I’ve been employed in tech for years, but I’ve almost never worked

by Emmanuel

March 15, 2023 in **Tech**

When Twitter fired half of its employees in 2022, and most tech giants followed suit, I wasn’t surprised. In fact, I think little will change for those companies. After being employed in the tech sector for years, I have come to the conclusion that most people in tech don’t work. I don’t mean we don’t work hard; I mean we almost don’t work at all. Nada. Zilch. And when we do get to do some work, it often brings low added value to the company and its customers. All of this while being paid an amount of money some people wouldn’t even dream of.

What is happening right now in tech may be one of the greatest market inefficiencies—or even deceptions—in history. I am writing this article because I think outsiders deserve to know what's really going on in the field.

I know my statement may sound a little bit hyperbolic—how could people be consistently paid a lot to do close to nothing? Surely that can't be right! Well, let me share some examples from my own experience.

Five months ago, I was hired as a software developer by one of the world's most prestigious investment banks. While I prefer to do freelance work because it involves real work, I was looking to have a bit more stability for a while, so I gave a chance to a normal corporate technology job. Since the beginning of my employment, five months ago, I've worked for around three hours in total (not counting non-focused Zoom meetings which I attended without paying much attention).

When I first joined the company, I was excited. However, since I joined they've only given me tasks that were exceedingly easy to complete, in just a few minutes, but allocating days or even weeks to them. At first, I wanted to speed things along. I genuinely wanted to build a cool product, so I connected with people across the organization to ask questions about our intended users, their needs and how our product would satisfy them. But it was soon made clear to me, a few times, that I shouldn't do that. One person told me, "I don't want to tell you not to ask too many questions, but…" and she basically told me not to ask too many questions.

I soon realized that the project was overstaffed and most people were *pretending* to work. And I also realized that was the job I was hired to do; my job was to pretend. If this had been the only time this ever happened to me, I would consider it an anomaly. Unfortunately, this has been the case with almost every tech job I've had for years.

Consider the case of my previous tech job, in which I was hired as a data engineer for one the world's largest telecommunications companies. In the year and a half that I worked for them, there was only one two-week period during which I worked at full capacity. Other than that, I did almost nothing at all for the remaining 18 months. Outside those two productive weeks, most of my work involved attending irrelevant meetings, performing small tasks to pretend that a broken product worked well and even

generate fake results. It felt ethically compromising and it was boring; I only worked half an hour a week or so, non-focused meetings aside.

I could go on with many other similar personal stories, but you get the gist.

This isn't just me. All of the people I know who work in tech seem to be going through the same thing. One of my former colleagues, for example, told me all he does at work is watch Coursera courses. He's considering resigning after the company-sponsored Coursera subscription ends. Another former colleague was hired a year ago as a data scientist for a large oil company. She's making 200,000 pounds a year. All she does is prepare a PowerPoint presentation every week, and she's utterly bored.

Another one of my friends was hired two years ago as a quant for one of the world's most important investment banks (yes, that one). The interview process for that kind of job is among the hardest you can imagine—brain teasers, differential equations, graph algorithms. He was very excited at first, thinking he'd be building cutting-edge technology. However, while people always seem to appear busy from the outside, in reality he does almost nothing at all and is horribly bored but well paid.

It is hard to speak about this with others. Someone once told me that my frustration with the traditional tech workplace reminded him of Meghan Markle and Prince Harry because I kept complaining about stuff despite being in a highly desirable situation. Indeed, for a lot of people, the idea of being paid a lot to do nothing sounds like a dream come true. However, while we may not do almost any real work, we do have to constantly pretend that we do. That can be extremely frustrating and soul draining. Moreover, this shaky situation cannot last forever; it's like a poorly balanced house of cards. With the recent massive layoffs and the collapse of SVB, the signs of strain are already there.

Let's take a closer look at what's going on in tech and why techies end up working so little.

Task bloating

Businesspeople love predictability. They like to estimate in advance how much a project would cost and how long it would take. This can be surprisingly difficult in some disciplines. Even when building a physical thing, like a railway, projects often overrun significantly. London's latest train line, for instance, was delayed by two years two days before its inauguration. Imagine how much worse the situation gets

with intangible products like software, where it's hard to define exactly what it takes to deliver the product and even what the product is.

