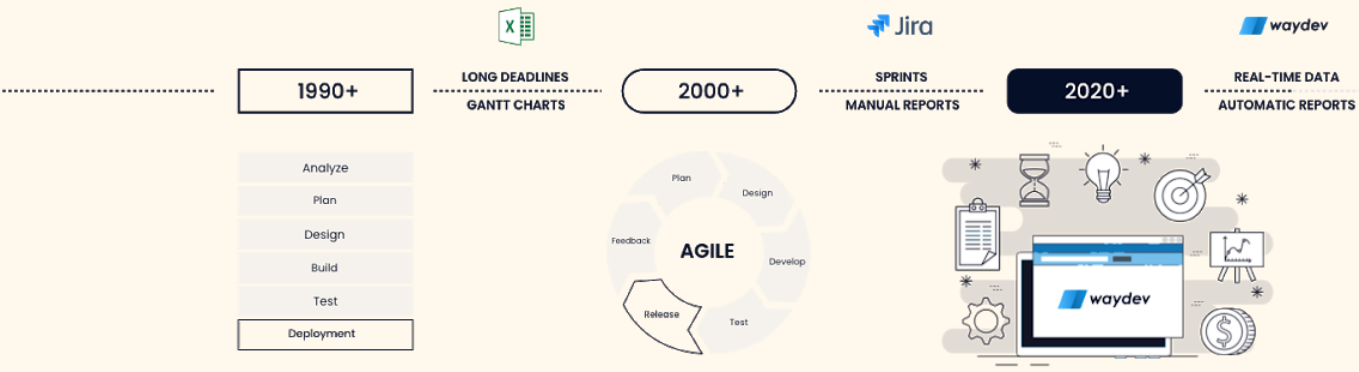
waydev

PLAYBOOK FOR DATA-DRIVEN ENGINEERING LEADERS

WAYDEV's PLAYBOOK FOR DATA-DRIVEN ENGINEERING LEADERS



INTRODUCTION

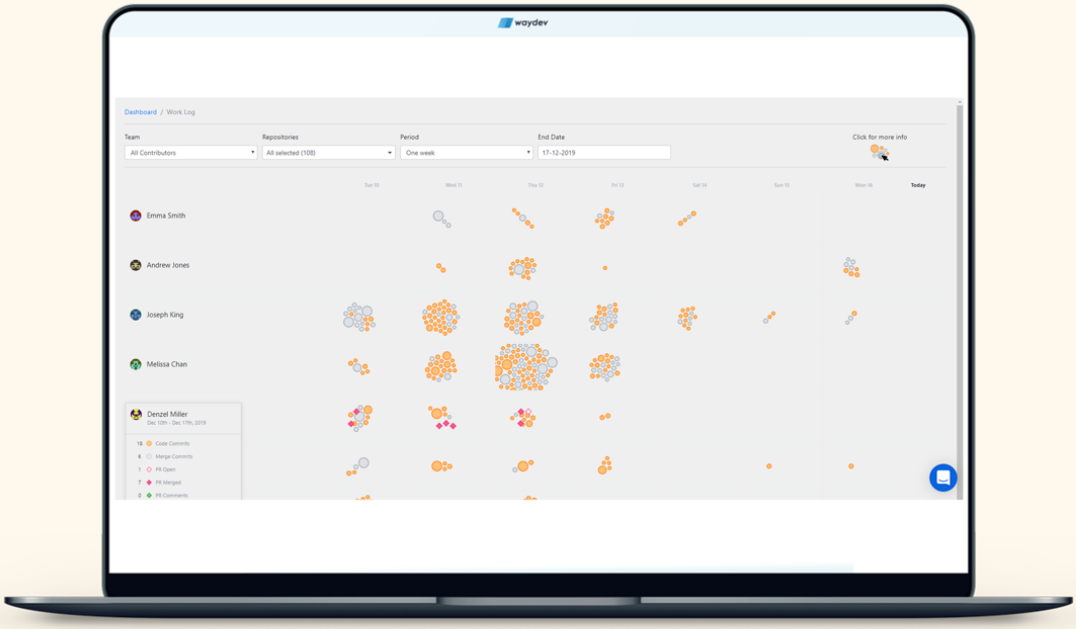
An engineering leader who works on the business understands how their team works. They understand their team collaboration dynamics and code reviews. They visualize trends in work patterns over time and recognize achievements that otherwise would be invisible. They identify coaching opportunities that elevate their team as a whole.

They use this data to push information up and raise visibility across the organization around how hard engineering is working, in a way that makes sense to the leadership that does not see the ground truth. By visualizing the full delivery process from idea to production, looking not only at what was built but also how it was built, they can spot bottlenecks or process issues that, when cleared, increase the overall health and capacity of the team.

Well developed qualitative analysis, senses, and gut feelings of the health of your team's development have brought software engineering a long way, and with the advent of a new age of robust qualitative analysis at the fingertips, it is our responsibility as engineering leaders to turn those gut feelings, and what healthy development looks like, into repeatable, measurable, quantitative insights that we can leverage to help our team deliver the best work and thrive.

Daily standups

Chances are you already run daily standups, and your engineering team is used to them. Your standups might involve product owners or a scrum master in addition to the team. Perhaps everyone's physically standing up in the corner of the office or calling in remotely. The logistics of your standups are probably under control. You've been using standups for their primary purpose according to their scrum methodology, which is to keep your team informed, connected, and calibrated throughout a sprint.



Standups are great check-ins to identify the status of a sprint, see what's in progress and see what work is in review. They're also helpful in identifying potential risks and blockers so you can remove roadblocks and keep your team on track.

In a standup, you might ask questions like:

- What did you work on yesterday?
- What are you working on today?
- What issues are blocking you?'

Those questions are meant to help flag blockers and create a sense of collaboration and shared contributions, but there may be times when you're not convinced that standups are that useful or necessary for the team. Maybe work is progressing as usual, and no one seems to be running into any blockers. There aren't any processes or systems that the team thinks should be changed, and everything that's brought up in meetings seems to be really positive.

Standups can be a compelling space for connecting the team and creating a rhythm, but it's not uncommon for these meetings to experience some attrition at times. There are a few overarching themes teams may experience. One of those common themes includes issues with **self-reporting**. When you're in the weeds with work, it can be challenging to notice the broader patterns or recurring issues that a manager or team lead might be able to help you with. Sometimes an engineer might be knee-deep working on a particularly challenging problem and not realize that only when they find themselves churning on this problem for days that they're stuck on. Maybe they just need someone to talk through the problem with, or perhaps the problem is larger than that.

The issue with self-reporting is often that team members face friction in moving their work forward all the time and might not realize when it's worth bringing up in a team meeting. So team leads and managers might think everything is progressing as usual without realizing that there's something they could help with.

Similarly, another limitation with standups is knowing when to offer assistance. It can be tempting to walk around and ask people how things are going and when they say 'it's fine,' you move along to the next person. Additionally, some managers might over-correct and check-in far too often on a team member's progress. **There can be a fine line between absentee management and micromanagement.** You don't want to assume things are going fine, but you also don't want to interrupt when you think someone's stuck or taking a project beyond what the business was asking for.

With data, you can **see work patterns and know when something seems off or when work is progressing**. As usual, you can be confident when offering assistance or when bringing up a discussion in a standup about something you see in the team's data. Data drives healthier, more productive conversations. Without being connected to what's going on, it can be more challenging to have timely and meaningful conversations.

With this said, what standups do really well is this: **they help drive momentum and create alignment within the team, and they provide a space for team members to connect with others who are working on similar projects or have faced similar challenges before.** The things that are harder to surface in a stand up are bottlenecks to work. Even more challenging to identify, are the underlying systematic issues that may be at hand. Wouldn't your standups be more productive if you could dig into the actual roadblocks and bottlenecks in a sprint or that the team has frequently been experiencing scope creep late in their sprints? Are there opportunities to cross-train team members on new areas of the code?

This chapter will show you how to use analytics about the development process so you can run more productive standups that lead an overall healthier way of working to help the team execute business outcomes more predictable. You'll see how you can have more targeted conversations about how things are going. You'll see how you can experiment with the timing or flow of your standups. You could begin with data instead of starting every standup by saying, 'so who wants to go first?'

