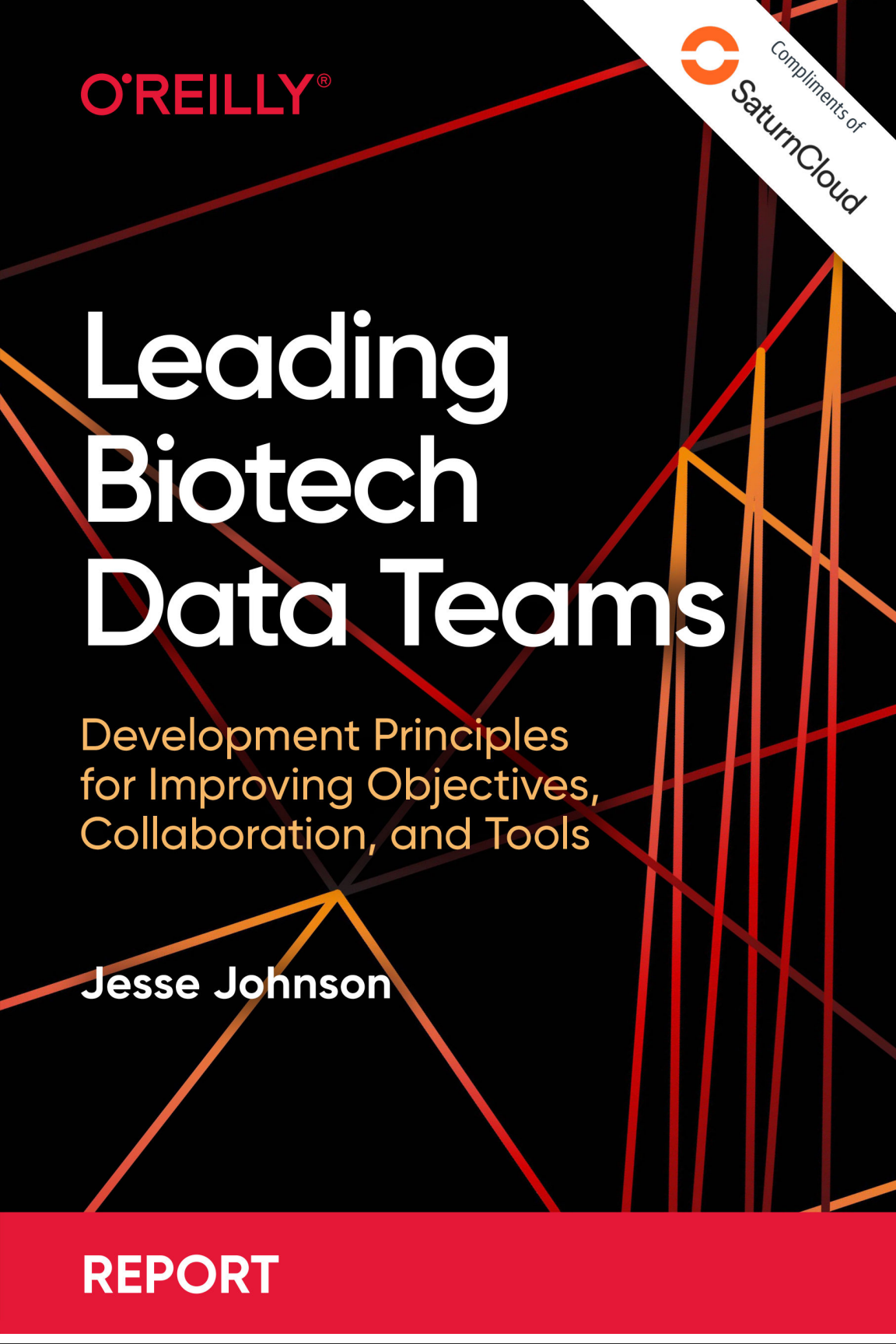
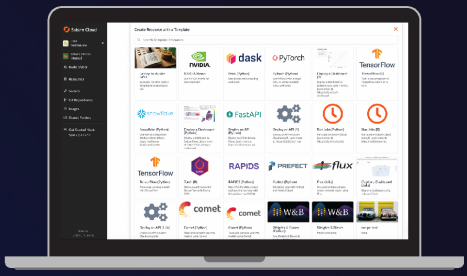

O'REILLY®
Compliments of
SaturnCloud
Leading Biotech Data Teams
Development Principles for Improving Objectives, Collaboration, and Tools
Jesse Johnson
REPORT

Saturn Cloud



Move to the cloud without having to switch tools.

Saturn Cloud runs compute with exactly the hardware, software, and Git repositories you want.

- Scalable data science with Python, R, and more
- Use up to 4TB RAM, large GPUs, and Dask clusters
- Cloud-hosted JupyterLab and RStudio
- Easily share work with colleagues
- Advanced security: SSO and installation into custom VPCs and private subnets
- Straightforward deployment environments
- Run jobs and deploy dashboards and APIs
- Connect from existing cloud resources
- Admin reporting tools

Trusted by

Mount Sinai · CUSTOM INK · DENALI · snowflake · NVIDIA · VITAL PROTEINS · brightline · AdventHealth · CFA Institute · checkout.com · Cellarity

Saturn Cloud

Get to work fast with the tools you prefer
and the compute you need.

saturncloud.io

Leading Biotech Data Teams

Development Principles for Improving Objectives, Collaboration, and Tools

Jesse Johnson

Beijing • Boston • Farnham • Sebastopol • Tokyo O'REILLY®

Leading Biotech Data Teams
by Jesse Johnson

Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

Acquisitions Editor: Michelle Smith
Development Editor: Virginia Wilson
Production Editor: Ashley Stussy
Copyeditor: Charles Roumeliotis

Interior Designer: David Futato
Cover Designer: Karen Montgomery
Illustrator: Kate Dullea

February 2023: First Edition

Revision History for the First Edition
2023-02-13: First Release

This work is part of a collaboration between O'Reilly and Saturn Cloud. See our statement of editorial independence.