So, in the early 2000s, a philosophy called *agile,* which sought to improve the software development process, became really popular in tech. In agile, software is developed in very short cycles, as short as two weeks, and the result is validated and goals realigned between cycles.

Every couple of weeks, the team gathers to plan the tasks for the next cycle. But in this planning phase something really strange happens: It has become the standard practice to highly exaggerate the efforts required to perform a task, which we'll call *task bloating.* In the planning session, hands-on people, managers and other stakeholders all agree that each task takes an abnormally high number of hours to complete and requires an abnormally high number of employees.

We'll analyze the reasons for task bloating in a minute, but let me share a couple of examples so you can see the extent of the problem.

One of my recent tasks at the investment bank was to analyze what a couple of software code templates provided by Microsoft could be used for. Anyone familiar with software development would be able to do this in a couple of hours at most. However, in our planning session, it was collectively considered that this task required many days of work and two people. The difficulty of tasks is chosen by vote in this company and, in this case, the collective wisdom decided that the task was relatively hard. I voted that it was easy, which didn't match the majority vote. When they inquired about this, I conceded that maybe it was more difficult than I thought after all. It is hard to disagree in those situations because it seems you're going against the collective wisdom of the team. Also, when you notice that every single task is bloated this way and no one says anything, you don't want to challenge the system. As two of us were assigned to this task, we ended up splitting it into two even easier parts, one for each. I completed my part in a few minutes and pretended it took much longer.

In the following cycle, I was assigned the task of writing a paragraph summarizing the results of the previous task. I voted for a difficulty score of 1, the lowest possible, as it was a 10-minute task. However, my colleagues didn't agree; they voted that my task was harder than it looked and worthy of several days of work.

It is also customary to break up a simple task into a dozen pieces and then determine that each subtask is artificially difficult. For example, we had the task of deciding which software tool to use for a certain application. We had a list of features that were needed and four candidate tools, so we had to find out which of the tools covered more of the necessary features. This sounded to me like a simple task that could take one person a couple of days if done really thoroughly. But the task was broken up into four different subtasks to individually investigate each of the four candidate tools. Each of these four subtasks was assigned a window of many days to complete by two employees.

If I had to guess, I'd say a bloating factor of at least 10 is the norm. But 50 or even 100 isn't uncommon. By following the practice of task bloating almost universally, the bar of what a tech employee is expected to do with their time has been collectively lowered.

What causes this bloating? It is tempting to become cynical and think that this is a major conspiracy that helps lazy employees not work and lets companies overcharge their clients. But I think that, although there may be some truth in that, there are more profound causes. Let's have a look.

How the agile recipe kills productivity

The principles behind agile software development are commendable and much more suitable for the job than old-school ideas. So, many organizations have strived to "be agile." However, they've done so by adopting agile "recipes," which are step-by-step guidelines that are supposed to make a team agile. The most famous one is *scrum.* The adoption of a recipe results in a box-ticking exercise that makes companies believe they become agile just by strictly abiding by a set of inflexible rules. The effect is the opposite.

We already spoke about one of those rigid rules—that work must be divided into clearly defined cycles, usually called *sprints,* and that we must try to predict the difficulty of tasks to make sure they fit within a sprint. This structure encourages task bloating—employees want to deliver on the promises they make for the sprint, so they end up bloating the tasks to be sure to complete them within the sprint. So, productivity is sacrificed in the name of predictability.

The rigidity of sprints also creates other inefficiencies. For example, if a new employee joins the team in the middle of the sprint, more often than not they aren't assigned any work until the beginning of the

following sprint. So, due to glorification of the methodology, the new joiner is left with nothing to do. Everyone seems okay with it. Also, if someone announces they've completed an assigned task before the end of the sprint, in practice they are rarely assigned a new task until the beginning of the next sprint, leaving them completely idle, or they are assigned a filler task like "provide support to James on his task."

The agile recipe also manages the day-to-day work, creating even more inefficiencies. For example, it prescribes exactly how often the team should meet and for how long. The recipe says, for instance, that the entire team is supposed to meet every day for a short debrief called a *stand-up meeting.* In this meeting, every team member gives an update on the current tasks and potential blockers. The idea of a stand-up meeting is to create some sort of team synergy, but I've never seen that happen. It soon becomes a box-ticking exercise. Most often, employees use this time to tell the manager what they've done and don't communicate or listen to one another. It works for the manager, who benefits from being updated, but not for the others.