You can start standups by going over what you see in the reports and then encourage more in-depth conversations about why different dynamics might be showing up in the data, what was challenging, what went smoothly, and what everyone would like to see moving forward. You'll see how data regarding the development process can help you spot roadblocks and surface discussions about trends and patterns like process issues and collaborative solutions to

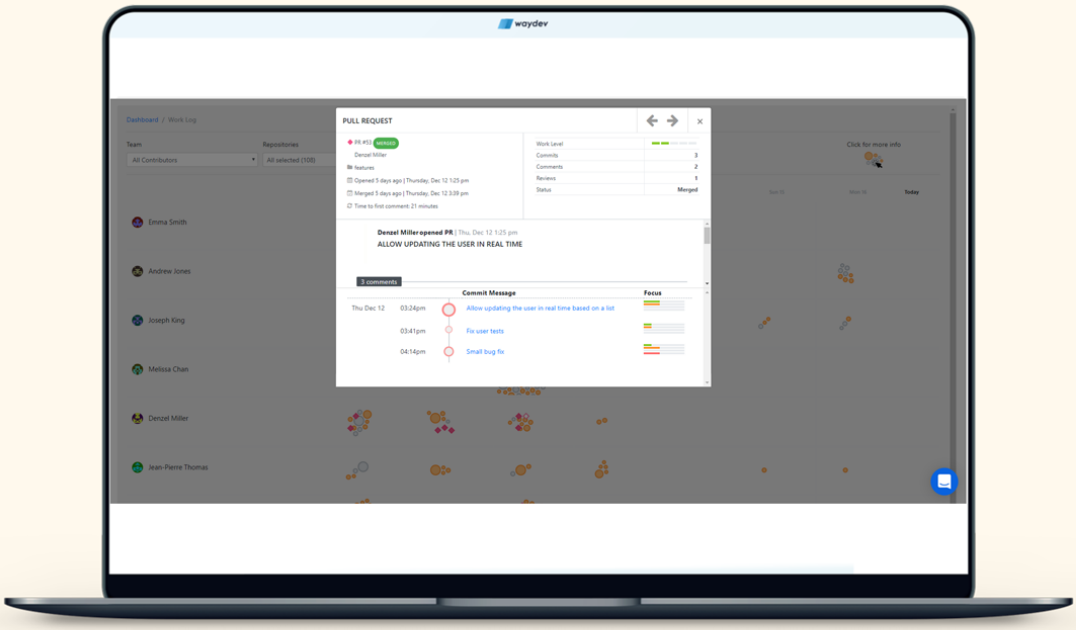
those issues. You'll see specific reports and use cases for each, so you can eliminate guesswork, not only about running productive standups, but also about how to actually use analytics about the development process to benefit the team as a whole. Data-driven standups can help your team feel more informed, connected, and calibrated, and can ultimately contribute to improved motivation and retention within the team.

The **Work Log** is a great way to find your bearings. The **Work Log** report helps you visualize work patterns and get a better understanding of how the team works. This report helps eliminate guesswork and knowing how things are going. It gives a voice to engineers who are quieter or working on less flashy work. It helps managers get visibility into their onboarding process, understand where their team is focusing their attention or notice when there's an unexpected spike in activity.

The **Work Log** also helps managers get a better understanding of their team's work patterns, which will differ from one individual to the next. It'll become easier to notice coaching opportunities to help team members practice healthier and more sustainable work behaviors, which in turn will benefit the team as a whole.

For example, when a team member has the habit of saving up their work for over a week or more and then submitting it for review all at once right before the deadline - that can create a lot of stress within the team as they will need to review a large amount of work with little time left in a sprint. That also limits the feedback the team can provide and increases the probability of introducing bugs to the codebase. If a manager sees this pattern, they can coach the team member to break up their work and make small, frequent commits, which will help their team review it earlier and provide for a more sustainable workflow for the team as a whole.

To understand the team's work patterns, identify bottlenecks, and spot timely and meaningful areas for coaching, you first need to have visibility into the team's progress. With the **Work Log**, you'll see a visualization of work patterns in detail. The **Work Log** report brings in data from code commits where you'll see a broader view of commit output. You can also see merge commits, PR open, PR merged, PR close and PR comments.



Depending on your use case, you can also view the work activity by team or by repo. With this report, we can see the team's work patterns throughout a week or sprint. When we start getting comfortable with visualizing how various team members work, we can then begin to notice when something might be off or experimenting with processes or meeting times and having a data-driven discussion with the team around how that affects their workflow.

So if for example, you notice that there's a drop in activity on Thursdays, the same day you have a specific weekly meeting, it might be worth talking to the team about whether it would be helpful to move that meeting to a different time or day, or even change how it's structured. All we need to do in that situation is to bring this report to the standup and use it as a starting point for discussion. Then if the team thinks it would be beneficial to change the date or structure of that meeting, you can create a hypothesis together around whether those changes will help the team and not slow them down. Then bring that report back into your standup in a week or two weeks and assess how that change has affected the team. Managers can also use this report to see how work is being carried throughout the team and whether there's anyone that's maybe taking on too much and could delegate some of that work.

For example, if you see that one of your engineers (let’s call her **Tracy**) has been working on some weekends it might be a one-off situation. But if we look at previous weeks, you’ll see that Tracy continues to work on Saturdays and Sundays, and we should take that as a signal that she might be stretched too thin. You can look at what she's working on in your project tracking system or by clicking on these specific commits and seeing the actual code. From there, you can start to look for opportunities to cross-train other team members on those work areas or otherwise, coach Tracy to delegate some of that work. This may also be a signal that you could set more realistic expectations with other teams or departments on what an individual contributor can and should produce. You can bring this report into a one-on-one to check in with Tracy to see if you can help with distributing her work.

You can also bring it into the standup to spark a discussion around who might be interested in the projects that Tracy is working on. By finding a healthier distribution of work across the team, you're helping free up Tracy to take on more exciting projects, and you're helping other team members grow their skills and experience in a specific area in the long term. It's a more sustainable way for a team to work.

The **Work Log** isn't just designed to help you understand the team's work patterns, how they're allocating their energy, and whether they've been pushed to over-investing in certain areas. The **Work Log** is equally suited to help you with one of the hardest jobs you have as a manager: **Advocacy** - your team needs you to be their advocate.

For instance, if we see that an engineer (let's call him **Chris**) has an immense amount of review activity because we all know that **Chris** is one of the most senior and key members of the team. But with the historically used ticket throughput or story point based reporting mechanisms that some organizations still rely on, this engineer probably wouldn't even show up on the list since all of their work this week has been happening in pull requests. Of course, that does not mean that this person is not contributing to the organization. Quite the opposite, this teammate is enabling their peers to move all of their work forward, so these contributions which don't show up are some of the most important work being done on the team.

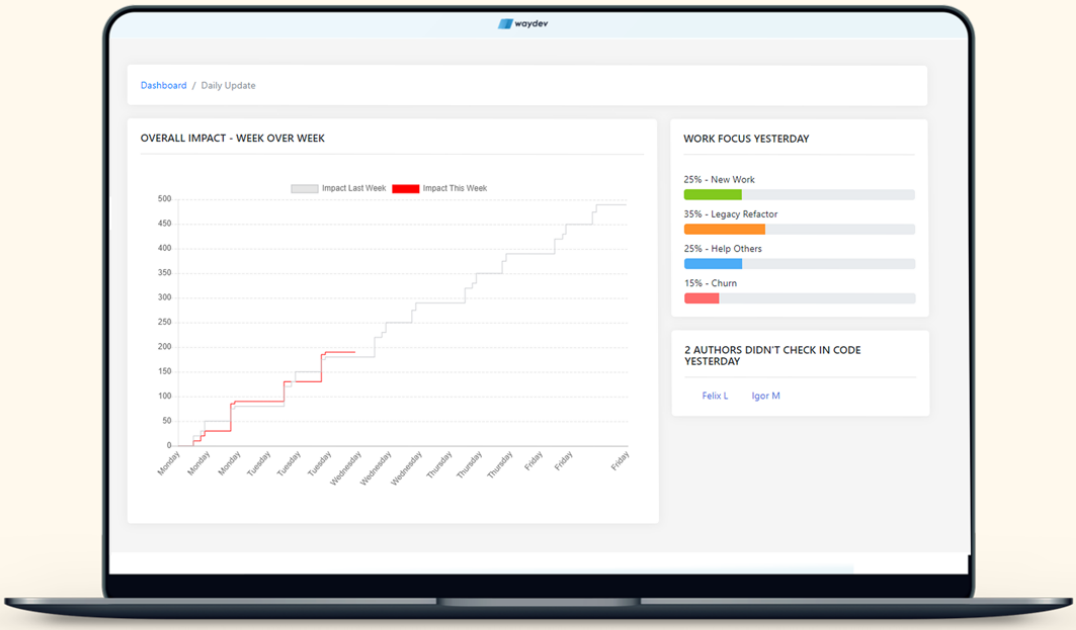
What an excellent opportunity to be your developer's voice within the organization to **advocate** for their fantastic work. Use this in standups to highlight all the types of good work and work patterns to the team. Use this report to also frequently show your manager how the team is doing and what different team members are working on. It will build trust over time and make it easier for you to make the case to promote specific team members.

