MANAGING ORGANIZATIONS

Why Data Science Teams Need Generalists, Not Specialists

by Eric Colson

MARCH 08, 2019

HIROSHI WATANABE/GETTY IMAGES

In *The Wealth of Nations*, Adam Smith demonstrates how the division of labor is the chief source of productivity gains using the vivid example of a pin factory assembly line: "One [person] draws out the wire, another straights it, a third cuts it, a fourth points it, a fifth grinds it." With specialization oriented around function, each worker becomes highly skilled in a narrow task leading to process efficiencies. Output per worker increases many fold; the factory becomes extremely efficient at producing pins.

This division of labor by function is so ingrained in us even today that we are quick to organize our teams accordingly. Data science is no exception. An end-to-end algorithmic business capability requires many functions, and so companies usually create teams of specialists: research scientist, data engineers, machine learning engineers, causal inference scientists, and so on. Specialists' work is coordinated by a product manager, with hand-offs between the functions in a manner resembling the pin factory: "one person sources the data, another models it, a third implements it, a fourth measures it" and on and on.

Alas, we should not be optimizing our data science teams for productivity gains; that is what you do when you know what it is you're producing—pins or otherwise—and are merely seeking incremental efficiencies. The goal of assembly lines is execution. We know exactly what we want—pins in Smith's example, but one can think of any product or service in which the requirements fully describe all aspects of the product and its behavior. The role of the workers is then to execute on those requirements as efficiently as possible.

But the goal of data science is not to execute. Rather, the goal is to learn and develop profound new business capabilities. Algorithmic products and services like recommendations systems, client engagement bandits, style preference classification, size matching, fashion design systems, logistics optimizers, seasonal trend detection, and more **can't be designed up-front**. They need to be learned. There are no blueprints to follow; these are novel capabilities with inherent uncertainty. Coefficients, models, model types, hyper parameters, all the elements you'll need must be learned through experimentation, trial and error, and iteration. With pins, the learning and design are done up-front, before you make it. **With data science, you learn as you go, not before you go.**

In the pin factory, when learning comes first, we neither expect nor want the workers to improvise on any aspect the product, except to produce it more efficiently. Organizing by function makes sense since task specialization leads to process efficiencies and production consistency (no variations in the end product).

But when the product is still evolving and the goal is to learn, specialization hinders our goals in several ways:

1. It increases coordination costs. Those are the costs that accrue in time spent communicating, discussing, justifying, and prioritizing the work to be done. These costs scale super-linearly with the number of people involved. (As J. Richard Hackman taught us, the number of relationships (r) grows as a function number of members (n) per this equation: $r = (n^2-n) / 2$. And, each relationship bares some amount of coordination costs). When data scientists are organized by function, the many specialists needed at each step, and with each change, and each handoff, and so forth, make coordination costs high. For example, statistical modeling specialists who want to experiment with new features will have to coordinate with data engineers who augment the data sets *every time* they want to try something new. Similarly, every new model trained means the modeler will need someone to coordinate with for deployment. Coordination costs act as a tax on iteration, making it more difficult and expensive, and more likely to dissuade exploration. That can hamper learning.

2. It exacerbates wait time. Even more nefarious than coordination costs is the time that elapses between work. While coordination costs can typically be measured in hours—the time it takes to hold meetings, discussions, design reviews—wait-times are commonly measured in days or weeks or even months! Schedules of functional specialists are difficult to align as each specialist is bound to be allocated to several initiatives. A one-hour meeting to discuss changes may take weeks to line up. And, once aligned on the changes, the actual work itself also needs to be scheduled in the context of multiple other projects vying for specialists' time. Work like code changes or research that requires just a few hours or days to complete still may sit undone much longer before the resources are available. Until then, iteration and learning languish.

3. It narrows context. Division of labor can artificially limit learning by rewarding people for staying in their lane. For example, the research scientist who is relegated to stay within her function will focus her energy towards experimenting with different types algorithms: regression, neural nets, random forest, and so on. To be sure, good algorithm choices could lead to incremental improvements. But there is usually far more to gain from other activities like integrating new data sources. Similarly, she may develop a model that exhausts every bit of explanatory power inherent to the data. Yet, her biggest opportunity may lie in changing the objective function or relaxing certain constraints. This is hard to see or do when her job function is limited. Since the research scientist is specialized in optimizing algorithms, she's far less likely to pursue anything else, even when it carries outsized benefits.

Telling symptoms can surface when data science teams are run like pin factories, for example in simple status updates: "waiting on data pipeline changes" and "waiting on ML Eng resources" are common blockers. However, I believe the more insidious impact lies in what you don't hear, because you can't lament what you haven't yet learned. Perfect execution on requirements and complacency brought on by achieving process efficiencies can mask the difficult truth, that the organization is blissfully unaware on the valuable learning they are missing out on.