Some versions of the agile recipe also require employees to do a "demo" after they finish each task, in which they showcase to all other team members what they've done. And also, at the end of the sprint, all team members participate in a long joint session to do self-reflection, in which everyone speaks about everything that's been done.

As you can see, a lot of the work in tech involved "meta work" that seeks to plan or discuss the actual work: You first speak about the work you will do, then you speak about the work every day for two weeks as you do it (since tasks are really bloated, this requires a lot of pretending), then you demo the result of your work, then you collectively reflect on it. Very often, the meta work is almost the only work you do, as the 15-minute debriefs and planning sessions take longer than completing the bloated tasks themselves. I once attended a planning meeting in which it was discussed for 20 minutes whether a task should be included in the next sprint or not. The task could be done in five minutes.

I have joked that the reason Facebook became "Meta" is that all its employees do is "meta work."

The day-to-day inefficiencies introduced by the agile recipe have the effect of aggravating task bloating. For one thing, meta work ends up taking people's time. I once asked a lawyer friend who's

very busy whether he does daily catch-ups with the associates. He told me, “No. We’re too busy for that. We cannot afford to be constantly meeting to catch up, as we have important things to do.” He was surprised to hear we do that in tech. But the constant meta work also kills productivity by interrupting an employee’s flow. For example, when a coder faces a difficult task, sometimes the best solution is to spend a couple of days thinking about it or doing research in a focused, freestyle way. It doesn’t help having to constantly interrupt work to speak about work and notify everyone of what you’re doing every step of the way.

Unfortunately, agile has become a cult; some even call it a religion. If you suggest that the methodology may be causing unproductivity, its proponents double down and say it’s because you haven’t been strict enough in following the recipe or that you have misunderstood the recipe (they may hire an agile consultant to help out). One of my friends mockingly used to say that agile is like communism: the reason it’s never worked is that it’s never been applied correctly.

Questioning the agile recipe is seen as heresy. For example, I once suggested that I wasn’t sure all the recurring meetings to catch up were the best use of our time. I was told, “Well, in this company we enjoy teamwork. Don’t you?”

The universal adoption of agile recipes has hijacked techies’ work. No one is incentivized to work at full capacity or keep the product and client as the top priority. Instead, employees’ main goal becomes to abide by the methodology. Amen.

How hype destroys motivation

In the world of tech, hype is rife. When a new technology emerges, people get overly enthusiastic and want to apply it everywhere—we’ve seen it with big data, AI, data science, blockchain, ChatGPT, and so on. I’ve seen companies hire as many as 70 people to build a product with the goal of following a trend without defining what the product would do or whether clients would want it.

Just a few weeks ago, for example, a company reached out to me looking for advice on how they could use ChatGPT in their accounting software. They said this was because they were trying to raise

funding, and investors wouldn't like it if they weren't using ChatGPT for something. I've documented many examples of this phenomenon in the context of AI in my book.

This cart-before-horse approach makes companies overstaff teams to build unnecessary products. I think this is another major reason for the task bloating and perpetual thumb twiddling that I mentioned above, as techies become severely demotivated when they see no point in their work. One of my friends lost all interest in his job when an additional five people were hired for his team because the topic was deemed trendy by an upper manager, even though they were coping just fine with the workload. The expansion of the team diluted the work among too many cooks, so task bloating season started.

In some cases, employees soon realize the product they're working on will never be used by real clients as it never really responded to a client's need—it just followed a fad. But they still have to build the product. In those cases, why bother working hard instead of bloating the tasks and using the time for something more meaningful like searching for the next job, mowing the lawn or perhaps writing this article?

I once joined a team to build a product that was meant to help data scientists within the organization. As I'd worked as a data scientist for years, I was a bit surprised when I saw the product's user interface, which didn't seem to reflect the way data scientists really work. I enquired about this, and found out they hadn't even interviewed data scientists across the organization to see what they needed—they'd blindly created the team and started the development of the product because it seemed trendy. When I inquired further, an upper manager told me, "We have decided not to speak with users; we will first build the product we think is right for them and see what they think." He called it a "fail fast approach," but I thought it was a "fail for sure" approach. Imagine the level of motivation and productivity in this team.

Why does this happen so much in tech?