Finally, another way this report is helpful is noticing when teams are backloading release or noticing what release might be at risk (this is easier to identify if you switched to a two-week view.) You'll see when team members are checking in work early and getting feedback from their peers throughout a sprint. You'll also be able to see if work is coming in late in a sprint and how that can lead to higher levels of stress, as a team scrambles to complete reviews and can lead to delayed releases.

For example, if the 12th April is the end of a sprint and you see a spike activity on the 9th, 10th and 11th April, that could be a sign that there was an unexpected increase in scope or there was a miscommunication around the original specs. Bring the support into your standups regularly, so when something seems off, the team will become comfortable with using the data as a starting point for discussions that end with a positive result and, worst case, you can now know earlier when a release needs to be pushed back so you can communicate that to other teams or departments that may be planning around that release date. Hence, it's not as much of a surprise and people can adjust accordingly. The **Work Log** will help you better understand the work patterns across your teams so you can use data to improve the quality and effectiveness of your standups.

The **Work Log** provides a unified view of the commit and pull request activity across the team. You can refer to it before your daily standup to check the progress on specific projects or to visualize work patterns in the team. Get a feel for how the team works so you can identify opportunities for coaching and know when something outside the normal development process might be causing a problem.

At a high level, the **Daily Update** provides a view across what happened in the code base the previous day. It is most commonly used in daily standups and can quickly help your team surface discussion around blockers in the code they are working in. The daily update report includes a breakdown of commit activity across our four work types: New Work, Legacy Refactoring, Help Others, and Churn. This breakdown helps engineer's surface more specific discussion around the work they achieved yesterday. If an individual realized that they had to do more replatforming than expected to deliver a new feature, this is a great natural way to communicate this to the team as a source of transparency and also to prompt the manager or scrum master to realign expectations with key stakeholders if a timeline feels like it needs to be pushed out an iteration or two.



Another dynamic that you can visualize in the **Daily Update** are potential blockers in the code. This is an incredibly fast way to showcase the changes themselves so they can quickly pull the team and identify the individual that has the bandwidth and domain knowledge to get them unblocked at 9:30 AM rather than spending all day trying to figure it out and burning almost a whole day of time that could have been allocated elsewhere.

This report allows you to streamline that process and provide your team with an effective medium to speak towards more granular blockers or curiosities in the code base that they would like to get their team's input on. The actual format of this discussion can vary, but one of the most common formats has the **Daily Update** open throughout your standup and to follow the standard ‘Go around the circle’ structure, where each person in the room voices any blockers. Now you can communicate to your team that they have another source of information at their fingertips to describe blockers in the code base solutions they are creating. Ideally, this report is used proactively by developers to bring up discussions and work with their peers to discuss ideas or collaborate on solutions to pressing problems with easy access to all the codebase work that happened yesterday in an easily digestible format and they should feel welcomed past more granular feedback from their peers.

For example, let's say that an engineer (let's call him **Donald**) was running into a blocker in the code. He might say to the group: 'Here's a solution. This might get us across the line, but we might have to revisit this just like a month later. I don't feel super comfortable with it.' **Donald** can actually click onto that commit that will redirect him over to the git host. **Donald** can point at the changes themselves and say: 'Hey team, here's the work. Here are the areas that I'm unsure of the best path forward. Has anyone worked in this area of code before? Does anyone have any domain knowledge here? I'd love to take a brief moment with you after this daily standup to get unblocked, rather than me chugging away at this for the next six hours until I figure it out.'

This allows engineers to unblock themselves and feel comfortable in their solution in the morning right after the standup rather than trying to power through to finally find a solution by the end of the workday or later. For that developer, this can be a small commanding win when it occurs multiple times throughout a year. It's a meaningful bonus to a team's collaboration ability to ship even better solutions.

For example, if your team shows an uptick in **Help Others** that coincides with some new members joining the team, it can be helpful to know that they are spending more time on helping the new hires ramp up, and you'd likely see that taper off over time. If you see an uptick in **Churn** in the back half of a sprint, you might open that up for discussion, but the team. Did something change? Were the specs unclear? Visibility into these dynamics can help spark productive conversations, facilitating knowledge transfer, and knowing where you can best support the team.

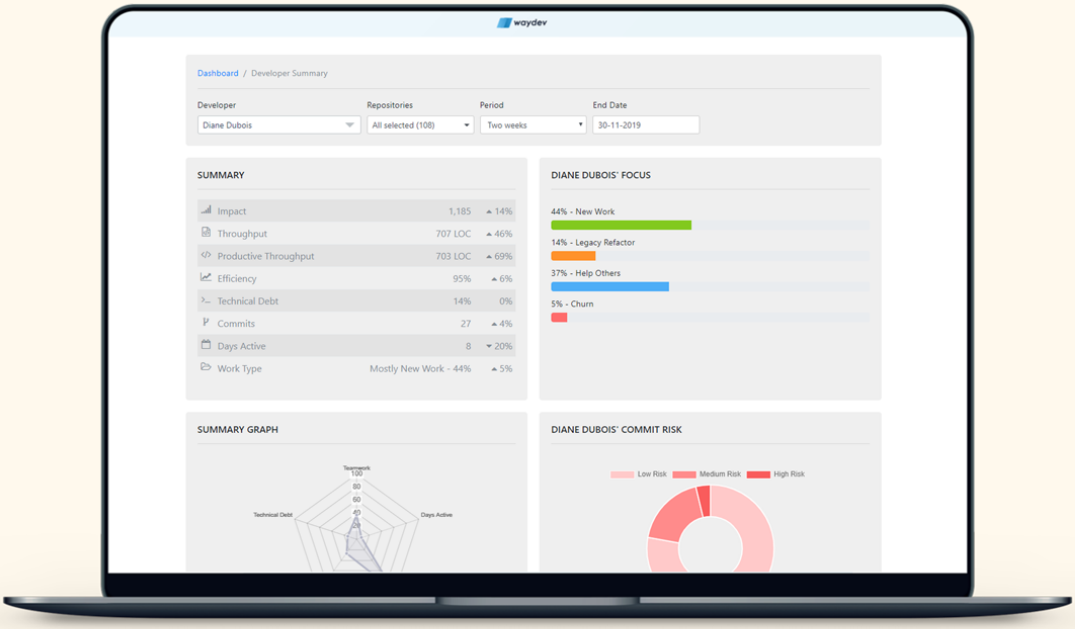
The goal with the daily update report is not for a manager to try and diagnose blockers on a daily basis as this is often too granular feedback from the manager. For most teams, this report is designed to provide your team with a quick, concise platform to communicate blockers at the code level that can easily be referenced and displayed so the team can support one another with solutions. Because few people feel comfortable enough to admit when they're stuck, especially when they're not quite sure if they're actually stuck yet. The data and the daily update report can help create a positive cultural shift towards transparency. Meanwhile, it can also help you get a feel for how the team works and be able to identify when something's off earlier before a team member gets stuck in a rut or before release is delayed.

Knowing how to incorporate these reports into your standups can help take the burden off your team to give you status updates consistently. It will help you identify where you can help earlier or know when there are broader systemic issues at hand. It'll help you identify opportunities for coaching, cross-training, and praise, so you can consistently grow a healthy and high performing engineering team. It can also help you schedule a block of time on your calendar before each standup, so you can get in the habit of reviewing these reports before the meeting and having them open on your device and ready for discussion.

One-on-ones

Patterns for communicating upward are almost the inverse of communication with your team. Team centric communication is often about fostering autonomy, and over communication at that layer can sometimes be distracting or can even come off as condescending. Upward communication is the opposite. Rarely will you run into a problem by over-communicating, and

the larger the company, the more updates serve as valuable reminders to your stakeholders about all pieces currently in play. Senior leaders need you to push information up to them, without being told, about what your team's focused on and how they're progressing. While you may not be speaking directly with senior leaders regularly, your manager may be, so it's helpful to consider them as a communication channel. One-on-ones can be a great place to showcase the work your team is doing.



When you're pushing information up, use data to support your narrative. By coupling data with your qualitative analysis, you can help your manager, nontechnical stakeholders, and senior leaders process the information more effectively. In doing so; in pushing visibility up while using concrete data, you're also simultaneously building trust and credibility for your team. When you support your team's narrative with concrete data, you can communicate to the organization before the release is delayed, so there are no surprises. You can show the consequences of ambiguous requests. You can advocate on behalf of specific team members, justify additional budget needs, and connect the dots between your team and the outcomes of the business. Let's dig into these lessons on running data-driven one-on-ones with your manager or senior leaders.

