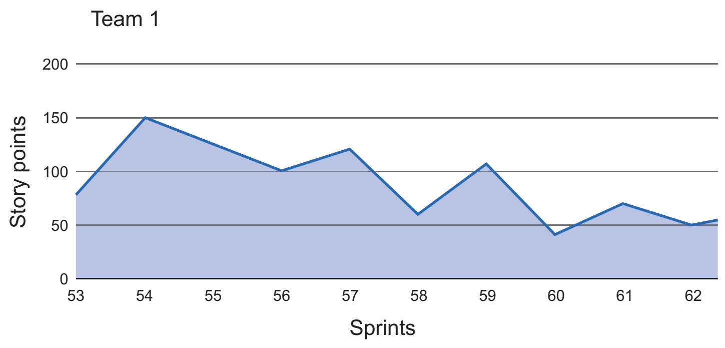


Figure 1.14 Team 1's complete story points over time

1.6.1 Getting buy-in

You can start collecting data and figuring out metrics on your own, but ideally you want to work with your team to make sure everyone is on the same page. These metrics are being used to measure the team as a whole so it's important that everyone understands what you're doing and why and also knows how to contribute to the process.

You can introduce the concepts we've talked about so far to get your team ready to do better performance tracking at any time, but perhaps the most natural time would be when your team breaks to see how STET doing. This could be at the end of a sprint, after a production release, at the end of the week, or even at the end of the day. Wherever that break is on your team is where you can introduce these concepts.

Let's look at a team which is using Scrum and can't seem to maintain a consistent velocity. Their Scrum master is keeping track of their accomplishments sprint after sprint and graphing it; as shown in figure 1.14, it's not looking good.

After showing that to leadership, everyone agrees that the team needs to be more consistent. But what's causing the problem? The development team decides to pull data from all the developers across all projects to see if there's someone in particular who needs help with their work. Lo and behold, they find that Joe Developer, a member of Team 1, is off the charts during some of the dips, as shown in figure 1.15. After digging a bit deeper, it is determined that Joe Developer has specific knowledge of the product a different team was working on and was actually contributing to multiple

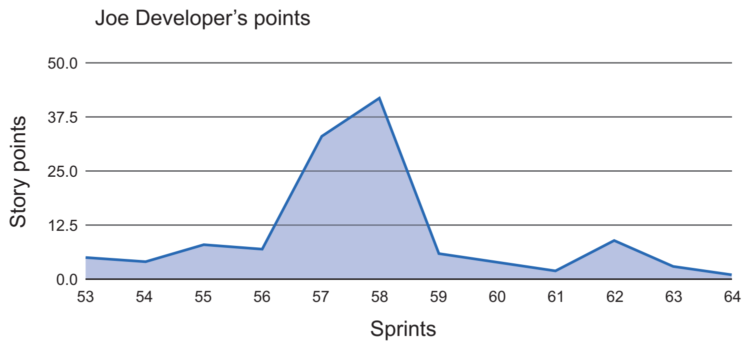


Figure 1.15 Joe Developer's contribution is off the charts, but his team is not performing well.

teams during that time. He was overworked, yet it looked as if the team were underperforming.

In this case it's pretty easy to show that the commitment to multiple projects hurts the individual teams, and apparently Joe Developer is way more productive when working on the other team's product. People are usually eager to show trends like this, especially when they indicate that individual performance is great but for some reason forces outside the team are hurting overall productivity. If you're being measured, then you might as well ensure that the entire story is being told. In this case the data ended up pointing to a problem that the team could take action on: fully dedicate Joe Developer to one team to help avoid missing goals on another team and to not burn your engineers as you're delivering product.

Collecting data is so easy that you shouldn't need buy-in from product sponsors to get it started. Remember that collecting metrics needs to be cheap, so cheap that you shouldn't have to ask for permission or resources to do it. If you can show the value of the data you're collecting by using it to point to problems and corresponding solutions and ultimately improve your code-delivery process, then you will likely make your product sponsor extremely happy. In this case it's usually better to start collecting data that you can share with sponsors to show them you know your stuff rather than trying to explain what you're planning to do before you go about doing it.