The solution to this problem is, of course, to get rid of the pin factory. In order to encourage learning and iteration, data science roles need to be made more general, with broad responsibilities agnostic to technical function. That is, organize the data scientists such that they are optimized to learn. This means hiring "full stack data scientists"—generalists—that can perform diverse functions: from conception to modeling to implementation to measurement. It's important to note that I am not suggesting that hiring full-stack data scientists results in fewer people overall. Rather, I am merely suggesting that when organized differently, their incentives are better aligned with learning vs. efficiency gains. For example, say you have a team of three creating three business capabilities. In the pin factory, each specialist will be one-third devoted to each capability, since no one else can do their job. In the full-stack, each generalist is completely devoted to a business capability, increasing scale and learning.

With fewer people to keep in the loop, coordination costs plummet. The generalist moves fluidly between functions, extending the data pipeline to add more data, trying new features in the model, deploying new versions to production for causal measurement, and repeating the steps as quickly as new ideas come to her. Of course, the generalist performs the different functions sequentially rather than in parallel—she is just one person after all. However, doing the work typically takes just a fraction of the wait-time it would take for another specialist resource to come available. So, iteration time goes down.

Our generalist may not be as adept as a specialist in any one function. But we are not seeking functional excellence or small incremental improvements. Rather, we seek to learn and discover all-new business capabilities with step-change impact. With full context for the holistic solution she sees opportunities that a narrow specialist won't. She has more ideas and tries more things. She fails more, too. However, the cost of failure is low and the benefits of learning are high. This asymmetry favors rapid iteration and rewards learning.

It is important to note that this amount of autonomy and diversity in skill granted to the full-stack data scientists depends greatly on the assumption of a solid data platform on which to work. A well-constructed data platform abstracts the data scientists from the complexities of containerization, distributed processing, automatic failover, and other advanced computer science concepts. In addition to abstraction, a robust data platform can provide seamless hooks into an experimentation infrastructure, automate monitoring and alerting, provide auto-scaling, and enable visualization of debugging output and algorithmic results. These components are designed and built by data platform engineers, but to be clear, there is not a hand-off from the data scientist to a data platform team. It's the data scientist that is responsible for all the code that is deployed to run on top of the platform.

I too was once lured to a function-based division of labor by the attraction of process efficiencies. But, through trial and error (there is no better way to learn) I've found that more generalized roles better facilitate learning and innovating, and provide the right kinds of scaling: to discover and build many more business capabilities than a specialist approach. (A more efficient way to learn about this approach to organization versus the trial and error I went through is to read Amy C. Edmondson's book "Teaming: How Organizations Learn, Innovate, and Compete in the Knowledge Economy").

There are some important considerations that may make this approach to organization more or less tenable in some companies. This process of iteration assumes low cost of trial and error. If the cost of error is high you may want to rethink (i.e., it is not advised for medical applications or manufacturing). In addition, if you are dealing with petabytes or exabytes of data, specialization in data engineering may be warranted. Similarly, if keeping a business capability online and available is more important than improving it, functional excellence may trump learning. Finally, the full-stack data science model relies on the assumption of great people. They are not unicorns; they can be found as well as made. But they are in high demand and it will require competitive compensation, strong company values, and interesting work to attract and retain them. Be sure your company culture can support this.

Even with all that said, I believe the full stack data scientist model provides a better starting place. Start with them, and then consciously (grudgingly) move toward a function-based division of labor only when clearly necessary.

There are other downsides to functional specialization. It can lead to loss of accountability and passion from the workers. Smith himself criticizes the division of labor, suggesting that it leads to the dulling of talent—that workers become ignorant and insular as their roles are confined to a few repetitive task. While specialization may provide process efficiencies it is less likely to inspire workers.

By contrast, generalist roles provide all the things that drive job satisfaction: autonomy, mastery, and purpose. Autonomy in that they are not dependent on someone else for success. Mastery in that they know the business capability from end-to-end. And, purpose in that they have a direct connection to the impact on the business they're making. If we succeed in getting people to be passionate about their work and making a big impact on the company, then the rest falls into place naturally.

Eric Colson is Chief Algorithms Officer at Stitch Fix. Prior to that he was Vice President of Data Science and Engineering at Netflix. @ericcolson

This article is about MANAGING ORGANIZATIONS

⊕ FOLLOW THIS TOPIC

Related Topics: ANALYTICS | TECHNOLOGY | LEADING TEAMS | TECHNOLOGY

Comments

Leave a Comment

POST

0 COMMENTS

JOIN THE CONVERSATION

POSTING GUIDELINES

We hope the conversations that take place on HBR.org will be energetic, constructive, and thought-provoking. To comment, readers must sign in or register. And to ensure the quality of the discussion, our moderating team will review all comments and may edit them for clarity, length, and relevance. Comments that are overly promotional, mean-spirited, or off-topic may be deleted per the moderators' judgment. All postings become the property of Harvard Business Publishing.