Let's say you want to make a bulletproof case for giving a key contributor on your team a raise and promotion at year-end. If you bring this ask your manager during your one-on-one and you don't have data to support the case, your manager is likely to ask you: ‘Great, why this person?’ It's a great question, but it's also a pretty loaded question. If we could instead, quickly and easily advocate for individual team members repeatedly, rather than trying to make a case all at once, it can help build rapport for each member on your team. So when you advocate for the promotion or raise for a team member, it's not a surprise for your manager, but rather an expectation. You can use the **Developer Summary** to showcase core individual metrics.

The **Developer Summary** shows an engineer's core metrics over the specified time period. It's specifically designed for developers to use in one-on-ones with team leads or managers. However, this report can also be valuable when managing up. Managers can bring these reports into one-on-ones with their managers to help them keep a pulse on how the team is doing and proactively make the team's work visible. Often, directors simply want to know that your team's output is relatively aligned with expectations and that you understand what motivates team members and are tailoring your leadership approach accordingly. You can start incorporating this report on a monthly cadence to help your manager know how things are going. If your organization is scaling fairly quickly, changing their workflows or otherwise, going through a time of change, you might consider giving these updates more often.

Some of the most common dynamics you'll want to showcase to your leader are the successful ramp of a new hire, how senior engineers are taking on even more expansive projects or mentoring more junior engineers and individuals that are embarking upon a major change in responsibilities or a type of solution they're building that is pushing their boundaries, and all the opportunities for you to advocate for your teammates in between.

Here's how you can keep your team members' achievements top of mind with your leadership. **First**, identify the team members that you want to focus your narratives around. In this case, let's call the engineer **Gary.** He has been on the team for a year and provides their peers a wealth of knowledge about the product and the code base, so that their teammates can excel. **Second**, tie their outputs to relevant business needs and objectives. In this example, I would communicate to my leader that this individual has the most domain knowledge on a relatively new team and is supporting their teammates to drive customers meaningful, maintainable solutions. **Third**, take notes on who you're talking about so you can remember the narrative you're telling. This will enable you to keep your story arc consistent and drive forward a compelling long term narrative around this individual's contributions. **Finally**, make the ask. This individual asked me last week for a pay raise since they were becoming a trusted advisor for their teammates, and they deserve it. Since I have kept my leader up to speed on this individual's growth in the past months, it will be no surprise to them when I asked that we extend their salary in the next review cycle.

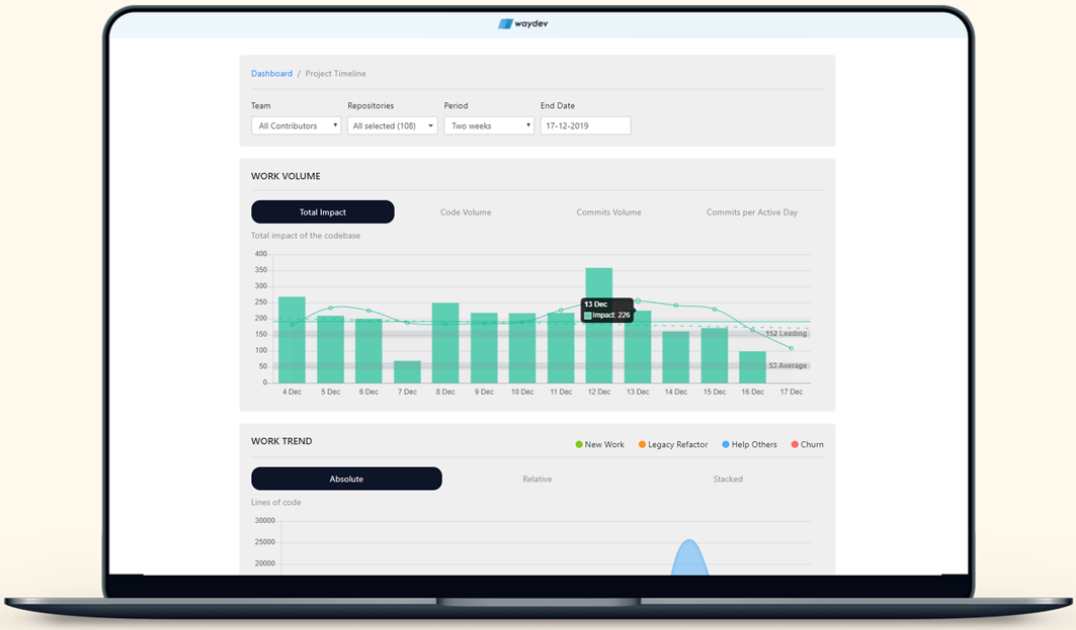
One of the most rewarding parts of management is the unique opportunity to further your team's growth, learning, and career. When you think it is time for one of your teammates to take on more responsibility and, of course, build the case for a corresponding increase in their pay, engineering managers are in a uniquely tough spot. Much of our advocacy for our teammates has had to be through stories and feelings. When we have gone to bat for one of our team members? How many of us can remember our leader retorting? How can we prove their contributions to the business? How do you know they've improved or even more frustrating? Now you have an incredible platform to leverage data as a conduit to prove this teammate's contribution to their team and strengthen the anecdotes you use to highlight the team member successes. Bring the **Developer Summary** to your one-on-one and focus on your team members' strengths.

A common conversation in one-on-ones between managers and directors is to discuss what the team is working on and how they're delivering relative to expectations. 'How's the team doing? What are they focusing on? How has additional headcount or process changes affected the team's outputs?' Those are all questions you might discuss from time to time. 'Then what can we expect from the team moving forward?' The overarching themes here are: 'Is the team

working on projects that are perceived as valuable, and how are they progressing relative to expectations?' To answer the question of how the team is doing, you need to start by knowing what the team is spending their time on. You know, the release that they're on and the goals of the sprint. But, do you know how much of each engineer's time is being spent on New Work, Legacy Refactoring, Help Others or Churn?

If your team is more focused on releasing new products or features, you'd probably expect around half of the work delivered or more to be **New Work**. Other engineering teams might be spending more time on **Legacy Refactoring**, which is defined as code that updates or edits at least 21-days old code. Legacy Refactoring is the process of paying down technical debt, which is traditionally more challenging to see. New feature development frequently implies the reworking of old code, but these activities are often not as clear until you're actually working in the codebase. You might include it in the scope of a project before you begin. Other times, the team might decide to focus an entire sprint or more on paying down **technical debt**, and this Legacy Refactoring is probably essential foundational work. Your business may not be able to achieve its goals in the next fiscal quarter if your team doesn't refactor old code.

That said, since it's more difficult to see and is less flashy than shipping new features, it can be harder to get buy-in for spending time on such projects. You have to help senior leaders see that time spent strategically on Legacy Refactoring this month can make it easier to ship New Work next month and going forward, but first, you have to show them where your team is spending their time. To help you and your manager both see what your team is working on, especially compared to what the business goals are for new features this quarter and the larger releases next year, you'll want to refer to the **Project Timeline**.



The **Project Timeline** shows you aggregate outputs. With it, you can see progress trends for the team, which can act a lot like a highlight reel. You can also show how much time is spent writing **new code** versus paying down **technical debt.**

Let’s say that a batch of new hires joined. Start with visualizing what the team is spending their time on. Here you can see the four work types. You can also see the time period in which the work was done, the total lines of code for each work type, and a list of your team members under each work type and which contributors contributed the most of each type of work focus. When you look at this data, you may want to filter to view a specific release cycle for the team. Is what you're seeing aligned with what you'd expect to see relative to prior periods and with the context of the team's current priorities? Is what they're primarily focusing on in line with the business objectives? When the team is focused on reworking older areas of the code base in order to pave the way for future releases, you can anticipate that senior leaders or external stakeholders may be curious as to what the engineering team is focusing on, since this type of work is not as visible to them.

If you change the code base output view from commit volume to total impact, you can say that actual impact to the code base is similar to the rest of the department, but what you can notice is that a lot of the department is focused on releasing new features, while another team is focused on cleaning up the code base for the rest of the department to benefit from. This doesn't necessarily translate immediately into new features, but it makes it possible for our team and other teams to build better features faster. Without this work, those new features simply would not be possible. Use these data points to help explain how your work on re-platforming your codebase will benefit the business moving forward.