978-1-098-14596-5

[LSI]

Table of Contents

Introduction

When I joined my first biotech startup as head of software engineering, after years in the tech world, I thought my job was to build an internal software platform so intuitive and powerful that it would revolutionize how the bench and data teams worked, both individually and together. But after months of working to build this vision, I realized I would first need to solve a bigger problem: how the organization fundamentally thought about experiments, predictions, and data. And a technology solution alone, no matter how brilliant, wasn't going to solve it.

Today's biotech organizations are increasingly adopting research programs that rely heavily on data. They sometimes call themselves tech biotechs or just techbio startups, but large pharmaceutical companies and everyone in between is joining in too. I'll use the term "data teams" to mean both the teams tasked with driving these new capabilities, whether they call it bioinformatics and computational biology or machine learning and data science, as well as the data and software engineering teams building the tools that make it all possible. This doesn't include explicitly service-oriented teams such as traditional IT that aren't expected to play a driving role.

NOTE

While the term "development" often has a very specific meaning in the context of biotech, referring to drug development or the second half of the R&D stage (the "D"), the term here is used more broadly to refer to the development of technical tools and resources such as analyses, pipelines, and analytical tools.

Like me when I joined my first biotech startup, the members of these data teams are used to solving problems by writing code, whether it's machine learning (ML) models, data pipelines, or digital tools for themselves and their colleagues. And like I did, they all eventually run into the same realization that their biggest problems won't be solved by any amount of code.

This has become a kind of mantra for me: *Bad software is a symptom, not a cause*. In other words, if your organization has a problem that seems to be technical, often in the form of slow, confusing, inaccurate, inadequate, and generally bad software, this is usually a symptom of some deeper underlying problem. It's processes and mindsets and all those fuzzy things that turn out to be much harder to fix. In particular, if you try to fix them with a technical solution, your users will just resist or keep using the lousy old tools that support their familiar ways of thinking and working.

Who This Report Is For

This report is for leaders of data teams working on early-stage biotech research who are ready to take on the challenge of helping their data teams, their bench scientists, and everyone in between work more effectively with data and digital tools.

Maybe you entered biotech from a tech background. Or maybe you became a data expert starting from a background in biology or chemistry. Either way, you were hired for this role because you understand the "tech" side of biotech. You know how to write code, build predictive models, and turn data into insights. You understand digital technology better than anyone in your organization. But the most important thing you know is how organizations can use data effectively. I want to teach you how to apply that knowledge.

In the pages that follow I focus on early-stage biotech research where there is the most flexibility to explore. Solving these problems in more highly regulated contexts requires a very different set of tools that I won't cover here. I will also focus on drug development startups because that's what I know. This approach may be effective for other types of biotech research organizations, but you'll have to decide that for yourself.

I don't assume the reader is familiar with any particular technology, but I will use some biotech-specific terminology such as the distinction between a bench scientist and a data scientist. And most importantly, I'll assume that you know what an organization that uses data effectively looks like and want to help make your own organization work that way too.

What You Will Learn

This report is not about technology or software or data. It's about how organizations interact with these things, how they can be nudged to interact with them more effectively, and the role that data teams can and should play in doing the nudging. And because these data teams need to nudge their colleagues just as much as they support them, I call this new approach Reciprocal Development. In particular, I present what I call the Reciprocal Development Principles, modeled on the Agile Manifesto but tailored to the context of data teams in biotech organizations. The goal of these principles is to reframe the way you think about your role, your team's role, and the tools at your disposal to fulfill that role.

After reading this report, you will begin to see the underlying process and organizational issues whose symptoms look like technology problems. You will begin to recognize how addressing these underlying causes will help you solve your technology problems. And the Reciprocal Development Principles will give you a toolbox to nudge your larger organization in the right direction.

A Need for Specificity

As a leader of a data team in a biotech organization, it isn't enough to build technical tools and systems that will support your organization. You need to build an organization that can use those systems to become more than the sum of its parts. The Reciprocal Development Principles will help you do this. But before I introduce them, it's important to understand why you can't just directly apply the Agile principles.

While the Agile Manifesto is now used in a wide range of contexts, some even beyond software development, the original authors wrote it to address a relatively narrow context—software teams writing applications for external clients—and in response to a specific

approach that was hindering them—the process-heavy, classical engineering approach to software development that is often called the Waterfall method. Many of the ideas in the Agile principles, such as favoring individual communication over process, are more broadly applicable and have been applied in a variety of settings. Others are more specific to the original context, such as measuring progress by working software.

Because of this specificity, the Agile Manifesto and the various tools and methodologies it has inspired have been the driving force in making modern software development teams more efficient and effective. To be effective in other contexts such as biotech, however, it's necessary to make changes to get back the specificity. That's what we're going to do here.

Each of the chapters that follows presents four of the Reciprocal Development Principles centered around a common theme, with the first two chapters building a solid foundation that enables more concrete practices in the third. The first chapter, "Defining Objectives," explores how to broaden the way your team views their work, shifting from purely technical objectives to organizational-level scientific objectives. The second chapter, "Building Collaborations," encourages your team to focus more of their energy on collaboration with partner teams, rather than guarding their time for technical work. The third chapter, "Deploying Tooling," explains how to build an effective development cycle by coordinating your team's work with the cadence of experiments and lab work.

Each of these chapters introduces the general theme, discusses each of the principles, then concludes with more specific suggestions for how to implement them. This is not a methodology like Scrum or Kanban with specific practices and rituals. You can follow the Reciprocal Development Principles using sprints or scrums or any other tools that fit your team's preferences. The important part is that you use the principles to change how your team thinks about their work and their role within the larger organization.

Let's take a look at them now.