Tech is not the only sector riddled with unproductivity. I'm sure some people would object that what I've said so far also plagues other industries. However, I think some characteristics of tech make it particularly prone to task bloating and unproductivity. I think one of the reasons for that is that technical

work is poorly understood by business people. So, if techies tell a decisionmaker that reading a code template takes a week, instead of the more realistic 30 minutes, they are likely to believe it. It would be much harder for, say, a barber, to pretend that cutting someone's hair takes days, as everyone has had a haircut before and roughly understands what a haircut entails.

Moreover, the benefits of many tech projects are intangible. If you promise to build an "AI-based insights engine," it is hard for businesspeople to understand the return on investment. You can roughly measure the output of a barber in terms of number of haircuts, but how do you measure the value of a data science or an analytics project?

The adoption of agile recipes also helps conceal the problem, as it creates the illusion of agility and the illusion of teamwork. If you look at it from the outside, the team seems productive because every couple of weeks new tasks are completed and other tasks are collectively planned. It also seems that there is a lot of teamwork, as everyone meets all the time.

These factors combined have helped tech companies become unbelievably bloated without consequences. If you haven't yet, you may want to check out this video, called "One day in the life of a Twitter employee."

What does truly agile teamwork look like?

The title of this article says I've almost never worked when *employed* in tech. But that doesn't mean I've never worked. In fact, I've worked really hard and been part of amazingly productive teams which delivered top-notch products in record time (and clients loved them). The thing is, this hasn't happened in a traditional environment, that is, when employed full time by a mid-sized or large tech company. It was only when working as a freelancer or for early-stage start-ups that I've witnessed productive tech work. I'd like to share why I think that happened, as I hope some of the features could be exported to the broader tech industry.

The most common characteristic I see behind truly productive tech work is that the people involved prioritize building the right product above everything else. They build it in small cycles and validate it, which follows the agile philosophy, but they don't limit themselves to a strict agile recipe. They sometimes borrow some elements of an agile recipe, but only because it helps them build the right

product. For example, they have recurring meetings only if there's a good justification for them. And, instead of letting the methodology dictate every single interaction, they hold impromptu meetings on an as-needed basis, even at five minute's notice, all in the name of improving the product.

Also, in a company that prioritizes building the right product and serving its clients, employees outright refuse building something they consider inadequate just because the boss said it was a good idea. For example, in a start-up I worked for, two software developers and the CTO once had an intense verbal fight in the office about aspects of the product, and one of them ended up crying, with tears and all. I know that might be a bit too much, but it shows that they all cared passionately about the product and didn't just do the assigned tasks.

This contrasts dramatically with how things work in the world of the traditional tech enterprise, in which the employees' focus is all too often on doing their tasks as requested in the allotted time frame without questioning their business value and while following the methodology to the letter. In the book *The Stupidity Paradox,* Mats Alvesson and André Spicer describe this phenomenon, saying, "You avoid thinking too much about exactly what you are doing, why you are doing it, and its potential implications. You hope to avoid punishments and many worries that might come from deviation. You sidestep the burdens of having to think too much and upsetting others by asking difficult questions."

It seems to me that most tech companies optimize for the wrong thing. They optimize for compliance with methodologies, for the adoption of trendy technology, for having a billion-dollar valuation, and so on, but they've forgotten about their clients. However, thinking about clients is what makes a business successful and a team truly productive.

A techy friend of mine told me today, "I'm not sure whether I should be pleased to save so much money and apply myself so little. It's a trap really, as in this age of cost-of-living crisis it feels like a no-brainer to have access to this amount of savings. But shouldn't I be striving to improve my skills for the future?" Scared to switch jobs, she's clinging to her current one in which she is paid a lot, does very little, and learns very little. So is the case with many tech workers. If you are one of them, I encourage you to look for a new job as soon as you can; I hear there are lots of companies out there looking to build real stuff but struggling to find tech talent.

About the author

Emmanuel

Emmanuel Maggiori, PhD, is a software engineer who helps companies build challenging software products from scratch. He's the author of Smart Until It's Dumb: Why artificial intelligence keeps making epic mistakes (and why the AI bubble will burst).

LEAVE A REPLY

Your email address will not be published. Required fields are marked

Comment

Name *

Email *

Website

Save my name, email, and website in this browser for the next time I comment.

Post Comment