On the other hand, if the product roadmap calls for new features to be developed in the near future, but you're clearly already in the midst of important refactoring, or your team is already stretched them, you may be able to leverage this data to make a case for adding additional resources or headcount, or adjust expectations and delivery dates to relieve the team. **Data like this makes the invisible work your team does visible to the people who need to understand it** - decision makers, executives, people who are responsible for approving budgets for new headcount. Without data, engineering doesn't get the recognition that other departments often get, and invisible work like Legacy Refactoring is a difficult communication

gap to bridge. Use the reports in the **Project Timeline** to create visibility around what engineering is working on, and regularly push that information up and out.

To improve your chances of getting additional resources, use data to build a before and after case for resource allocation. Data can also help you solidify your narrative, tell your team's story and broaden that story over time, so your manager understands why you're asking for resources, and how those resources will help the engineering team deliver items that contribute to the businesses objectives. The metrics that are most commonly used in these scenarios are **Impact** and **Efficiency. Impact** helps you speak towards the complexity of output into the product, while **Efficiency** speaks towards the survivability of that output.

Let's start with a common difficult scenario to handle as a manager. During a trying quarter this year for your business, or because of a shift in business strategy, the decision was made to retract budget from your team, and you weren't able to make the additional hires that you'd planned on adding. Your team needed a few new engineers with some rather specific skillsets in order to complete the product demands the business was asking of you. Without the hires, Your team was asked to work in areas that pushed beyond their expertise and both the team's morale and the team's ability to execute suffered. But now it's a quarter or two later and the business demands of your team to continue to grow, and you're ready to ask for that additional headcount again, but you don't just want to go to your director and try to rely on persuasion for new hires.

Look at the team's **Impact** for the quarter, following the misalignment between your team's skillsets and the skill sets required for you to deliver appropriately. If you notice a steep decline in your team’s impact, that is strong proof that the hunches about the deficit and skills and the business requirements at hand were affecting your team's ability to hit key milestones in getting our customers the software they need.

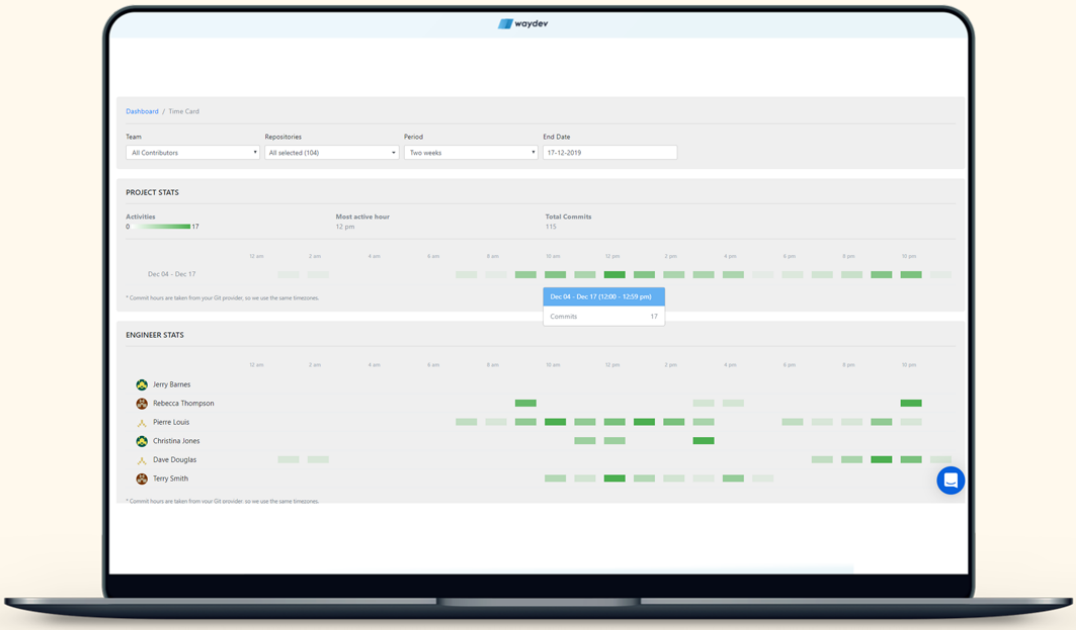
Bring the visualizations into your next headcount planning meeting and showcase to your leader that in order for you to deliver the products your customers need, your team needs to hire for that role that can fill these critical skill gaps and reverse the abrupt trend you see after the decision was made to retract your hiring plans earlier in the year. Measure any fluctuations in the team's output following that event in order to help bring the team back to the output it had before, at least one engineer will need to be hired.

When you can provide visualizations like these that are **easy to understand**, even for leaders who are far removed or nontechnical, it's much easier to align your team's needs with the business objectives and ensure your team is set up for success to deliver. So now that you've hired this individual and they're successfully ramping up, it's time to close the loop and demonstrate to your leadership the positive effect that this hire is having for your team. You can use impact and efficiency together to help tell this narrative. As a brief reminder, the **impact is the complexity of output**, whereas **efficiency is the survivability of that output**. So, let's say you see that the impact score went up 11% when you made the hire and efficiency for that same period stayed flat. What this indicates is that the increase in impact thanks to that new hire is a real change in output.

You can then tell your manager or executives that past hires were effective, allowing your team to do better work. When the data shows that the impact of business decisions or additions of engineers or resources directly impact the team's health and productivity, you help your senior leadership make more informed decisions about the health of their team's development. If you

are able to make the case to have a budget allocated to resource your needs, your objective will then be to show senior leaders that they made the right decision. That is, after hiring a new engineer, you'll want to measure the raw aggregate daily impact for your team in the **Project Timeline**. How did that hire impact your scores? How is it impacting the code base, the product, and the team? By keeping your senior leaders looped in on how investments in engineers directly lead to improvements in outputs, you're always making a case for maintaining your team and growing it to support every major initiative.

A one-on-one with your manager is the ideal time and place to demonstrate that your team is focused on the right areas and on delivering meaningful solutions. This is your chance to advocate for your team, not based on emotions, but clear quantitative data. If you want to advocate for various individuals on your team, show your manager their **Developer Summary** and demonstrate their progress as a contributor to the team relative to their career track. If you want to show off the work your team is doing and how that work is directly tied to product enhancements and business growth, go to the **Project Timeline** and show where exactly your team is spending their time. And if you want to make a case for getting additional resources or headcount, use the **Project Timeline** to directly map the work your team is doing and show your plan for how the additional hires or resources would be allocated. When communicating with executives or entering one-on-ones, bring data to help support your narrative and make visible what your team is working on.



Usually, engineers have the best output when working in big blocks of time. The **Time Card** helps you find out what is the golden hour in terms of productivity for your engineers and use this feature to schedule meetings outside their peak productivity time so you do not turn one-on-ones into a disruptive meeting.

One-on-ones for engineers

Whether you're leading two engineers or multiple teams of engineers, there may have been a time you wondered about this when sitting in a one-on-one with a direct report. Is this 30-minute meeting actually useful for this person? Is it time well spent? To date, maybe you've used one-on-ones as a time to build rapport, discuss specific reports, or mentor the individual on a particular challenge that they're facing, or maybe you've asked your report to guide the one on one by bringing a list of questions, concerns, and ideas to the meeting.

Perhaps your one-on-ones are a bit about career goals and skill development. Maybe you don't quite know what to do with your one-on-one time. You might've started chipping away at setting goals for one-on-ones. Those goals might include building trust between you and your report, creating a safe place to discuss feedback and personal matters and provide recognition, reflecting on sprints, sorting out stretch assignments and planning, career development, or reflecting on the engineer's happiness and their connectedness with the team and the organization, or even discussing product vision and direction.

Those are all great goals to have for your one-on-ones, but there's one goal on that list that's particularly important that engineering management doesn’t focus on.