The Reciprocal Development Principles

This report is organized around the following 12 principles, with each chapter covering four principles defining a common theme.

Defining Objectives

1. Your highest priority is to drive progress towards your organization's scientific objectives.
2. Design projects around deliberate scientific objectives coordinated with the overall organization.
3. The primary measure of progress is data-driven scientific discovery.
4. The simplest technical solution that will reliably meet scientific objectives should be chosen over complex or novel approaches with marginal improvements.

Building Collaboration

5. The most effective form of communication across functions and specialties is direct communications between individuals in each team.
6. Collaboration across teams is more important than technical excellence within any one team.
7. The key to successful collaboration is empathy for team members with different perspectives, priorities, and responsibilities.
8. Delegating decisions and accountability as far down as you can is the only way to continuously adapt to an unknown and changing environment.

Deploying Tooling

9. Information should be captured in a FAIR system as early as possible and anything derived from it should remain in a FAIR system.
10. Technical tools can only be effective when deployed in the context of good processes and communication patterns.
11. Evolve processes and tools incrementally, in parallel, based on continuous feedback, rather than introducing major changes all at once.
12. Development timelines should be deliberately coordinated with experiments to maximize opportunities for feedback.

In the chapters that follow, you'll learn how to begin using these principles to make your data team as effective and successful as you need them to be.

CHAPTER 1
Defining Objectives

One of the biggest failure modes for data teams in biotech organizations, particularly the junior members on these teams, is to view their role and their responsibilities in terms of a narrow, mostly technical scope. Once the members of your data team know what their ML model needs to predict or how their app needs to function, they want to get those things done and move on to the next technical project.

This makes sense in the context of a traditional tech company like Google, Meta, or Apple where the deliverables are primarily technical. But in a biotech organization where the overall objectives are very different, someone needs to make sure the data teams' technical objectives actually match up with the larger scientific goals. And while it would be great if the scientists could dictate their technical requirements precisely and accurately, they rarely have the knowledge and background to do it well. So the responsibility often falls to you to make sure objectives are aligned.

This is even more important in the context of a techbio organization where the tech is expected to play an equal (or greater) role relative to the bio. For these organizations, the tech side should not only contribute to the overall objectives, but help shape them. You and your team will have the deepest understanding of the capabilities that data and software have to power more ambitious scientific objectives. So if you focus exclusively on technical objectives, that's a missed opportunity.

The four Reciprocal Development Principles covered in this chapter address different aspects of how you and your team can begin to adopt a broader view of your scope and objectives. The final section of the chapter summarizes practical measures you can take to put these principles into practice.

Don't Stop at Technical Objectives

Principle 1: Your highest priority is to drive progress towards your organization's scientific objectives.

I'm going to start a bit abstractly with what you care about, what you pride yourself on accomplishing, and what you think fundamentally makes your work valuable. These are the things you prioritize because they define who you are.

If you come from a technical background, it's natural to frame these priorities in terms of technical excellence. You want to build the most accurate models and the fastest pipelines. You want to minimize technical debt and build intuitive interfaces, all while maintaining thorough documentation. If your scientist colleagues don't appreciate it, that's on them.

But when your priorities stop there, you abdicate the job of defining these technical goals and aligning them with the bigger picture. Maybe your predictive model answers an interesting (and publishable) question, but it ignores the less interesting yet more existential questions that would actually drive the pipeline. Maybe your dashboard gives bench scientists the information they asked for, but doesn't fit into their workflows in a way that they can use. If the model is accurate and the dashboard is deployed ahead of schedule, but the drug never makes it to the clinic, then the data and the bench teams have both failed.

If you narrow your scope to technical objectives, you push your data team towards a service role while allowing the bench teams to take sole responsibility for driving scientific goals. Forget about the equal footing between data and bench teams that was supposed to make your biotech company a techbio company.

Prioritizing the organization's scientific objectives rather than just your own technical objectives means taking responsibility for aligning those technical deliverables and requirements with the science. Every single person on your team, no matter how junior, should

be accountable for ensuring the work they're doing is what the organization needs, and that it's integrated into a coherent pipeline and platform. Every technical decision should be traceable through a deliberate and defensible sequence of "whys" to an organizational objective. You can't rely on someone else to do it for you.

Explicitly Define Project Goals

Principle 2: Design projects around deliberate scientific objectives coordinated with the overall organization.

The first principle was about how you think about your own role and priorities. This one is about how you communicate internally and externally, to reinforce this broader conception of scope.

Many teams don't deliberately write down project goals because they seem obvious: the goal is for the thing that you're building to work. But that's a technical goal, reinforcing a narrow scope of responsibility. To push yourself and your team to adopt a broader scope, and push your stakeholders to recognize this broader scope, you need to deliberately identify and communicate objectives, rather than let team members and stakeholders make assumptions.

To get from technical goals to broader scientific goals, start by asking why the technical goals were chosen in the first place. How will the information be used downstream? How are they doing it today? What would happen if you couldn't meet the requirements? What would happen if you could do better than expected? Find out what scientific goal your project is addressing, then make that your own objective.

If you're building a predictive model, the goal of the project isn't to create an accurate model. The goal is to improve whatever process or larger project is going to use the model. If the model is accurate but it doesn't improve the larger process/project for some other reason, the modeling project has failed. The same is true for building an app or a data pipeline, or anything else.

Even internal projects that don't directly address a specific scientific goal should have indirect impacts on the science. If you make your pipeline ten times faster and nobody notices, it probably wasn't worth the effort. But if someone does notice, and it makes it possible for them to use the predictions in a new way, then there's your scientific impact. If your code refactor eliminates 90% of your technical

debt, but you never touch the code again, then you could've just kept the technical debt. But if the refactor allows you to deliver your next update in half the time, enabling a new experiment, there's your impact.