1.6.2 Metric naysayers

There will likely be people in your group who want nothing to do with measuring their work. Usually this stems from the fear of the unknown, fear of Big Brother, or a lack of control. The whole point here is that teams should measure themselves, not have some external person or system tell them what's good and bad. And who doesn't want to get better? No one is perfect—we all have a lot to learn and we can always improve. Nevertheless here are some arguments I've heard as I've implemented these techniques on various teams:

- *People don't like to be measured.* When we are children, our parents/guardians tell us we are good or bad at things. In school, we are graded on everything we do. When we go out to get a job, we're measured against the competition, and when we get a job, we're constantly evaluated. We're being measured all the time. The question is, do you want to provide the yardstick or will someone else provide it?
- *Metrics are an invasion of my privacy.* Even independent developers use source control. The smallest teams use project tracking of some kind, and ideally you're using some kind of CI/CD to manage your code pipeline. All of the data to measure you is already out there; you're adding to it every day and it's a byproduct of good software practice. Adding it all together for feedback on improvement potential isn't an invasion of privacy as much as it's a way to make sure you're managing and smoothing out the bumps in the agile development road.

- *Metrics make our process too heavy.* If anything, metrics help you identify how to improve your process. If you choose the right data to look at, then you can find parts of your process that are too heavy, figure out how to streamline them, and use metrics to track your progress. If it feels like metrics are making your process too heavy because someone is collecting data and creating the metrics manually, then you have a golden opportunity to improve your process by automating metrics collection and reporting.
- *Metrics are hard; it will take too much time.* Read on, my friend! There are easy ways to use out-of-the-box technology to obtain metrics very quickly. In appendices A and B we outline open source tools you can use to get metrics from your existing systems with little effort. The key is using the data you have and simply making metrics the byproduct of your work.

1.7 Summary

This chapter showed you where to find data to measure your team and your process and an overview of what to do with it. At this point you've learned:

- Measuring agile development is not straightforward.
- By collecting data from the several systems used in your SDLC, you can answer simple questions.
- By combining data from multiple systems in your SDLC, you can answer big-picture questions.
- By using mind mapping you can break questions down into small enough chunks to collect data.
- By using simple techniques, measuring agile performance isn't so hard.
- Showing metrics to your teammates easily demonstrates value and can easily facilitate buy-in.

Chapter 2 puts this in action through an extended case study where you can see a team applying these lessons firsthand.

AGILE DEVELOPMENT

Agile Metrics IN ACTION

Christopher W. H. Davis

The iterative nature of agile development is perfect for experience-based, continuous improvement. Tracking systems, test and build tools, source control, continuous integration, and other built-in parts of a project lifecycle throw off a wealth of data you can use to improve your products, processes, and teams. The question is, how to do it?

Agile Metrics in Action teaches you how. This practical book is a rich resource for an agile team that aims to use metrics to objectively measure performance. You'll learn how to gather the data that really count, along with how to effectively analyze and act upon the results. Along the way, you'll discover techniques all team members can use for better individual accountability and team performance.

What's Inside

- Use the data you generate every day from CI and Scrum
- Improve communication, productivity, transparency, and morale
- Objectively measure performance
- Make metrics a natural byproduct of your development process

Practices in this book will work with any development process or tool stack. For code-based examples, this book uses Groovy, Grails, and MongoDB.

Christopher Davis has been a software engineer and team leader for over 15 years. He has led numerous teams to successful delivery using agile methodologies.

To download their free eBook in PDF, ePub, and Kindle formats, owners of this book should visit manning.com/AgileMetricsinAction

Free eBook
SEE INSERT

"A steadfast companion on your expedition into measurement."
—From the Foreword by Olivier Gaudin, SonarSource

"The perfect starting point for your agile journey."
—Sune Lomholt, Nordea Bank

"You'll be coming up with metrics tailored to your own needs in no time."
—Chris Heneghan, SunGard

"Comprehensive and easy to follow."
—Noreen Dertinger
Dertinger Informatics, Inc.

ISBN 13: 978-1-617292-48-4
ISBN 10: 1-617292-48-6
54499
9 781617 292484

MANNING

$44.99 / Can $51.99 [INCLUDING eBOOK]