It's this one: **reviewing the historical outputs an individual has contributed to their team and mapping that to their impact on the team's objectives and tying it back to expectations creates a sense of clarity that can boost motivation amongst the team.**

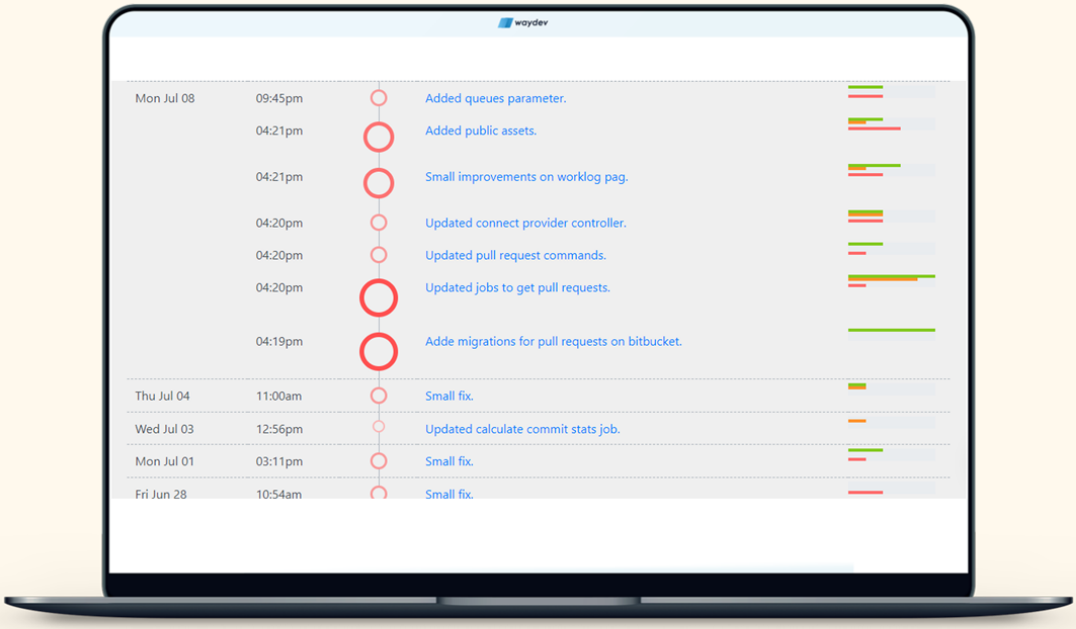
Being clear and consistent with expectations, encouraging team members' development, and recognizing good work these are elements of excellent management. So how do you review an engineer's historical outputs? A part of the job is quantitative review, finding **what** is going on exactly. The other part is qualitative, where the goal is to better understand the **why** behind the patterns and activities you're seeing. We will teach you how to use the analytics regarding the development process to get the full picture of the **what**, so you can use that information to get to the **why,** and have more targeted conversations in one-on-ones. Ultimately, the metrics we'll walk through should help you rely less on assumptions and hunches and be less susceptible to being swayed by unconscious biases. They should give you a starting point and a conduit to ask your team even better questions and have healthier and more productive discussions.

The **Developer Summary** was designed to support the engineer's desire to visualize their progress on core contributions to their team, but also for managers that wanted to easily pull up data to help inform even more actionable and specific one-on-ones and quarterly reviews. As you'll see, the **Developer Summary** shows you a condensed view of an individual's core metrics with trends and averages for an engineer's metrics relative to those of their team. It includes a view of your engineer’s output, which is particularly useful to help you understand the shape of an engineer's week or month across all work types. You'll get a stronger sense of how they perform in their code-level activities. Over time, this will help you and your team members visualize work patterns and progress over time, which can be useful information to drive conversations in one-on-ones.

With the **Developer Summary**, it's now possible to have a better understanding of what's going on before you enter a one-on-one so you can spend less time talking about **what**'s going on and

more time talking about **why**, which might shine a light on opportunities where you could help and when you're advocating for a specific individual on your team, you can leverage trends from the **Developer Summary** to tell a more compelling, proven story.

Because the **Developer Summary** is designed for engineers to use on their own and with you in one-on-ones, it may be helpful to dedicate your time in this next one on one, help each engineer understand what the data in this report helps show and get a feel for the different filtering options and views within the report.



Right off the bat, you can see that the first visualizations in this report show the engineer's core code fundamentals. You can also see summary stats, commit risk and a summary graph. If you scroll down, you will find a commits timeline, from which you can jump directly into your Git provider. When viewing this report, we want to visualize data in context with the team's trends and averages during that specified time period. In doing so, we're less susceptible to being reductionist about a single data point and are instead looking to understand the broader trends and look for deviations from those trends.

Quick looks at the stability of their impact and efficiency relative to time periods when they are working on things they're comfortable with will help us gauge the health of the transition and know when to stay out of the way or when to provide some assistance. With the **Developer Summary**, it's now possible for engineers to visualize their progress across core metrics, and for managers and team leads to quickly pull up a report that serves as a starting point for more productive conversations. One of the toughest and most important responsibilities in engineering management is asking great questions and communicating the actionable answers.

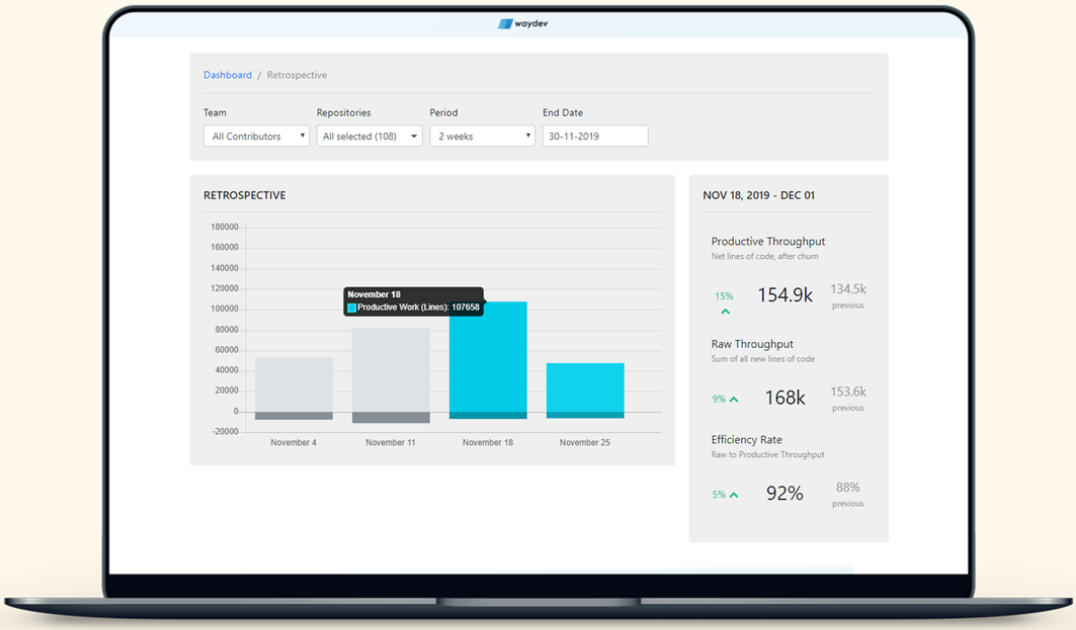
The **Developer Summary** is useful for triangulating different trends across an individual's metrics to help inform your questions before going into a one-on-one. It provides a unified view of how an engineer is working in a team and contributing to the code base. It's important

to remember that you're seeing an engineer's long term progress relative to that of their team, not the whole organization. In your one-on-ones, pull up the **Developer Summary** and swivel your monitors so you and the team member can see it and help them get comfortable with the data and how it can be used to visualize work patterns and recognize their wins. Then, as your teammates get more comfortable with the data, you'll likely begin to see a transition where the engineers will be coming to you with the data to tell their story better and to make more specific asks from you to enable them to do their best work.

By using the metrics from the **Developer Summary**, it's now possible to have a more precise understanding of what's going on before entering a one-on-one. As a leader, you should be thinking about people's day to day and their year to year, helping them find a trajectory that matches their goals, while also having an explicit framework for how people can grow at your company. Then incorporating those best practices and finding coaching opportunities and your weekly one-on-ones will help individuals envision their future and how you can help them get there.

Reports

Retrospectives are a time for the team to reflect on the previous sprint and discuss their successes and failures and collaborate on what actions they can take to improve the process and the product in the next iteration. Many teams' retrospectives take 45 minutes to an hour and take place at the end of the sprint, project, or other predetermined points in the production life cycle. A commonly used framework for facilitating team discussion in retrospective is start, stop, and keep. You might use this framework to drive a round table discussion where team members are asked to share approaches they would like to add to the team's process or start doing, such as specifying acceptance tests early with customers. When asked what they would stop doing, team members will express changes that would help the team save time or even help them find better solutions. This could mean a change in how the team interfaces with external stakeholders or the time when their daily standups are each day.



And the list of options your team might offer to keep could include anything that seems to be working but isn't quite habitual yet. This could consist of a particular way of communicating decisions or a mentorship program. The company has recently been incorporating, but it's not unusual for retrospectives to occasionally start to feel repetitious. Maybe you hear the same things, or perhaps everything seems to be going well. During those times, it can be tempting to put less effort into these ceremonies or even eliminate them altogether, at least for a couple of weeks. One of the primary reasons for these ceremonies is alignment, otherwise known as team cohesiveness. They're also helpful for noticing reoccurring bottlenecks or process issues. On a week to week basis, they might seem like one-off issues, but when you're in the weeds with work as individual contributors always are, it can be difficult to recall all of the challenges or past accomplishments from the past month or so.