Deliberately identifying these broader goals doesn't just help you communicate with stakeholders; it helps you prioritize your work and even identify shortcuts to meet the scientific goals faster. Can you answer the scientific question with a bar chart instead of the deep learning model that the scientists wanted? Are there easy questions the bench scientists didn't bother asking because they thought they were too hard?

Whether you repeatedly ask "why," map out downstream dependencies, or use any other method to identify the scientific objectives, deliberately writing down these goals forces you to think about that broader scope. This increases the chances you'll notice the issue that would make your 100% accurate model unusable, or realize that an afternoon of work could make it unnecessary. Even if you can't predict the exact impact of a project, you should be able to estimate it, or identify potential impacts. And if you can't come up with a compelling scientific impact then maybe you should put that project on hold until you can.

Technical Milestones Aren't Progress

Principle 3: The primary measure of progress is scientific discovery.

The third principle appears to be about metrics, but it's really about how you coordinate with bench teams. It's an intentional riff on the principle from the Agile Manifesto that working software is the only measure of progress. The Agile version was a response to early software development approaches in which separate teams would follow detailed specifications to separately write software components over the course of months or years before fitting them together at the very end.

These early development practices measured progress in terms of the separate components so they felt like they were making progress the whole time. But measuring it in terms of working software encourages you to put the components together as early as possible so you can show progress. And while this may seem to be a minor

difference, it turns out to have a huge impact by reducing the need for detailed specifications and the risk of missing issues that don't arise until everything comes together.

In the context of an embedded data team, it's best to measure your progress in a way that both pushes you to more effective development practices and aligns with your new, broader scientific objectives. Since you want your objectives to be about scientific progress, your measure of progress should be too.

Deploying your ML model isn't progress, but using it to design a better experiment is. Adding a table to the database isn't progress, but using it to explain an unexpected outcome is. Even analyzing data from an experiment isn't progress until it leads to understanding that drives a decision.

In the Agile context, the key insight from adopting a new metric was that you could put all the components of your system together before they're completely functional. This seemed crazy and scary at the time, but turned out to be one of the most impactful parts of Agile methodologies. In the Reciprocal context, the key insight may be similarly scary, but it's not that bad once you get used to it:

> You can often make as much or more scientific progress with a prototype or even a proof-of-concept as with an optimized, production-grade tool/model/application.

The failure mode when you don't realize this is to wait to try things in the lab until you're absolutely sure of your technical work. Even if you're getting regular feedback on your preliminary results, you may wait to release your model until you can make it a few percent more accurate or until you build a user interface for the scientists. You may wait for the bench scientists to ask for it instead of asking them to try it. You may be using Agile methods to ensure you have working software the whole time, but if you haven't created scientific impact, you haven't made progress.

If, on the other hand, you push yourself to make scientific progress by having the bench scientists use your tools as early as possible, you'll start to discover a new bag of tricks. Instead of making your model a few percent more accurate, do a sanity-test experiment in the lab to see if the current accuracy makes a difference. If it doesn't, then you've got some other work to do before improving the accuracy. Instead of making a user interface, manually run the

analysis for the first few experiments. It's more work but you'll learn a whole lot more about how your users want to interact with your tools.

By trying to get to scientific impact as soon as possible, not only will you identify potential issues before they become real problems, you'll also get a head start on building the processes and habits around the tools. The processes and habits that integrate your tools into the larger organization and drive scientific discovery.

You'll still get to make your model more accurate (if it's actually necessary) and build that user interface. Just not until you see that initial impact and progress.

Sometimes the Boring Solution Is the Best

Principle 4: The simplest technical solution that will reliably meet scientific objectives should be chosen over complex or novel approaches with marginal improvements.

So far, you've expanded your scope beyond just technical work, you've deliberately identified broad, scientific objectives, and you've started seeking scientific impact as early in the development process as possible. But if you thought that was difficult, just wait. Because to do those things well, you often need to select boring technical solutions that will get the job done sooner.

You were hired for this job because you understand industry best practices and you can build complex, advanced, sophisticated systems blindfolded. You probably have a list of tools and libraries and programming languages you've been wanting an excuse to learn. Not to mention all the things you want to be able to add to your resume.

Well, too bad.

Building a project in that new language you want to learn is just going to make it harder for the next person to maintain, not to mention how much longer it will take you to write it. Using the latest libraries and tools often means questionable documentation and undiscovered bugs.

Moreover, the most interesting tools and frameworks are often built for complexity and scale that is common in the tech sector but often overkill in a biotech lab. Think Kubernetes or Redshift. Your

application doesn't need to scale to billions of users if it's only for your bench team. Your pipeline doesn't need to be able to handle petabytes of data if your datasets are all measured in gigs.

If you do need to handle those sorts of technical requirements, or legitimately expect to in the future, then go ahead and design for that scale. But those tools generally come with overhead and complexity that aren't worth it otherwise. A relational database is much easier to use than a distributed data warehouse if you only have millions of rows. A microservices framework is going to be more headache than it's worth for a single server and a few dozen users.

At the end of the day, the goal is to get to a solution that is good enough to drive the science, and start driving it soon enough to have an impact before the bench team moves on to something else. If the cutting-edge technical solution is going to take twice as long to implement and three times as much effort to maintain, then it's time to choose the boring solution.

Putting It All Together

Taking responsibility for your organization's scientific objectives (Principle 1) requires you to both shift how you, your team, and your stakeholders think about the role of data teams, and to begin adopting specific development practices that may seem a bit uncomfortable at first.

Shifting how you and others view your team is a long-term process. Most importantly, you'll begin to deliberately talk about your scope and objectives as you want them to be, with the team and with stakeholders. This includes setting explicit project goals that address scientific rather than technical objectives (Principle 2). But you'll also push your team to think beyond technical objectives in their own work, and measure their progress accordingly (Principle 3). Plus you'll push your stakeholders to expect and support this broader scope.