Even more, some new team members may not feel comfortable yet to bring up issues, and others might be so used to business as usual, that is either hard to imagine what could or should start, stop, or continue. Thus, how do you continuously facilitate engaging the team discussions in retrospectives that result in incremental improvements to the process and the product? **Start incorporating data about what was shipped**, meaning the product, and **how it was shipped**, meaning the process. By bringing this information in, you can spend less time talking about what happened and more time talking about why and what the team would like to start, stop, or keep.

First, prepare by identifying the most relevant data points according to your team's workflows. Then gain buy-in from the team on what target ranges signal healthy development, and then use that information to facilitate a conversation about any deltas you see between your target ranges and your actuals, and have a productive discussion about what learnings could be applied to the next sprint, including what the team should stop, start and keep doing.

In this chapter, you'll see how to use data across the development life cycle as a conduit to enable your team to more readily present innovative ideas, advocate for progress, or speak more openly about the hard conversations — ultimately enabling your team to grow retrospective over retrospective.

Comparing your team's most recent sprint against the last one is a strong starting point and is also likely to introduce this vital question. **What do metrics typically look like for a team like mine?** Before digging into discussions with the team around data, most managers will do a little bit of exploration on where their team is at the moment and where the team could be in the future. **How do we compare to industry benchmarks? How do we compare to other teams that are similar in structure?** Comparing data from one team to the next can help you get an understanding of the team's baseline. Then you can experiment with the workflows and processes and ultimately keep a pulse on how things are trending relative to that baseline. **An important foundational step is to understand the best common starting point,** which is this: **Compare your team with like teams. A great place to start doing this is by using the Project Timeline.**

It will help you visualize how teams are doing relative to other teams, relative to industry benchmarks, and relative to your own trends, by using four metrics that customers find most actionable in code-based activity. **Commits per Active Day** represents the average number of commits per team per active day (a day when an engineer checks in code). **Commits Volume**, which quantifies the number of commits a team created in a day. **Total Impact**, which shows you the total impact of the codebase in a day and **Code Volume**, which represents the raw number of lines of code changed in the codebase. In the **Project Timeline**, you can visualize how those metrics are trending during the sprint or over the trailing week, month, quarter, 6-month period or a custom time frame.

This report is called **Project Timeline** because the four metrics you'll find at the top of the report are metrics for measuring code activity. We've talked about each one of the metrics already, but let's clarify what they mean so you'll know how to use them and talk about them in retrospectives. First, we have **Commits per Active Day**, which measures the average number of commits per active author per active day. If you consistently see that your team had several commits everyday, this means that they'd been working almost every day of the week, including weekends. The team may have too much on their plate, and it's time for some expectation, realignment upwards in the work funnel. This is worth making a note of before a retrospective begins so it can surface tactfully into your team discussion.

On the other hand, if the **Commits per Active Day** is unusually low for the team compared to prior periods, that might be a signal that something outside the normal delivery process is slowing them down. It could be due to untimely or unnecessary meetings, time spent waiting for code reviews, back and forth with indecisive stakeholders, or some other process or way of working that may be getting into the team's way. It could be worth using that trend as a starting point in a retrospective for the team to think about what they could start, stop, or keep that might benefit the team as a whole.

Then, there's **Total Impact.** You'll see this metric on a few different reports. It helps you and your team understand the general complexity of changes that your team is moving forward in your product.

Through a relatively simple metric like **Commits per Active Day**, you provide your team with a platform and safe space to voice innovative ideas on team efficiencies that may otherwise fall under the rug during the day to day workflows. At a glance, **Project Timeline** will help you and your team quickly see signals of process blockers affecting the health of your team's software development during conversations in your retrospective. **Commits per Active Day** will help your team identify whether there are any bottlenecks to moving work forward and understand your team's preference for frequent iteration of solutions within a day. As with all of the metrics that you leverage in the platform, use this data to generate an open conversation around team-led expectations versus actuals to provide a platform for innovative ideas that otherwise may have passed with the agenda items.

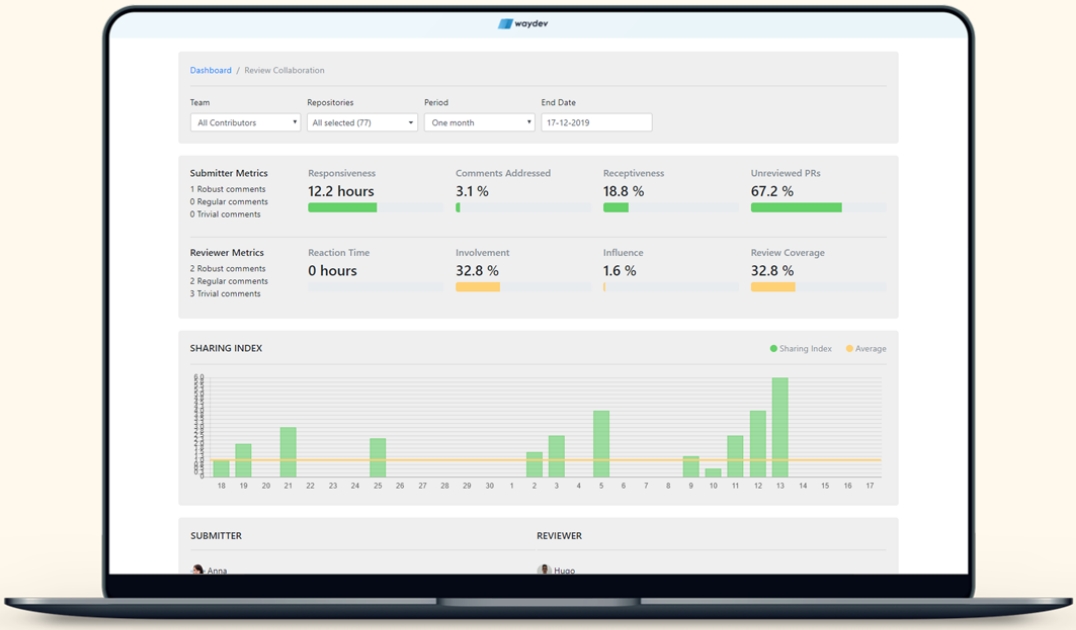
In order to visualize and evaluate release success and to compare sprints, bring up the **Retrospective** report. Depending on the size of your sprints, select a period and compare it to the previous to see how your team’s performance evolved. Spot outliers in the graphs and correlate them to events.

Code Review

As you dig into what took place during a sprint, a question that is likely to come up is

How long did it take for our team to complete the code review during this sprint?

For sprints where deadlines were missed or were during the sprint review, stakeholders felt there was not enough time to iterate to the correct solution. You want data to help drive a productive discussion about the length of time it takes your team to resolve pull requests. The time to resolve PRs ties directly to concepts that many engineering organizations identify with, such as creating smaller stories, providing quick, meaningful feedback, and iterate. And while agile scrum and other similar methodologies are designed to help engineers iterate swiftly and ship faster, undetected bottlenecks along the way can upset any development schedule. The **PR resolution** report can help you identify the bottlenecks in your PR cycles over the course of the sprint.



This report focuses on six core PR cycle metrics: Time to Resolve (a PR in hours), Time to First Comment (in hours), Number of Follow-on Commits, Number of reviewers on a PR, Number of reviewer comments on a PR, and the Average number of comments per reviewer. While these metrics can be a high level benchmarking, the **PR resolution** is designed to help you find outliers. You've probably already found that a lot of your PRs follow a similar behavior or path, so the primary thing to pay attention to is when something jumps out as noticeably out of the ordinary. This really helps us see the ground truth of our code review and can be a great visualization to incorporate into team retrospectives.

We often hear from customers that one of the biggest sources of pain and frustration in the delivery process is when an engineer opens a PR, and then they wait an enormous time before the reviewer picks it up.

Time to resolve, is the time that it takes for a pull request to be resolved, whether that means merging it, closing it, or withdrawing it. This metric looks at the broader picture of how work is being handled during the review process. The team can look at outliers, that is, the PRs that took an unusually long time to be resolved and work backward from there to figure out why those PRs, in particular. We'll often see common tendencies here: the PRs are large, maybe the PR was submitted late on a Friday. Ideally, you can bring the report into retrospectives with a few hypotheses to spark a discussion, but remember that this is a team's discussion about what patterns or trends are seeing and causing PRs to take longer to resolve.