The development practices that support this broader scope may take some getting used to, but as with the new practices that came with Agile, this new development approach will pay off in the long run. First, once you begin measuring progress in terms of scientific discovery rather than technical development (Principle 3), you'll

begin integrating your models and tools into experiments and wet lab processes as early as is feasible, rather than waiting until they feel ready. And to speed things up even more, you'll begin choosing boring solutions that match the scale you're faced with rather than the scale you might find at a tech company (Principle 4).

These practices are fundamentally different from the traditional role of data teams in biotech and the practices you may have learned in the tech sector, but they create a solid foundation for the rest of the Reciprocal Development Principles, and for your data teams to be successful.

CHAPTER 2
Building Collaborations

Now that you've broadened your team's scope and objectives, you'll need to adjust how you interact with the other teams in your organization to reflect your new role driving the organization's scientific objectives. The four Reciprocal Development Principles in this chapter will help you rethink how you interact with your colleagues, particularly those in the wet lab.

The Agile principles were developed in the context of software teams working with external users where the requirements are built around supporting what the user needs to do. The technical team doesn't ask the users to do things differently—collecting more data or redesigning experiments. And technical teams within the same organization as their users tend to be seen the same way—supporting the teams doing the real work, instead of driving it.

To drive the organization's scientific objectives on equal footing with your bench scientist colleagues, your data team needs to interact with them in a new way. Supporting your colleagues in their own objectives is part of that, but they need to support you too—generating the data you need and adjusting experiments to incorporate your results. You need to be willing and able to push them to work in ways that will make the overall organization more effective.

The four principles in this chapter will help you build this new kind of relationship with the other teams in your organization, and adjust your leadership to reflect the needs of this new arrangement.

Don't Make Yourself an Information Bottleneck

Principle 5: The most effective form of communication across functions and specialties is direct communications between individuals in each team.

When technical teams have a clear client/developer relationship with stakeholder teams, they also have clear expectations for information flow. Stakeholders communicate their requirements. Developers communicate their progress and deliver the final product. There may even be an intermediary like a product manager to direct the information flow. In practice, it's never quite this simple; developers may work with users to discover and clarify requirements, as well as to get feedback on progress. But this regulated information flow is the expectation.

It can be tempting to try and create this kind of communication pattern for your own team. In a fast-moving environment where everyone's figuring it out as they go along, many leaders try to control the chaos—to always know what's going on and actively make sure that everyone's understanding is consistent. But when the information is changing too fast for you to keep everyone up to date, this approach is actually counterproductive. All it does is make you an information bottleneck.

As a leader, your role will always include encouraging and facilitating communication between your team and the rest of the organization. But when you try to control it directly, you just reinforce a client/developer model. You may be pushing the other teams on the things you need them to change. You may be communicating broader objectives and scope. But if your team is just responding to requirements, even if you're the one communicating them, they're playing the role of developers working for a client.

To play an equal, active role and address a broader scope, every individual on your team needs to have an active relationship with colleagues in the larger organization. Not only is this necessary for them to keep up with changing information, but you also need their help to reinforce the other teams' understanding of this new role and to push them on the things they need to change.

But there's an even more important reason you need everyone on your team to be directly communicating with others: when they're working with their colleagues outside the data team, they will inevitably stumble across risks, opportunities, and issues that you would never have found yourself. In a well-established and stable field, you might be able to identify all these things from experience. But biotech isn't there yet, let alone techbio.

If you aren't used to this kind of communication pattern, it can be scary to let go of the information bottleneck. Instead of directly keeping information consistent and up to date across the different teams, you'll start coaching your team to communicate better. Instead of announcing information to the team, you'll start making sure everyone else has a chance to talk. These are new tricks you'll have to learn. But the clear, properly scoped objectives from the first four Reciprocal Development Principles will help keep everyone's understanding of the context consistent even if the details are in flux.

Stop Protecting Your Team from Communication

Principle 6: Collaboration across teams is more important than technical excellence within any one team.

There's another reason leaders of technical teams often try to manage the communication between their team and others: they want to protect their team's time for focus and technical work. This is a noble goal, and it feels like taking one for the team, saving them from the painful task of communication. Many of the members of your team probably appreciate it. But it's important to recognize that you're making a trade-off here, trading away context and details in favor of technical goals.

This trade makes sense in the client/developer model because the developer is removed from the context, so they're evaluated entirely on whether or not their technical work meets the technical requirements. Once they understand those requirements, any additional communication with users/clients is a waste. So protecting their focus time will help them succeed.

But in your new context where everyone on the team has objectives that reach beyond the technical, protecting your team's focus time

just increases the chance that their technical objectives won't match up with their broader goals to fundamentally change the way your organization drives towards scientific objectives. To do that, they need to bring everyone else along. They need to not only coordinate with bench teams and others, but to help these teams work in new ways. Without that, it doesn't matter if they meet their technical objectives.

Of course, you may have members of your team who expect and want you to protect their focus time. That's the fun part of their work, and the thing that they think will get them their next promotion, or their next job. But most of your data team went into biotech because they care about the science and the broader impact. Many of them have domain experience that they can apply. The more you protect their technical work time, the more you insulate them from the things that made them want to join your team over a (probably higher paying) tech team.

The point of this isn't to abandon all your team's technical work and standards. It's more about balancing the two, or better yet finding compromises that provide the best of both. Every individual can find the balance that works for them.

If you don't put enough energy into collaboration, even the best technical solution is going to fail every time. Replacing a few hours of your team's focus time to better understand the bench teams' problems could save them from having to redo a lot more work later. The more they see this, the more they'll be glad you aren't being overprotective of their technical time.

Cultivate Empathy by Focusing on Trade-offs

Principle 7: The key to successful collaboration is deliberate empathy for team members with different perspectives, priorities, and responsibilities.

Empathy is a core part of being human, but we sometimes limit our abilities to feel it by how we frame a situation. For our purposes here, I'm going to focus on a very narrow and tangible definition of empathy: the habit of assuming, when someone does something frustrating or seemingly nonsensical, that they have a good reason to do it that you're just not aware of.

When the bench team doesn't format their plate maps consistently, you might assume it's because they're too stubborn and disorganized. But if you cultivate empathy, you might suspect that their experiments are highly heterogeneous, they're already stretched thin with other priorities, they don't have experience with how to do it properly, and they don't realize the mess they're causing downstream.

If your scientist colleagues complain that they don't understand what you're working on or how it's supposed to help them, you might jump to the conclusion that they're afraid of change or being political. But if you cultivate empathy, you'll look for ways your own communication may be falling short.

When you face a frustrating situation, the temptation is always to make assumptions that protect your ego. But usually the assumptions that make you feel better in the moment actually block you from finding a solution. Every person and team has their own priorities, their own problems, and their own experiences that shape how they approach their work and what they can or can't do. If you can understand these circumstances you can address the root issue. When you forget or ignore this, you tend to write off the frustrating behavior as unnecessary resistance.

Deliberate empathy allows you to look past the initial frustration and search for the underlying cause. You ask questions. You view your colleagues as part of your team instead of an opposition. You work towards finding a mutual solution instead of a place to lay the blame.

In a client/developer model, the developer isn't asking the user to make major changes to how they work. The developer may roll their eyes, but they only need to understand the "why" of the users' processes well enough to support them. If your data team is defining goals and requirements on equal terms with their colleagues, they need to understand when to push back and when to change their own minds. And the only way to do that successfully is with empathy.

While the general concept of empathy is a deep and complex emotion, the particular type I'm discussing here is a habit that can be learned like any other habit. As a successful leader, you've probably already worked hard to develop it. Now your job is to help your team learn it.

Approaching collaboration with deliberate empathy means reminding yourself that everyone is facing a different set of constraints and their own list of priorities. It means asking the questions to figure out what those constraints and priorities are and incorporating them into your own planning. If you help your team recognize this, you'll see them discover that the issues they found so frustrating are actually the key to much deeper collaboration. And the more they do this, the more they'll fulfill the broader role and objectives that you've built.

Don't Make Yourself a Decision Bottleneck Either

Principle 8: Delegating autonomy, decisions, and accountability as far down as you can is the only way to continuously adapt to an unknown and changing environment.

At this point, you've started pushing your team to communicate more directly with their colleagues, protecting less of their focus time, and helping them to cultivate empathy. Now comes the hard part: delegating.

Most leaders will agree that delegation is important, but many are still bad at it. In particular, there's some subtlety to how you go about it. Delegating work and responsibility means handing off tasks to someone who can act as an extra set of hands, an extension of yourself. Delegating decisions and accountability means giving someone else the authority and resources to make sure the work happens, so they're answerable for the outcome, not just the work.

If you delegate responsibility, but not autonomy and accountability, you lose almost all the benefits of the last three Reciprocal Development Principles. If the members of your team are communicating directly with colleagues, giving up some of their focus time and cultivating empathy to understand root causes, they're going to have a good idea of what decisions need to be made. If they have to go up the chain to you to finalize all these decisions, then you're still a bottleneck. And if you're the one who's accountable for every outcome across your team, you're limiting both the number of objectives your team can pursue and your own bandwidth to think more strategically.

A model of leadership that emphasizes delegating accountability is fairly well established in the tech world. But biotech is heavily influenced by academic labs where lab heads tend to exert more control over the details and there are lots of student employees who need more direction. So to build effective collaborations between bench and data teams, you often need to be very deliberate about how you delegate and how you encourage other leaders to delegate.

Putting It All Together

For your team to take on a broader role addressing the scientific objectives you established based on the first four Reciprocal Development Principles, you need to shift the way your team behaves, and the way other teams think of you, away from the default client/developer model.

This starts with encouraging direct communication between members of your team and their colleagues across the organization (Principle 5) and shifting how you and members of your team think about the trade-off between technical focus time and communication (Principle 6). If you're used to playing a direct role in your team's communication, you'll have to learn some new tools and tricks to ensure that they don't drop the ball. But the harder part is often trusting your team to take care of this as you let go of the control you're used to. What you get in return is a team that feels empowered to take on the role that you need them to play.

As your team starts to spend more time working directly with members of other teams with different priorities and expectations, you'll need to help them cultivate deliberate empathy (Principle 7). This is something you can model when they come to you with frustrations. Ask them to think about why their colleagues might be reacting how they are. Explain the other teams' goals and assumptions. Acknowledge their frustration, but help them frame it as a frustrating situation rather than a frustrating colleague. Then brainstorm creative and collaborative ways to find a solution.

These first three principles will get your team ready to play an equal role in defining and driving the organization's scientific objectives. All that's left is to get out of their way by delegating accountability down the org chart, instead of just responsibility (Principle 8). Again, this may require a new set of tools and a leap of faith compared to how you're used to delegating. But the clear, high-level

objectives that you defined in the previous chapter make a big difference; if you're confident that your team understands the objectives and has the technical expertise they need, it's much easier to trust them with the outcome.

The Reciprocal Development Principles up to this point will give your data team a solid foundation to have the impact that you know it can have. The next four principles address the nuts and bolts of how they can put that foundation to work.

CHAPTER 3
Deploying Tooling

So far, the Reciprocal Development Principles have focused on high-level, more abstract ideas, namely expanding the scope of your team's objectives and improving how you communicate and coordinate with other teams, particularly in the wet lab. In this final chapter, we'll get a bit more practical and technical. But as you'll see in the following, the first eight principles provide a foundation that will allow you to effectively implement the final four.

In the first chapter, you established goals around scientific impact. But for the most part, your team won't make that impact directly—you'll do it through your colleagues on the science side by providing tools or predictions or something else. I'm going to use the shorthand "tools" for whatever this is, even if you don't think of it as a tool.