The **PR resolution** report can help you see both the high-level team level dynamics and the underlying activities that can contribute to those dynamics. You'll be able to see how metrics like **Time to First Comment**, number of **Reviewers**, and number of **Follow-on Commits** are leading indicators of **Time to Resolve**, or how long it takes for your team's work to get through the review process to production. As with all metrics, your team surfaces in retrospectives, identifying the most relevant data points for your team's workflows, then gain buy-in with the

team on targets that signal healthy development. Facilitate conversations on the why between your teams target, and actual, then determine which actions, if any, to take. Start incorporating metrics from the **PR resolution** into your retrospectives, and coaching opportunities will be easier to identify. It'll be more obvious when processes need to be adjusted, and as a team, you can work towards a more collaborative, productive, and engaging code review process.

Most teams require code to go through a review process before it's merged. This process is intended to improve the quality of work before it goes into production, which reduces the risk of introducing bugs and to improve the knowledge transfer and collaboration within a team. In an ideal environment, everyone on the team is involved in the review process and collaborating to improve solutions to problems and ensure that the work meets the team standards. Reviewers are providing thoughtful and timely feedback, and submitters are responding to that feedback, engaging in discussion, and incorporating suggestions.

However, the process doesn't always go as planned. Unexpected tests land on people's desks, they get interrupted, and sometimes PRs get merged without ever having been reviewed. One key opportunity for discussion during a retrospective may be answering this question: ‘How can we drive down on reviewed PRs?’ Things come up, but when we have data, we can see the broader patterns and encourage specific behaviors.

Bringing this discussion up in a retrospective can drive awareness within the team, improve the work that's going into production, and ultimately enhance our knowledge sharing throughout the team. In the **Fundamentals** report. You'll see how the visualized trends in the review process. The **Fundamentals** help you understand how team members are handling their own work in the review process. If we're looking at the **PR submitter side** of the review as opposed to the reviewer side, one of the most important pieces of analysis this report is the percentage of your team's PRs that are merged without a review.

There are four submitter metrics. We have: Responsiveness, which means how long it takes for a submitter to respond to a comment in hours. There's the **Comments Addressed**, which shows you the percentage of comments that a submitter addresses versus those that go unaddressed. **Receptiveness** represents the submitters' tendency measured as a percentage to incorporate changes based on the feedback the team provides, and **Unreviewed PRs** shows what percentage of PRS are opened and merged without ever receiving a comment nor an approval. This is a fundamental metric for us to understand our general tolerance for the risk of solutions that we're moving forward to our customers. **The higher the rate of unreviewed PRs, the riskier solutions we are moving to customers.** One approach to incorporating this report into teams’ retrospectives, is being able to help them understand what these metrics are, and the narrative they collectively help us understand. Once a team is comfortable with the data, you can have a discussion together to come up with expectations around what the team would like to see in the report, whether the team sets specific targets or establishes target ranges. You can observe how the team is trending relative to previous sprints and relative to the industry. If metrics start deviating from the norm, this can be a great start for a discussion and a team retrospective about what the team could start, stop or keep.

The PR reviewer metrics help us to understand how team members are handling their teammates' work in the review process. As you can see, this report also has four core metrics. They're designed to balance the four metrics in the submitter fundamentals. The first metric is **Reaction Time**, which measures the time it takes for the reviewer to respond to a comment

addressed to them. **Involvement** looks at the percentage of pull requests that the reviewer participated in. **Influence** is a ratio of follow-on commits made after the reviewer commented, and **Review Coverage** is the percentage of hunks commented on by the reviewer. It helps us get an understanding of the thoroughness of a review between the submitter metrics and the reviewer metrics. You can get an understanding of how the team is leveraging the pull request process and how effective the process is. The code review process is intended to be a place where team members can work together to raise the quality of their work and learn from each other's approaches and techniques.

By bringing the reviewer and submitter fundamentals into retrospectives, teams can finally visualize how the activities in the review process contribute to the team's overall effectiveness. As with all metrics that your team surfaces in retrospectives, identify the most relevant data points for your team's workflows, then gain buy-in with the team on target ranges that signal healthy development. Facilitate conversations on the **why** between your team's target range and actual, then determine which actions, if any, to take. Regularly surfacing your key metrics in the reviewer and submitter fundamentals reports in the retrospectives, will help you raise the team's awareness around the importance of timely and thorough review.

Review Collaboration shows you the metrics from both the submitter and the reviewer sides of the PR process in a single unified view, and it includes a visualization of the distribution of review amongst the team. There are a few key narratives that we can gather from this report. One of those is the **speed of feedback.** Team members don't want to get stuck waiting on others for an answer or a response, and as managers, we generally want to help ensure that the team is getting what they need to move their work forward. The other side of that is **thoroughness, feedback, and actionability of the feedback.** Ideally, we can move toward finding a balance between responding in a timely way and also providing a substantive review. We can measure the timeliness of feedback and the thoroughness of feedback using four of the metrics that you'll find in the review collaboration report.

The first two of those metrics are responsiveness and reaction time. **Responsiveness** is a time it takes a submitter to respond to the feedback left on their PR, and **reaction time** is the exact same metric but flipped for the reviewer. That is reaction time tells you how long it takes for a reviewer to provide his or her peers with feedback. Both responsiveness and reaction time help you understand how quickly the team members are responding to and communicating with each other in reviews. When it comes to measuring the thoroughness of the feedback, we can look at unreviewed PRs and influence. **Unreviewed PRs** shows you the number of pull requests that are open and merged without ever getting a comment. While there are always edge cases, this never should be as close as possible to zero. You always want a second set of eyes on the code your customers will be interacting with.

Influence is a ratio of follow on commits made after the reviewer commented. When this metric is viewed across a longer period of time, it can provide some insight into the likelihood that reviewers' comments will lead to a follow on commit. Thus their influence. There may be individual team members that are more vocal or experienced that you would naturally expect at higher levels of influence amongst the team. So this metric can be particularly interesting in identifying those silent influencers. The four metrics again, our responsiveness, reaction time, unreviewed PRs, and influence. Together. These metrics help you see how quickly and

effectively your team responds to each other. Let's see how to find these metrics in the review collaboration report.

How does your team work together in the review process? How responsive or receptive are they, and how thorough or substantial is the feedback they are providing to each other? The review collaboration report helps to answer these questions. It provides a way for software teams to see the ground truth of what's happening in the code review process. You can find this report by hovering over the review and collaborate module and then clicking on the second report on the list. These eight metrics help teams look at how PR submitters are handling their own work in the review process and how reviewers are handling their teammates work in the review process and by breaking these dynamics down into eight specific behaviors, teams can get actionable insights that can lead to quick wins that will provide for a more healthy collaborative and productive code review process.

To understand the team's speed of feedback we can look at responsiveness, which is on the submitter side and reaction time on the reviewer side. Pull this report open in retrospectives with the time period set to the previous sprint or iteration and compare that to the sprint prior. When this median time trends towards 24 hours, meaning it's taking a full day for people to respond to each other, that can mean that there's a lot of time spent waiting, and the collaboration that is happening may not be timely enough to be effective, so it can be helpful to surface these metrics with the team In retrospectives and set expectations around response times together. Then track how changes in these metrics amount to changes in the team's time to resolve to understand a team's thoroughness of feedback.

We can look at unreviewed PRs in the submitter metrics and influence in the reviewer metrics. We generally want to see a low number of unreviewed PRs. High rates of unreviewed PRs increase the chances of introducing bugs and eliminates the opportunity for engineers to learn from their teammates. You will also want to look for a healthy balance of influence, which speaks to the likeliness that a reviewer's comments will lead to follow-on commits. We may be participating in code review, which will lower unreviewed PRs, but we also want to make sure that participation is meaningful, actionable feedback that drives improvements to the code and distributes knowledge between teammates. By bringing the review collaboration report to retrospectives, the team can, over time, foster a culture that values, and can effectively communicate the healthy tension between speed and thoroughness in code review.

A successful sprint is often measured by the completion of items in a sprint. That's the most basic measure of success. It's also about technical debt. You want to avoid introducing technical debt with reopening tickets, bugs spawns, quality errors, and more, via the newly published code. You and your team now have data at your disposal to monitor and diagnose any deviations in healthy collaboration, enabling you to deliver consistently and avoid introducing new technical debt. The patterns you and your team identify in a retrospective, the actions you agree to take, and the target ranges you'll set will all impact your next scrum ceremony.

With a well-run and productive retrospective, you can plan better sprints. You can identify developers that may need more assistance during the next sprint. You can make informed resource allocation calls, and you could ultimately help your team align toward common goals. Every minute spent in a data-driven retrospective brings your team closer to these recurring meetings, mission, alignment, and calibration. To run a more specific actionable retrospective, integrate the key metrics that you've identified as the most relevant for your team and set aside

time in the agenda for your team to surface conversations around these trends. The data will drive better discussions because your team will be able to offer their own anecdotal experiences as well as rich, real data.