Building tools that the bench team can and will use is important. But the harder part is getting these users to use them effectively. That's because the way that users interact with tools is defined by a host of factors outside the code itself: the tasks and processes that it's used for. The habits and mental models that define how users think about their tools. The stories that these teams tell themselves, or hear from others, about the tools. I'm going to use the shorthand "processes" for all these things, even though it's much more complex than just processes.

As a technical leader of a technical team, you've been trained to think about tools. The last four Reciprocal Development Principles will help you switch your focus to include building processes that will ensure your tools are adopted and used effectively.

Collect Data as Far Upstream as Possible

Principle 9: Information should be captured in a FAIR system as early as possible and anything derived from it should remain in a FAIR system.

If you're in the target audience for this report, you've probably heard of the FAIR data standards. The name stands for Findable, Accessible, Interoperable, and Reusable. (See the GO FAIR website for details.) Even if you haven't heard of them, they probably reflect how you're already thinking about data. That's because the FAIR standards aren't meant to change how you do things; they're meant to help you explain it to your stakeholders.

On the surface, the FAIR principles seem to be more about tools than processes. But as with any tools, they only work if the processes that surround them are compatible. So instead of going into the details about FAIR data, I want to focus on the other two parts of Principle 9, which require carefully designed processes: (1) collecting data as early (or upstream) as possible and (2) keeping the data inside a central system once it's collected.

Most bench teams are used to working with Excel sheets saved on laptop hard drives, then collecting and sharing everything in a slide deck at the very end. Many Electronic Lab Notebooks (ELNs) and Lab Information Management Systems (LIMS) reinforce this habit. When these users need to analyze data that's already in a shared system, they pull it out, do what they need to in Excel, then upload it back (if you're lucky). But for your data team to work effectively, you need the data as soon as possible, in a form you can trust. Which means you need the bench teams to capture decisions and analysis as it happens (before they have time to forget) and in a consistent form.

As soon as a scientist decides to do an experiment, they should record the experiment design in a central location where they can update it as things change. As they do the experiment, they should be able to record their progress. And any data that their instruments

generate should be transferred to a central location with as little manual intervention as physically possible.

Whether or not your bench colleagues do this doesn't depend a whole lot on the tools. They could do it all in Excel, as long as they use consistent templates and store their files in a consistent, shared place. What makes it work are the deliberate processes and rules for how the data is collected and shared. On the other hand, if you build a well-designed database, a fast API, and an intuitive user interface, but they still wait until the end of the experiment to use it, then you haven't solved the problem.

In other words, this principle isn't so much about what you build as what you push your colleagues in the wet lab to do. If you've followed the first eight Reciprocal Development Principles, you should have a healthy, mutual relationship with the bench teams. You also have a clear list of scientific objectives that you can only achieve if the wet lab fixes their processes. So put that relationship to use by helping them to establish these processes and adopt the tools that will allow them to capture FAIR data as far upstream as possible.

Treat the Problem, Not the Symptom

Principle 10: Technical tools can only be effective when deployed in the context of good processes and communication patterns.

This is a more general restatement of a point I made in the discussion of Principle 9, but it's important enough to make it its own principle. Bad software is a symptom not a cause. A team can only use tools effectively if the ways they work and think about their work are effective. If you try to fix a team's problems by swapping out the software or other tools without addressing the processes around them, they'll just find a way to continue the ineffective processes with the new tools, or they won't use the new tools at all.

You may be tempted to just avoid the issue—if a technical solution won't solve the problem, then it's someone else's problem. But if that's the approach you choose then you haven't been reading the earlier chapters. Adopting broader, scientific objectives over immediate technical goals means that this is very much your problem.

There are two major reasons that bench teams, or teams in general, don't adopt better tools, both of which the first eight Reciprocal Development Principles have put you in a good position to address.

The first reason is that the current processes may align with their immediate priorities, or don't impact their priorities enough to be worth fixing. Often bench teams collect data in Excel sheets on their laptops instead of in databases because it's easy, flexible, and works well enough for what they want to do with the data. The same is true for how they design their experiments, do the analysis, and everything else. So before you can change their processes, you have to change their priorities. And the best way to do that is to use deliberate empathy to understand their priorities and constraints (Principle 7) then point to your broad, scientific objectives (Principle 1). If you can draw a direct line from those objectives to process changes in the wet lab, you can make an argument for shifting priorities.

The second reason is that they may not know a better way to design their processes, or they don't understand how those changes will address their priorities. This is where the communication model that you set up in Chapter 2 comes in. Everyone on your team can now practice empathy to understand their colleagues' priorities and constraints, identify ways to improve the processes based on those priorities, then communicate the changes and benefits. The key to Principle 10 is to recognize that that's now part of your teams' job as well as yours.

Whatever tools you're working on right now, or whatever you're going to work on in the future, don't just build it to reflect the existing process if that process is flawed. But also don't just build for some new, idealized process that is completely foreign to the target users. Instead, before you start building, work with your colleagues to identify processes that work for both your needs and theirs, then work with them to roll out the new process along with the tooling.

Evolve Process Along with Tooling

Principle 11: Evolve processes and tools incrementally, in parallel, based on continuous feedback, rather than introducing major changes all at once.

Now that you know you need to build processes, not just tools, you'll notice a chicken-and-egg problem: you can't introduce better tools if better processes aren't in place. But these better processes often require new tools, particularly software, to make them work. So the only way forward is to address both processes and tools at

the same time. This is where the incremental approach of Agile development can be very effective, if applied carefully. But there are some common ways you can fail, even while following the Agile principles.

The first is to incrementally develop a tool or system without trying to integrate it into ongoing processes until it's "ready." The longer you wait, the bigger the rollout and migration. This is more common with off-the-shelf software, particularly ELNs and LIMS, where it feels like you need to implement everything before anything will work. But it also happens with internally developed tools when teams push back the initial rollout so they can fix or add just a few more features.

Shifting processes is harder than updating tools, so the sooner you can get started the better. Moreover, the first-time users start working with a new tool, you'll immediately notice issues that you hadn't anticipated. The sooner you identify those issues, the easier it will be to fix them. So it's important to get tools into users hands as early as possible.

Often, this could mean sharing a prototype or manually handling steps that will be automated later. If you're building a pipeline that they will eventually access through a user interface, you might start by emailing them a spreadsheet. If your model isn't as accurate as you want it, ask them to try some of the predictions anyway. You may be surprised at what you learn, and you'll be glad you learned it sooner than later. (Note that I'm writing about early-stage research here. Later stages of development are much more regulated so this may not be possible.)

A common concern is that if you show users an early, buggy version of a tool, they may be unwilling to try it again later. But in my experience, that mostly happens when the tool isn't addressing a problem they care about, or isn't compatible with how they think about their work. If you build a good communication model and practice empathy, as in Chapter 2, you can set expectations appropriately and your colleagues will not only trust you to address their concerns in the next iteration, they'll ask for it.

The second failure mode is to iterate on the tooling without addressing the processes around it. Often data teams will assume that what the bench teams do is set in stone and not up for debate. But in practice, there are always aspects that can be adjusted, particularly

if you have an open discussion about trade-offs and objectives. The objectives you defined in Chapter 1 and the empathy your team developed in Chapter 2 are exactly the tools you need to have these conversations.

Coordinate Rollouts with Experiment Timelines

Principle 12: Development timelines should be deliberately coordinated with experiments to maximize opportunities for feedback.

At this point, we've built the foundation to begin shifting processes and inserting the tools to support it. But there's one last issue you're going to run into: when your tool is ready to try, you may find that you just missed the perfect opportunity.

Software and data teams, particularly those influenced by Agile, tend to define timelines by working forward based on estimates. They figure out what they want the final product to look like, decide the most logical steps to get there, estimate how long each step will take, then plot out what this looks like on the calendar. (This is if they write down a timeline at all.)

The problem if you do this on a biotech data team is that your bench colleagues work on their own, completely separate schedule and often have very long lead times. The next experiment that could use your work may already be too deep into planning to change course now. There may not be another suitable experiment for months.

Software teams that keep a release schedule deal with a similar problem internally: they have a deadline when the release "bus" will leave, and any feature that isn't ready by the deadline has to wait for the next release. So instead of saying it'll be done when it's done, they often split their work into smaller chunks to make sure they can get something onto the bus.

With experiment schedules, this is a bit more complicated because someone else controls the schedule, and it rarely has the regular cadence of software releases. But if you deliberately coordinate with the bench teams, you can find out when the next bus is leaving and make sure you have something ready for it.

Instead of planning forwards based on the technical work, you're now planning backwards based on when you can get feedback from the lab. As in the discussion of Principle 11, the version you build for the next bus may just be a prototype. It may involve manual steps. It may not be anywhere close to the Minimum Viable Product you would want it to be, but if it has an impact on the experiment and helps to make scientific progress, that doesn't matter. The most important thing is that you catch the bus.

Putting It All Together

The four principles in this chapter build off the foundation of the first eight Reciprocal Development Principles to help you design and implement the processes and tooling that will allow your team to have more impact.

First, to make sure that your team has data that you can rely on, you need to work with the bench teams to collect it as early as possible (Principle 9). While this may feel very distant from the broad scientific objectives you established in Chapter 1, it's the foundation for all the higher-level work that your team needs to do. It's also the canonical example of how any tools you build will only be effective if you can work with your bench colleagues to develop effective processes (Principle 10).

This is where the first eight Reciprocal Development Principles come in: draw a direct line from the goals you developed in Chapter 1 to the processes that you need the bench teams to adopt. Use the empathy your team developed in Chapter 2 to align these changes with your colleagues' priorities and limitations. Then work with them to iterate on their processes in parallel with the tools your team builds (Principle 11).

Finally, once you begin aligning your team's development timeline more closely with the wet lab, you'll notice that your opportunities to get feedback are often limited by the experiment schedule. So you'll need to start planning early and working backwards to coordinate tool rollouts with experiments (Principle 12). To make this work, you may need to start putting your tools in users' hands before you feel like they're ready. But remember: a prototype that makes an impact on your scientific goals is infinitely better than a production-grade tool that never gets used.

CHAPTER 4
Conclusion

You've now seen how the three areas covered by the Reciprocal Development Principles come together to help your data team have the impact you're looking for. The first four principles push you to expand your team's scope by adopting scientific objectives rather than limited technical goals. The next four help you build a coordination model with your bench colleagues that defines a mutual relationship in which you expect as much from them as they expect from you. The final four principles apply this foundation to develop effective processes, particularly within the bench team, that ensure that the tools your team builds will have the impact you're looking for.

The Reciprocal Development Principles push you to think about your scope and work in a way that you might not expect if you come from the tech world, or from traditional biotech IT. Instead of just supporting the rest of your organization, you'll begin pushing them in a better direction. But if you can make the shift, you'll find that you have a new toolbox that will allow you and your team to turn your overall organization into a data-driven biotech.

About the Author

Jesse Johnson is Vice President of Data Science and Data Engineering at Dewpoint Therapeutics, a drug development biotech startup founded in 2019 around a scientific field called biomolecular condensates. In this role, Jesse's diverse set of experiences from academic math departments, engineering teams at Google, and data science teams at large, medium and small life science companies provide a unique perspective on the ways that data and wet lab teams communicate differently, or sometimes don't communicate at all.

To better understand the nuances of this problem, Jesse has conducted interviews with dozens of biotech leaders in similar roles in other organizations. He developed the insights from these conversations and his own experience into a cohesive and relatively simple theory, which he explores in a weekly newsletter called *Scaling Biotech*.